International Handbooks on Information Systems

Christoph Schwindt
Jürgen Zimmermann *Editors*

Handbook on Project Management and Scheduling Vol. 1

Springer

International Handbooks on Information Systems

Series Editors

Peter Bernus, Jacek Błażewicz, Günter J. Schmidt, Michael J. Shaw

Titles in the Series

M. Shaw, R. Blanning, T. Strader and
A. Whinston (Eds.)
Handbook on Electronic Commerce
ISBN 978-3-540-65882-1

J. Błażewicz, K. Ecker, B. Plateau and
D. Trystram (Eds.)
Handbook on Parallel and Distributed Processing
ISBN 978-3-540-66441-3

H.H. Adelsberger, Kinshuk,
J.M. Pawlowski and D. Sampson (Eds.)
Handbook on Information Technologies for Education and Training
ISBN 978-3-540-74154-1, 2nd Edition

C.W. Holsapple (Ed.)
Handbook on Knowledge Management 1
Knowledge Matters
ISBN 978-3-540-43527-3
Handbook on Knowledge Management 2
Knowledge Directions
ISBN 978-3-540-43848-9

J. Błażewicz, W. Kubiak, I. Morzy and
M. Rusinkiewicz (Eds.)
Handbook on Data Management in Information Systems
ISBN 978-3-540-43893-9

P. Bernus, P. Nemes and G. Schmidt (Eds.)
Handbook on Enterprise Architecture
ISBN 978-3-540-00343-4

S. Staab and R. Studer (Eds.)
Handbook on Ontologies
ISBN 978-3-540-70999-2, 2nd Edition

S.O. Kimbrough and D.J. Wu (Eds.)
Formal Modelling in Electronic Commerce
ISBN 978-3-540-21431-1

P. Bernus, K. Merlins and G. Schmidt (Eds.)
Handbook on Architectures of Information Systems
ISBN 978-3-540-25472-0, 2nd Edition

S. Kirn, O. Herzog, P. Lockemann
and O. Spaniol (Eds.)
Multiagent Engineering
ISBN 978-3-540-31406-6

J. Błażewicz, K. Ecker, E. Pesch,
G. Schmidt and J. Węglarz (Eds.)
Handbook on Scheduling
ISBN 978-3-540-28046-0

F. Burstein and C.W. Holsapple (Eds.)
Handbook on Decision Support Systems 1
ISBN 978-3-540-48712-8

F. Burstein and C.W. Holsapple (Eds.)
Handbook on Decision Support Systems 2
ISBN 978-3-540-48715-9

D. Seese, Ch. Weinhardt and
F. Schlottmann (Eds.)
Handbook on Information Technology in Finance
ISBN 978-3-540-49486-7

T.C. Edwin Cheng and
Tsan-Ming Choi (Eds.)
Innovative Quick Response Programs in Logistics and Supply Chain Management
ISBN 978-3-642-04312-3

J. vom Brocke and M. Rosemann (Eds.)
Handbook on Business Process Management 1
ISBN 978-3-642-00415-5
Handbook on Business Process Management 2
ISBN 978-3-642-01981-4

T.-M. Choi and T.C. Edwin Cheng
Supply Chain Coordination under Uncertainty
ISBN 978-3-642-19256-2

C. Schwindt and J. Zimmermann (Eds.)
Handbook on Project Management and Scheduling Vol. 1
ISBN 978-3-319-05442-1
Handbook on Project Management and Scheduling Vol. 2
ISBN 978-3-319-05914-3

More information about this series at
http://www.springer.com/series/3795

Christoph Schwindt • Jürgen Zimmermann
Editors

Handbook on Project Management and Scheduling Vol. 1

Springer

Editors

Christoph Schwindt
Institute of Management and Economics
Clausthal University of Technology
Clausthal-Zellerfeld
Germany

Jürgen Zimmermann
Institute of Management and Economics
Clausthal University of Technology
Clausthal-Zellerfeld
Germany

ISBN 978-3-319-05442-1 ISBN 978-3-319-05443-8 (eBook)
DOI 10.1007/978-3-319-05443-8
Springer Cham Heidelberg New York Dordrecht London

Library of Congress Control Number: 2014957172

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.
While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

This handbook is devoted to scientific approaches to the management and scheduling of projects. Due to their practical relevance, project management and scheduling have been important subjects of inquiry since the early days of Management Science and Operations Research and remain an active and vibrant field of study. The handbook is meant to provide an overview of some of the most active current areas of research. Each chapter has been written by well-recognized scholars, who have made original contributions to their topic. The handbook covers both theoretical concepts and a wide range of applications. For our general readers, we give a brief introduction to elements of project management and scheduling in the first chapter, where we also survey the contents of this book. We believe that the handbook will be a valuable and comprehensive reference to researchers and practitioners in project management and scheduling and hope that it might stimulate further research in this exciting and practically important field.

Short-listing and selecting the contributions to this handbook and working with more than one hundred authors have been a challenging and rewarding experience for us. We are grateful to Günter Schmidt, who invited us to edit these volumes. Our deep thanks go to all authors involved in this project, who have invested their time and expertise in presenting their perspectives on project management and scheduling topics. Moreover, we express our gratitude to our collaborators Tobias Paetz, Carsten Ehrenberg, Alexander Franz, Anja Heßler, Isabel Holzberger, Michael Krause, Stefan Kreter, Marco Schulze, Matthias Walter, and Illa Weiss, who helped us to review the chapters and to unify the notations. Finally, we are pleased to offer special thanks to our publisher Springer and the Senior Editor Business, Operations Research & Information Systems Christian Rauscher for their patience and continuing support.

Clausthal-Zellerfeld, Germany

Christoph Schwindt
Jürgen Zimmermann

Contents

Part I The Resource-Constrained Project Scheduling Problem

Part II The Resource-Constrained Project Scheduling Problem with Generalized Precedence Relations

Part III Alternative Resource Constraints in Project Scheduling

Part IV Preemptive Project Scheduling

Part V Non-Regular Objectives in Project Scheduling

Part VI Multi-Criteria Objectives in Project Scheduling

Part VII Multi-Mode Project Scheduling Problems

Part VIII Project Staffing and Scheduling Problems

Part IX Discrete Time-Cost Tradeoff Problems

Contents of Volume 2

Part X Multi-Project Scheduling

Part XI Project Portfolio Selection Problems

Part XII Stochastic Project Scheduling

Part XIII Robust Project Scheduling

Part XIV Project Scheduling Under Interval Uncertainty and Fuzzy Project Scheduling

Part XV General Project Management

Part XVI Project Risk Management

Part XVII Project Scheduling Applications

Part XVIII Case Studies in Project Scheduling

Part XIX Project Management Information Systems

List of Symbols

Miscellaneous

$:=$	Equal by definition, assignment
$\square$	End of proof
$\lceil z \rceil$	Smallest integer greater than or equal to z
$\lfloor z \rfloor$	Greatest integer smaller than or equal to z
$(z)^+$	Maximum of 0 and z

Sets

$\emptyset$	Empty set
$]a, b[$	Open interval $\{x \in \mathbb{R} \mid a < x < b\}$
$[a, b[$	Half open interval $\{x \in \mathbb{R} \mid a \leq x < b\}$
$]a, b]$	Half open interval $\{x \in \mathbb{R} \mid a < x \leq b\}$
$[a, b]$	Closed interval $\{x \in \mathbb{R} \mid a \leq x \leq b\}$
$\|A\|$	Number of elements of finite set A
$A \subset B$	A is proper subset of B
$A \subseteq B$	A is subset of B
$A \setminus B$	Difference of sets A and B
$A \cap B$	Intersection of sets A and B
$A \cup B$	Union of sets A and B
$conv(A)$	Convex hull of set A
$f : A \to B$	Mapping (function) of A into B
$\mathbb{N}$	Set of positive integers
$\mathcal{NP}$	Set of decision problems that can be solved in polynomial time by a non-deterministic Turing machine

$\mathscr{O}$	Landau's symbol: for $f, g : \mathbb{N} \to \mathbb{R}_{\geq 0}$ it holds that $g \in \mathscr{O}(f)$ if there are a constant $c > 0$ and a positive integer n_0 such that $g(n) \leq c\, f(n)$ for all $n \geq n_0$
$\mathbb{R}$	Set of real numbers
$\mathbb{R}^n$	Set of n-tuples of real numbers
$\mathbb{R}_{\geq 0}$	Set of nonnegative real numbers
$\mathbb{Z}$	Set of integers
$\mathbb{Z}_{\geq 0}$	Set of nonnegative integers

Projects, Activities, and Project Networks

δ_{ij}	Weight of arc (i, j), start-to-start minimum time lag between activities i and j
$\mathscr{A}$	Set of all maximal feasible antichains of the precedence order (non-dominated feasible subsets)
$\overline{\mathscr{A}}$	Set of all feasible antichains of the precedence order (feasible subsets)
$A \in \overline{\mathscr{A}}$	Feasible antichain (feasible subset)
$\mathscr{A}(S, t)$	Set of activities in execution at time t given schedule S
d_{ij}	Longest path length from node i to node j in project network N
d_{ij}^{max}	Maximum time lag between the starts of activities i and j
d_{ij}^{min}	Minimum time lag between the starts of activities i and j
$\overline{d}$	Prescribed maximum project duration
E	Arc set of directed graph G or project network N
E_i^-	Set of arcs leading to node i
E_i^+	Set of arcs emanating from node i
$\mathscr{F}$	Set of all minimal forbidden sets
$F \in \mathscr{F}$	Minimal forbidden set
$G = (V, E)$	Directed graph with node set V and arc set E (precedence graph)
i, j	Activities or events of the project
(i, j)	Arc with initial node i and terminal node j
n	Number of activities of the project, without project beginning 0 and project completion $n + 1$
$N = (V, E, \delta)$	Project network with node set V, arc set E, and arc weights δ
p_i	Duration (processing time) of activity i
$Pred(i)$	Set of immediate predecessors of activity i in project network N
$\overline{Pred}(i)$	Set of all immediate and transitive predecessors of activity i in project network N
$Succ(i)$	Set of all immediate successors of activity i in project network N
$\overline{Succ}(i)$	Set of all immediate and transitive successors of activity i in project network N
TE	Transitive closure of the arc set

V	Node set of direct graph G or project network N; Set of activities in an activity-on-node network
V^a	Set of real activities in an activity-on-node network

Resources and Skills

Π_k	Set of periods associated with partially renewable resource k
k	Single (renewable, nonrenewable, partially renewable, or storage) resource
$K = \lvert\mathscr{R}\rvert$	Number of renewable resources
$l \in \mathscr{L}$	Single skill
$L = \lvert\mathscr{L}\rvert$	Number of skills
$L_i = \lvert\mathscr{L}_i\rvert$	Number of skills required by activity i
$\mathscr{L}$	Set of skills
$\mathscr{L}_i$	Set of skills required by activity i
$\mathscr{L}_k$	Set of skills that can be performed by resource k
r_{ik}	Amount of resource k used by activity i
$r_{ik}(t)$	Amount of resource k used by activity i in the t-th period of its execution
r_{il}	Number of resource units with skill l required by activity i
$r_k(S,t)$	Amount of resource k used at time t given schedule S
R_k	Capacity or availability of resource k
$R_k(t)$	Capacity of renewable resource k in period t
$\mathscr{R}$	Set of (discrete) renewable resources (e.g., workers)
$\mathscr{R}_l$	Set of workers possessing skill l
$\mathscr{R}^n$	Set of nonrenewable resources
$\mathscr{R}^p$	Set of partially renewable resources
$\mathscr{R}^s$	Set of storage resources
wc_i	Work content of activity i
$wl_{ik} = p_i \cdot r_{ik}$	Workload of renewable resource k incurred by activity i
$WL_k = R_k \cdot \overline{d}$	Workload capacity of renewable resource k

Multi-Modal Project Scheduling

m	Execution mode
$\mathscr{M}_i$	Set of alternative execution modes for activity i
$M_i = \lvert\mathscr{M}_i\rvert$	Number of modes of activity i
p_{im}	Duration of activity i in execution mode m
r_{ikm}	Amount of resource k used by activity i in execution mode m
x	Mode assignment with $x_{im} = 1$, if activity i is processed in execution mode $m \in \mathscr{M}_i$

	Staff assignment with $x_{ikl} = 1$, if a worker of resource k performs activity i with skill l

Discrete Time-Cost Tradeoff

b	Budget for activity processing
$c_i(p_i)$	Cost for processing activity i with duration p_i $(= c_{im}$ with $p_i = p_{im})$
c_{im}	Cost of executing activity i in mode m
p_{im}	Duration of activity i in mode m

Multi-Project Problems

α_q	Dummy start activity of project q
ω_q	Dummy end activity of project q
d_q	Due date for completion of project q
$\overline{d}_q$	Deadline for completion of project q
n_q	Number of real activities of project q
$q \in Q$	Single project
Q	Set of projects
V_q	Set of activities of project q

Project Scheduling Under Uncertainty and Vagueness

λ	Arrival rate of projects
$\mu_{\hat{z}}(z)$	Membership function of fuzzy set $\hat{z}$
π_σ	Probability of scenario σ $(\sum_{\sigma \in \Sigma} \pi_\sigma = 1)$
$\sigma \in \Sigma$	Single scenario
Σ	Set of scenarios
Σ_i	Set of scenarios for activity i
$E(\tilde{x})$	Expected value of $\tilde{x}$
$f_{\tilde{x}}(x)$	Probability density function (pdf) of random variable $\tilde{x}$ $(= \frac{dF_{\tilde{x}}}{dx}(x))$
$F_{\tilde{x}}(x)$	Cumulative probability distribution function (cdf) of random variable $\tilde{x}$ $(= P(\tilde{x} \leq x))$
$\tilde{p}_i$	Random duration of activity i
$P(A)$	Probability of event A
p_i^{min}, p_i^{max}	Minimum and maximum duration of activity i
$\hat{p}_i$	Fuzzy duration of activity i
$var(\tilde{x})$	Variance of $\tilde{x}$

$\tilde{x}, \tilde{\xi}$	General random variables
x_α	α-quantile ($F_{\tilde{x}}(x_\alpha) = \alpha$)
z	(Crisp) Element from set Z
$\hat{z}$	General fuzzy set

Objective Functions

α	Continuous interest rate
$\beta = e^{-\alpha}$	Discount rate per unit time
c_i^F	Cash flow associated with the start or completion of activity i
$c_i^{F-} > 0$	Disbursement $-c_i^F > 0$ associated with activity or event i
$c_i^{F+} > 0$	Payment $c_i^F > 0$ associated with activity or event i
c_k	Cost for resource k per unit
$C_{max} = S_{n+1}$	Project duration (project makespan)
$f(S)$	Objective function value of schedule S (single-criterion problem); Vector $(f_1(S), \dots, f_\nu(S))$ of objective function values (multi-criteria problem)
$f(S, x)$	Objective function value of schedule S and mode assignment x
f_μ	Single objective function in multi-criteria project scheduling
LB	Lower bound on minimum objective function value
npv	Net present value of the project
$\mathscr{PF}$	Pareto front of multi-criteria project scheduling problem
UB	Upper bound on minimum objective function value
w_i	Arbitrary weight of activity i

Temporal Scheduling

C_i	Completion time of activity i
EC_i	Earliest completion time of activity i
ES	Earliest schedule
ES_i	Earliest start time of activity i
LC_i	Latest completion time of activity i
LS	Latest schedule
LS_i	Latest start time of activity i
S	Schedule
S_i	Start time of activity i or occurrence time of event i
TF_i	Total float of activity i

Models and Solution Methods

ϕ_{ij}^{k}	Amount of resource k transferred from activity i to activity j
ρ_{mut}	Mutation rate
σ_{pop}	Population size
ℓ	Activity list $(i_1, i_2, \ldots, i_n)$
$\mathscr{C}$	Set of activities already scheduled (completed set)
$\mathscr{D}$	Decision set containing all activities eligible for being scheduled
$S^{\mathscr{C}}$	Partial schedule of activities $i \in \mathscr{C}$
t	Time period, start of period $t + 1$
T	Last period, end of planning horizon

Computational Results

$\Delta_{LB}^{\varnothing}$	Average relative deviation from lower bound
Δ_{LB}^{max}	Maximum relative deviation from lower bound
$\Delta_{opt}^{\varnothing}$	Average relative deviation from optimum value
Δ_{opt}^{max}	Maximum relative deviation from optimum value
$\Delta_{UB}^{\varnothing}$	Average relative deviation from upper bound
Δ_{UB}^{max}	Maximum relative deviation from upper bound
LB_0	Critical-path based lower bound on project duration
LB^*	Maximum lower bound
n_{best}	Number of best solutions found
$n_{iter}^{\varnothing}$	Average number of iterations
n_{iter}^{max}	Maximum number of iterations
n_{opt}	Number of optimal solutions found
OS	Order strength of project network
p_{feas}	Percentage of instances for which a feasible solution was found
p_{inf}	Percentage of instances for which the infeasibility was proven
p_{opt}	Percentage of instances for which an optimal solution was found
p_{unk}	Percentage of instances for which it is unknown whether there exists a feasible solution
RF	Resource factor of project
RS	Resource strength of project
t_{cpu}^{lim}	CPU time limit
$t_{cpu}^{\varnothing}$	Average CPU time
t_{cpu}^{max}	Maximum CPU time

Three-Field Classification $\alpha \mid \beta \mid \gamma$ for Project Scheduling Problems[1]

Field α: Resource Environment

PS	Project scheduling problem with limited (discrete) renewable resources
$PS\infty$	Project scheduling problem without resource constraints (time-constrained project scheduling problem)
PSc	Project scheduling problem with limited continuous and discrete renewable resources
PSf	Project scheduling problem with limited renewable resources and flexible resource requirements (problem with work-content constraints)
PSS	Project staffing and scheduling problem with multi-skilled resources of limited workload capacity
$PSS\infty$	Project staffing and scheduling problem with limited multi-skilled resources of unlimited workload capacity
PSp	Project scheduling problem with limited partially renewable resources
PSs	Project scheduling problem with limited storage resources
PSt	Project scheduling problem with limited (discrete) time-varying renewable resources
$MPSm, \sigma, \mu$	Multi-mode project scheduling problem with m limited (discrete) renewable resources of capacity σ and μ nonrenewable resources
MPS	Multi-mode project scheduling problem with limited renewable and nonrenewable resources
$MPS\infty$	Multi-mode project scheduling without resource constraints (time-constrained project scheduling problem)

Field β: Project and Activity Characteristics

The second field $\beta \subseteq \{\beta_1, \beta_2, \ldots, \beta_{13}\}$ specifies a number of project and activity characteristics; $\circ$ denotes the empty symbol.

$\beta_1 : mult$	Multi-project problem
$\beta_1 : \circ$	Single-project problem
$\beta_2 : prec$	Ordinary precedence relations between activities

[1] The classification is a modified version of the classification scheme introduced in Brucker P, Drexl A, Möhring R, Neumann K, Pesch E (1999) Resource-constrained project scheduling: notation, classification, models, and methods. Eur J Oper Res 112:3–41.

$\beta_2 : \mathit{temp}$ Generalized precedence relations between activities (minimum and maximum time lags between start or completion times of activities)

$\beta_2 : \mathit{feed}$ Feeding precedence relations between activities

$\beta_3 : \overline{d}$ Prescribed deadline $\overline{d}$ for project duration

$\beta_3 : \circ$ No prescribed maximum project duration

$\beta_4 : \mathit{bud}$ Limited budget for activity processing

$\beta_4 : \circ$ No limited budget for activity processing

$\beta_5 : p_i = \mathit{sto}$ Stochastic activity durations

$\beta_5 : p_i = \mathit{unc}$ Uncertain activity durations from given intervals

$\beta_5 : p_i = \mathit{fuz}$ Fuzzy activity durations

$\beta_5 : \circ$ Deterministic/crisp activity durations

$\beta_6 : c_i = \mathit{sto}$ Stochastic activity cost

$\beta_6 : c_i = \mathit{unc}$ Uncertain activity cost from given intervals

$\beta_6 : c_i = \mathit{fuz}$ Fuzzy activity cost

$\beta_6 : \circ$ Deterministic/crisp activity cost

$\beta_7 : \mathit{Poi}$ Stochastic arrival of projects with identical project network according to Poisson process

$\beta_7 : \circ$ Immediate availability of project(s)

$\beta_8 : \mathit{act} = \mathit{sto}$ Set of activities to be executed is stochastic

$\beta_8 : \circ$ Set of activities to be executed is prescribed

$\beta_9 : \mathit{pmtn}$ Preemptive problem, activities can be interrupted at any point in time

$\beta_9 : \mathit{pmtn}/\mathit{int}$ Preemptive problem, activities can be interrupted at integral points in time only

$\beta_9 : l\text{-}\mathit{pmtn}/\mathit{int}$ Preemptive problem, activities can be interrupted at integral points in time, the numbers of interruptions per activity are limited by given upper bounds

$\beta_9 : \circ$ Non-preemptive problem (activities cannot be interrupted)

$\beta_{10} : r_{il} = 1$ Each activity requires at most one resource unit with skill l for execution

$\beta_{10} : \circ$ Each activity i requires an arbitrary number of resource units with skill l for execution

$\beta_{11} : \mathit{cal}$ Activities can only be processed during certain time periods specified by activity calendars

$\beta_{11} : \circ$ No activity calendars have to be taken into account

$\beta_{12} : s_{ij}$ Sequence-dependent setup/changeover times of resources between activities i and j

$\beta_{12} : \circ$ No sequence-dependent changeover times

$\beta_{13} : \mathit{nestedAlt}$ The project network is given by a nested temporal network with alternatives, where only a subset of the activities must be executed

$\beta_{13} : \circ$ No alternative activities have to be taken into account

Field γ: Objective Function

f	General (regular or nonregular) objective function
reg	Regular objective function
mac	General mode assignment cost
$staff$	General project staffing cost (project staffing and scheduling)
rob	Robustness measure
$mult$	General multi-criteria problem
$f_1/f_2/\ldots$	Multi-criteria problem with objective functions f_1, $f_2, \ldots$
C_{max}	Project duration
$\Sigma c_i^F \beta^{C_i}$	Net present value of project
$\Sigma c_k \max r_{kt}$	Total availability cost (resource investment problem)
$\Sigma c_k \Sigma r_{kt}^2$	Total squared utilization cost (resource leveling)
$\Sigma c_k \Sigma o_{kt}$	Total overload cost (resource leveling)
$\Sigma c_k \Sigma \Delta r_{kt}$	Total adjustment cost (resource leveling)
$\Sigma c_i(p_i)$	Total cost of activity processing (time-cost tradeoff problem)
wT	Weighted project tardiness

Examples

$PS \mid prec \mid C_{max}$	Basic resource-constrained project scheduling problem (RCPSP)
$PS \mid temp, pmtn \mid C_{max}$	Preemptive resource-constrained project scheduling problem with generalized precedence relations
$MPS\infty \mid prec, \overline{d} \mid \Sigma c_i(p_i)$	Discrete time-cost tradeoff problem (deadline version)
$MPS \mid temp \mid \Sigma c_i^F \beta^{C_i}$	Multi-mode resource-constrained net present value problem with generalized precedence relations
$PS \mid prec \mid C_{max}/\Sigma r_{kt}^2$	Bi-criteria resource-constrained project scheduling problem (project duration, total squared utilization cost)
$PS \mid prec, p_i = sto \mid C_{max}$	Stochastic resource-constrained project scheduling problem

Project Management and Scheduling

Christoph Schwindt and Jürgen Zimmermann

1 Projects, Project Management, and Project Scheduling

Nowadays, *projects* are omnipresent. These unique and temporary undertakings have permeated almost all spheres of life, be it work or leisure, be it business or social activities. Most frequently, projects are encountered in private and public enterprizes. Due to product differentiation and collapsing product life cycles, a growing part of value adding activities in industry and services is organized as projects. In some branches, virtually all revenues are generated through projects. The temporary nature of projects stands in contrast with more traditional forms of business, which consist of repetitive, permanent, or semi-permanent activities to produce physical goods or services (Dinsmore and Cooke-Davies 2005, p. 35).

Projects share common characteristics, although they appear in many forms. Some projects take considerable time and consume a large amount of resources, while other projects can be completed in short time without great effort. To get a clear understanding of the general characteristics of a project, we consider the following two definitions of a project, which are taken from Kerzner (2013, p. 2) and PMI (2013, p. 4).

1. "A project can be considered to be any series of activities and tasks that:
   - have a specific objective to be completed within certain specifications,
   - have defined start and end dates,
   - have funding limits (if applicable),
   - consume human and nonhuman resources (i.e., people, money, equipment),
   - are multifunctional (i.e., cut across several functional lines)."

C. Schwindt (✉) • J. Zimmermann
Institute of Management and Economics, Clausthal University of Technology,
Clausthal-Zellerfeld, Germany
e-mail: christoph.schwindt@tu-clausthal.de; juergen.zimmermann@tu-clausthal.de

2. “A project is a temporary endeavor undertaken to create a unique product, service, or result.”

According to these definitions, we understand a project as a one-time endeavor that consists of a set of activities, whose executions take time, require resources, and incur costs or induce cash flows. Precedence relations may exist between activities; these relations express technical or organizational requirements with respect to the order in which activities must be processed or with respect to their timing relative to each other. Moreover, the scarcity of the resources allocated to the project generally gives rise to implicit dependencies among the activities sharing the same resources, which may necessitate the definition of additional precedence relations between certain activities when the project is scheduled. A project is carried out by a project team, has a deadline, i.e., is limited in time, and is associated with one or several goals whose attainment can be monitored.

Typical examples for projects are:

- construction of a building, road, or bridge,
- development of a new product,
- reorganization in a firm,
- implementation of a new business process or software system,
- procurement and roll-out of an information system,
- design of a new pharmaceutical active ingredient, or
- conducting an election campaign.

Project management deals with the coordination of all initiating, planning, decision, execution, monitoring, control, and closing processes in the course of a project. In other words, it is the application of knowledge, skills, tools, and techniques to project tasks to meet all project interests. According to the Project Management Institute standard definition (PMI 2013, p. 8), managing a project includes

- identifying requirements,
- establishing clearly understandable and viable objectives,
- balancing the competing demands for time, quality, scope, and cost, and
- customizing the specifications, plans, and approach to the concerns and expectations of the different stakeholders.

Consequently, successful project management means to perform the project within time and cost estimates at the desired performance level in accordance with the client, while utilizing the required resources effectively and efficiently.

From a project management point of view, the life cycle of a project consists of five consecutive phases, each of which involves specific managerial tasks (cf., e.g., Klein 2000; Lewis 1997). At the beginning of the first phase, called *project conception*, there is only a vague idea of the project at hand. By means of some feasibility studies as well as economic and risk analyses it is decided whether or not a project should be performed. In the *project definition phase* the project objectives and the organization form of the project are specified. In addition, the

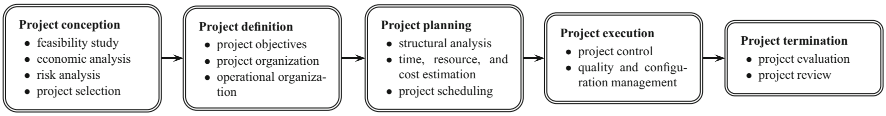


Fig. 1 Project life cycle

operational organization in the form of a roadmap (milestone plan) is conceived. In the *project planning phase* the project is decomposed into precedence-related activities. Then, for each activity the duration, the required resources, and the cost associated with the execution of that activity are estimated. Furthermore, the precedence relations among the activities are specified. Finally, a project schedule is determined by some appropriate planning approach (project scheduling). After these three phases the project is ready for implementation and the *project execution phase* starts. By monitoring the project progress, project management continuously evaluates whether or not the project is performed according to the established baseline schedule. If significant deviations are detected, the plan has to be revised or an execution strategy defined in the planning phase is used to bring the project back to course. Moreover, quality and configuration management are performed in this phase (PMI 2013; Turner 2009). The final *project termination phase* evaluates and documents the project execution after its completion. Figure 1 summarizes the five phases of the project life cycle. Next, we will consider the project scheduling part of the planning phase in more detail.

Project scheduling is mainly concerned with selecting execution modes and fixing execution time intervals for the activities of a project. One may distinguish between time-constrained and resource-constrained project scheduling problems, depending on the type of constraints that are taken into account when scheduling the project. In time-constrained problems it is supposed that the activities are to be scheduled subject to precedence relations and that the required resources can be provided in any desired amounts, possibly at the price of higher execution cost or unbalanced resource usage. In the setting of a resource-constrained project scheduling problem, the availability of resources is necessarily assumed to be limited; consequently, in addition to the precedence relations, resource constraints have to be taken into account. Time-cost tradeoff and resource leveling problems are examples of time-constrained project scheduling problems. These examples show that time-constrained problems also may include a resource allocation problem, which consists in assigning resource units to the execution of the activities over time.

Different types of precedence relations are investigated in this handbook. An ordinary precedence relation establishes a predefined sequence between two activities, the second activity not being allowed to start before the first has been completed. Generalized precedence relations express general minimum and maximum time lags between the start times of two activities. Feeding precedence relations require that an activity can only start when a given minimum percentage of its predecessor activity has been completed. The difference between generalized and feeding precedence relations becomes apparent when the activity durations are not fixed in advance or when activities can be interrupted during their execution.

Throughout this handbook, the term “resource” designates a pool of identical resource units, and the number of resource units available is referred to as the capacity or availability of the resource. In project scheduling, several kinds of resources have been introduced to model input factors of different types. Renewable resources represent inputs like manpower or machinery that are used, but not consumed when performing the project. In contrast, nonrenewable resources comprise factors like a budget or raw materials, which are consumed in the course of the project. Renewable and nonrenewable resources can be generalized to storage resources, which are depleted and replenished over time by the activities of the project. Storage resources can be used to model intermediate products or the cash balance of a project with disbursements and progress payments. Resources like electric power or a paged virtual memory of a computer system, which can be allotted to activities in continuously divisible amounts, are called continuous resources. Partially renewable resources refer to unions of time intervals and can be used to model labor requirements arising, e.g., in staff scheduling problems.

A common assumption in project scheduling is that activities must not be interrupted when being processed. There exist, however, applications for which activity splitting may be advantageous or even necessary. Examples of such applications are the aggregate mid-term planning of project portfolios composed of subprojects or working packages and the scheduling of projects in which certain resources cannot be operated during scheduled downtimes. The preemptive scheduling problems can be further differentiated according to the time points when an activity can be interrupted or resumed. Integer preemption problems assume that an activity can only be split into parts of integral duration, whereas continuous preemption problems consider the general case in which activities may be interrupted and resumed at any point in time.

An important attribute of a project scheduling problem concerns the number of execution modes that can be selected for individual activities. The setting of a single-modal problem premises that there is only one manner to execute an activity or that an appropriate execution mode has been selected for each activity before the scheduling process is started. A multi-modal problem always comprises a mode selection problem, the number of alternative modes for an activity being finite or infinite. Multiple execution modes allow to express resource-resource, resource-time, and resource-cost tradeoffs, which frequently arise in practical project scheduling applications.

With respect to the scheduling objectives, one may first distinguish between single-criterion and multi-criteria problems. A problem of the latter type includes several conflicting goals and its solution requires concepts of multi-criteria decision making like goal programming or goal attainment models. Second, objective functions can be classified as being regular or non-regular. Regular objective functions are defined to be componentwise nondecreasing in the start or completion times of the activities. Obviously, a feasible instance of a problem with a regular objective function always admits a solution for which no activity can be scheduled earlier without delaying the processing of some other activity. Since in this case, the search for an optimal schedule can be limited to such "active" schedules, problems with regular objective functions are generally more tractable than problems involving a non-regular objective function.

A further attribute of project scheduling problems refers to the level of available information. The overwhelming part of the project scheduling literature addresses deterministic problem settings, in which it is implicitly assumed that all input data of the problem are precisely known in advance and no disruptions will occur when the schedule is implemented. In practice, however, projects are carried out in stochastic and dynamic environments. Hence, it seems reasonable to account for uncertainty when deciding on the project schedule. This observation leads to stochastic project scheduling problems or project scheduling problems under interval uncertainty, depending on whether or not estimates of probability distributions for the uncertain parameters are supposed to be available. Fuzzy project scheduling problems arise in a context in which certain input data are vague and cannot be specified on a cardinal scale, like assessments by means of linguistic variables.

Finally, project scheduling problems may be categorized according to the distribution of information or the number of decision makers involved. Most work on project scheduling tacitly presumes that the projects under consideration can be scheduled centrally under a symmetric information setting, in which there is a single decision maker or all decision makers pursue the same goals and are provided access to the same information. However, in a multi-project environment, decentralized decision making may be the organization form of choice, generally leading to an asymmetric information distribution and decision makers having their own objectives. In this case, a central coordination mechanism is needed to resolve conflicts and to achieve a satisfying overall project performance.

Table 1 summarizes the classification of project scheduling problems considered in this handbook. For further reading on basic elements and more advanced concepts of project scheduling we refer to the surveys and handbooks by Artigues et al. (2008), Demeulemeester and Herroelen (2002), Hartmann and Briskorn (2010), and Józefowska and Węglarz (2006).

2 Scope and Organization of the Handbook

Given the long history and practical relevance of project management and scheduling, one might be tempted to suppose that all important issues have been addressed and all significant problems have been solved. The large body of research papers, however, that have appeared in the last decade and the success of international project management and scheduling conferences prove that the field remains a very active and attractive research area, in which major and exciting developments are still to come.

Table 1 Classification of project scheduling problems

Attributes	Characteristics
Type of constraints	Time-constrained problem
	Resource-constrained problem
Type of precedence relations	Ordinary precedence relations
	Generalized precedence relations
	Feeding precedence relations
Type of resources	Renewable resources
	Nonrenewable resources
	Storage resources
	Continuous resources
	Partially renewable resources
Type of activity splitting	Non-preemptive problem
	Integer preemption problem
	Continuous preemption problem
Number of execution modes	Single-modal problem
	Multi-modal problem
Number of objectives	Single-criterion problem
	Multi-criteria problem
Type of objective function	Regular function
	Non-regular function
Level of information	Deterministic problem
	Stochastic problem
	Problem under interval uncertainty
	Problem under vagueness
Distribution of information	Centralized problem (symmetric distribution)
	Decentralized problem (asymmetric distribution)

This handbook is a collection of 62 chapters presenting a broad survey on key issues and recent developments in project management and scheduling. Each chapter has been contributed by recognized experts in the respective domain. The two volumes comprise contributions from seven project management and scheduling areas, which are organized in 19 parts. The first three areas are covered by Vol. 1 of the handbook, the remaining four areas being treated in Vol. 2. The covered topics range from basic project scheduling problems and their generalizations through multi-project planning, project scheduling under uncertainty and vagueness, recent developments in general project management and project risk management to applications, case studies, and project management information systems. The following list provides an overview of the handbook's contents.

- Area A: Project duration problems in single-modal project scheduling
  - Part I: The Resource-Constrained Project Scheduling Problem
  - Part II: The Resource-Constrained Project Scheduling Problem with Generalized Precedence Relations

  - Part III: Alternative Resource Constraints in Project Scheduling
  - Part IV: Preemptive Project Scheduling

- Area B: Alternative objectives in single-modal project scheduling

  - Part V: Non-Regular Objectives in Project Scheduling
  - Part VI: Multi-Criteria Objectives in Project Scheduling

- Area C: Multi-modal project scheduling

  - Part VII: Multi-Mode Project Scheduling Problems
  - Part VIII: Project Staffing and Scheduling Problems
  - Part IX: Discrete Time-Cost Tradeoff Problems

- Area D: Multi-project problems

  - Part X: Multi-project scheduling
  - Part XI: Project Portfolio Selection Problems

- Area E: Project scheduling under uncertainty and vagueness

  - Part XII: Stochastic Project Scheduling
  - Part XIII: Robust Project Scheduling
  - Part XIV: Project Scheduling Under Interval Uncertainty and Fuzzy Project Scheduling

- Area F: Managerial approaches

  - Part XV: General Project Management
  - Part XVI: Project Risk Management

- Area G: Applications, case studies, and information systems

  - Part XVII: Project Scheduling Applications
  - Part XVIII: Case Studies in Project Scheduling
  - Part XIX: Project Management Information Systems

The parts of Areas A to E, devoted to models and methods for project scheduling, follow a development from standard models and basic concepts to more advanced issues such as multi-criteria problems, project staffing and scheduling, decentralized decision making, or robust optimization approaches. Area F covers research opportunities and emerging issues in project management. The chapters of the last Area G report on project management and scheduling applications and case studies in various domains like production scheduling, R&D planning, make-or-buy decisions and supplier selection, scheduling in computer grids, and the management of construction projects. Moreover, three chapters address the benefits and capabilities of project management information systems.

Most chapters are meant to be accessible at an introductory level by readers with a basic background in operations research and probability calculus. The intended audience of this book includes project management professionals, graduate students in management, industrial engineering, computer science, or operations research, as well as scientists working in the fields of project management and scheduling.

3 Outline of the Handbook

Area A of this handbook is dedicated to single-modal project scheduling problems in which the activities have to be scheduled under precedence relations and resource constraints and the objective consists in minimizing the duration (or makespan) of the project. In practice, these project scheduling problems have a large range of applications, also beyond the field of proper project management. For example, production scheduling and staff scheduling problems can be modeled as single-modal project scheduling problems. In order to model specific practical requirements like prescribed minimum and maximum time lags between activities, availability of materials and storage capacities, or divisible tasks, project scheduling models including generalized precedence relations, new types of resource constraints, or preemptive activities have been proposed. These extensions to the basic model are also addressed in this portion of the handbook.

Part I is concerned with the classical resource-constrained project scheduling problem RCPSP. Solution methods for the RCPSP have been developed since the early 1960s and this problem is still considered the standard model in project scheduling. In Chap. 1 Rainer Kolisch reviews shifts, schedule types, and schedule-generation schemes for the RCPSP. A shift transforms a schedule into another schedule by displaying sets of activities. Based on the introduced shifts, different types of schedules, e.g., semi-active and active schedules, are defined. Furthermore, two different schedule-generation schemes are presented. The serial schedule-generation scheme schedules the activities one by one at their respective earliest feasible start times. The parallel schedule-generation scheme is time-oriented and generates the schedule by iteratively adding concurrent activities in the order of increasing activity start times. Variants of the two schemes for the resource-constrained project scheduling problem with generalized precedence relations and for the stochastic resource-constrained project scheduling problem are discussed as well. Chapter 2, written by Christian Artigues, Oumasr Koné, Pierre Lopez, and Marcel Mongeau, surveys (mixed-)integer linear programming formulations for the RCPSP. The different formulations are divided into three categories: First, time-indexed formulations are presented, in which time-indexed binary variables encode the status of an activity at the respective point in time. The second category gathers sequencing formulations including two types of variables. Continuous natural-date variables represent the start time of the activities and binary sequencing variables are used to model decisions with respect to the ordering of activities that compete for the same resources. Finally, different types of event-based formulations are considered, containing binary assignment and continuous positional-date variables. In Chap. 3 Sigrid Knust overviews models and methods for calculating lower bounds on the minimum project duration for the RCPSP. Constructive and destructive bounds are presented. The constructive lower bounds are based on the relaxation or Lagrangian dualization of the resource constraints or a disjunctive relaxation allowing for activity preemption and translating precedence relations into disjunctions of activities. Destructive lower bounds arise from disproving hypotheses on upper bounds on the minimum objective function value. Knust reviews destructive lower bounds for the RCPSP that are calculated using constraint propagation and a linear programming formulation. Chapter 4 by Anurag Agarwal, Selcuk Colak, and Selcuk Erenguc considers meta-heuristic methods for the RCPSP. Important concepts of heuristic methods as well as 12 different meta-heuristics are presented. Amongst others, genetic algorithms, simulated annealing methods, and ant-colony optimization are discussed. A neuro-genetic approach is presented in more detail. This approach is a hybrid of a neural-network based method and a genetic algorithm.

Part II deals with the resource-constrained project scheduling problem with generalized precedence relations RCPSP/max. Generalized precedence relations express minimum and maximum time lags between the activities and can be used to model, e.g., release dates and deadline of activities or specified maximum makespans for the execution of subprojects. In Chap. 5 Lucio Bianco and Massimiliano Caramia devise lower bounds and exact solution approaches for the RCPSP/max. First, a new mathematical formulation for the resource-unconstrained project scheduling problem is presented. Then, they propose a lower bound for the RCPSP/max relying on the unconstrained formulation. The branch-and-bound method is based on a mixed-integer linear programming formulation and a Lagrangian relaxation based lower bound. The mixed-integer linear program includes three types of time-indexed decision variables. The first two types are binary indicator variables for the start and the completion of activities, whereas the third type corresponds to continuous variables providing the relative progress of individual activities at the respective points in time. Chapter 6 presents a constraint satisfaction solving framework for the RCPSP/max. Amedeo Cesta, Angelo Oddi, Nicola Policella, and Stephen Smith survey the state of the art in constraint-based scheduling, before the RCPSP/max is formulated as a constraint satisfaction problem. The main idea of their approach consists in establishing precedence relations between activities that share the same resources in order to eliminate all possible resource conflicts. Extended optimizing search procedures aiming at minimizing the makespan and improving the robustness of a solution are presented. Chapter 7, written by Andreas Schutt, Thibaut Feydy, Peter Stuckey, and Mark Wallace, elaborates on a satisfiability solving approach for the RCPSP/max. First, basic concepts such as finite domain propagation, boolean satisfiability solving, and lazy clause generation are discussed. Then, a basic model for the RCPSP/max and several expansions are described. The refinements refer to the reduction of the initial domains of the start time variables and the identification of incompatible activities that cannot be in progress simultaneously. The authors propose a branch-and-bound algorithm that is based on start-time and/or conflict-driven branching strategies and report on the results of an experimental performance analysis.

Part III focuses on resource-constrained project scheduling problems with alternative types of resource constraints. The different generalizations of the renewable-resources concept allow for modeling various kinds of limited input factors arising in practical applications of project scheduling models. Chapter 8, written by Sönke Hartmann, considers the resource-constrained project scheduling problem with time-varying resource requirements and capacities RCPSP/t. After a formal description of the problem, relationships to other project scheduling problems are discussed and practical applications in the field of medical research and production scheduling are treated. The applicability of heuristics for the RCPSP to the more general RCPSP/t is analyzed and a genetic algorithm for solving the RCPSP/t is presented. In Chap. 9 Jacques Carlier and Aziz Moukrim consider project scheduling problems with storage resources. In particular, the general project scheduling problem with inventory constraints, the financing problem, and the project scheduling problem with material-availability constraints are discussed. For the general problem setting, in which for each storage resource the inventory level must be maintained between a given safety stock and the storage capacity, two exact methods from literature are reviewed. The financing problem corresponds to the single-resource case in which the occurrence times of the project events replenishing the storages are fixed and no upper limitation on the inventory levels are given. This problem can be solved by a polynomial-time shifting algorithm. Eventually, the authors explain how the general problem can be solved efficiently when the storage capacities are relaxed and a linear order on all depleting events is given. Chapter 10, written by Grzegorz Waligóra and Jan Węglarz, is concerned with the resource-constrained project scheduling problem with discrete and continuous resources DCRCPSP. First, the authors survey the main theoretical results that have been achieved for the continuous resource allocation setting. Then, the DCRCPSP with an arbitrary number of discrete resources and a single continuous resource with convex or concave processing rate, respectively, is analyzed. For the case of concave processing rates, a solution method based on feasible sequences of activity sets is presented. In Chap. 11 Ramon Alvarez-Valdes, Jose Manuel Tamarit, and Fulgencia Villa discuss the resource-constrained project scheduling problem with partially renewable resources RCPSP/π. After the definition of the problem, the authors review different types of requirements of real-world scheduling problems that can be modeled using partially renewable resources and survey the existing solution procedures for RCPSP/π. Preprocessing procedures and two heuristic approaches, a GRASP algorithm and a scatter search method, are treated in detail.

Part IV is devoted to preemptive project scheduling problems, in which activities can be temporarily interrupted and restarted at a later point in time. In some applications, especially if vacation or scheduled downtimes of resources are taken into account, the splitting of activities may be unavoidable. Chapter 12 by Sacramento Quintanilla, Pilar Lino, Ángeles Pérez, Francisco Ballestín, and Vicente Valls considers the resource-constrained project scheduling problem Maxnint_PRCPSP under integer activity preemption and upper bounds on the number of interruptions per activity. Existing procedures for the RCPSP are adapted to solve the Maxnint_PRCPSP, and procedures tailored to the Maxnint_PRCPSP are presented. In addition, the chapter reviews a framework for modeling different kinds of precedence relations when activity preemption is allowed. In Chap. 13 Christoph

Schwindt and Tobias Paetz first present a survey on preemptive project scheduling problems and solution methods. Next, they propose a continuous preemption resource-constrained project scheduling problem with generalized feeding precedence relations, which includes most of the preemptive project scheduling problems studied in the literature as special cases. Based on a reduction of the problem to a canonical form with nonpositive completion-to-start time lags between the activities, structural issues like feasibility conditions as well as upper bounds on the number of activity interruptions and the number of positive schedule slices are investigated. Moreover, a novel MILP problem formulation is devised, and preprocessing and lower bounding techniques are presented.

Area B of the handbook is dedicated to single-modal project scheduling problems with general objective functions, including multi-criteria problems. Nonregular objective functions motivated by real-world applications are, e.g., the net present value of the project, the resource availability cost, or different resource leveling criteria. In practice, project managers often have to pursue several conflicting goals. Traditionally, the respective scheduling problems have been tackled as single-objective optimization problems, combining the multiple criteria into a single scalar value. Recently, however, more advanced concepts of multi-criteria decision making received increasing attention in the project scheduling literature. Based on these concepts, project managers may generate a set of alternative and Pareto-optimal project schedules in a single run.

Part V treats project scheduling problems with single-criteria non-regular objective functions. These problems are generally less tractable than problems involving a regular objective function like the project duration because the set of potentially optimal solutions must be extended by non-minimal points of the feasible region. The resource-constrained project scheduling problem with discounted cash flows RCPSPDC is examined in Chap. 14. The sum of the discounted cash flows associated with expenditures and progress payments defines the net present value of the project, and the problem consists in scheduling the project in such as way that the net present value is maximized. Hanyu Gu, Andreas Schutt, Peter Stuckey, Mark Wallace, and Geoffrey Chu present an exact solution procedure relying on the lazy clause generation principle. Moreover, they propose a Lagrangian relaxation based forward-backward improvement heuristic as well as a Lagrangian method for large problem instances. Computational results on test instances from the literature and test cases obtained from a consulting firm provide evidence for the performance of the algorithms. In Chap. 15 Savio Rodrigues and Denise Yamashita present exact methods for the resource availability cost problem RACP. The RACP addresses situations in which the allocation of a resource incurs a cost that is proportional to the maximum number of resource units that are requested simultaneously at some point in time during the project execution. The resource availability cost is to be minimized subject to ordinary precedence relations between the activities and a deadline for the project termination. An exact algorithm based on minimum bounding procedures and heuristics for reducing the search space are described in detail. Particular attention is given to the search strategies and the selection of cut candidates. The authors report on computational results on a set of randomly generated test instances. Chapter 16, written by Vincent Van Peteghem and Mario Vanhoucke, considers heuristic methods for the RACP and the RACPT, i.e., the RACP with tardiness cost. In the RACPT setting, a due date for the project completion is given and payments arise when the project termination is delayed beyond this due date. Van Peteghem and Vanhoucke provide an overview of existing meta-heuristic methods and elaborate on a new search algorithm inspired by weed ecology. In Chap. 17 Julia Rieck and Jürgen Zimmermann address different resource leveling problems RLP. Resource leveling is concerned with the problem of balancing the resource requirements of a project over time. Three different resource leveling objective functions are discussed, for which structural properties and respective schedule classes are revisited. A tree-based branch-and-bound procedure that takes advantage of the structural properties is presented. In addition, several mixed-integer linear programming formulations for resource leveling problems are given and computational experience on test sets from the literature is reported. In Chap. 18 Symeon Christodoulou, Anastasia Michaelidou-Kamenou, and Georgios Ellinas present a literature review on heuristic solution procedures for different resource leveling problems. For the total squared utilization cost problem they devise a meta-heuristic method that relies on a reformulation of the problem as an entropy maximization problem. First, the minimum moment method for entropy maximization is presented. This method is then adapted to the resource leveling problem and illustrated on an example project.

Part VI covers multi-criteria project scheduling problems, placing special emphasis on structural issues and the computation of the Pareto front. Chapter 19, written by Francisco Ballestín and Rosa Blanco, addresses fundamental issues arising in the context of multi-objective project scheduling problems. General aspects of multi-objective optimization and peculiarities of multi-objective resource-constrained project scheduling are revisited, before a classification of the most important contributions from the literature is presented. Next, theoretical results for time- and resource-constrained multi-objective project scheduling are discussed. In addition, the authors provide a list of recommendations that may guide the design of heuristics for multi-objective resource-constrained project scheduling problems. Chapter 20, contributed by Belaïd Aouni, Gilles d'Avignon, and Michel Gagnon, examines goal programming approaches to multi-objective project scheduling problems. After presenting a generic goal programming model, the authors develop a goal programming formulation for the resource-constrained project scheduling problem, including the project duration, the resource allocation cost, and the quantity of the allocated resources as objective functions. In difference to the classical resource allocation cost problem, the model assumes that the availability cost refers to individual resource units and is only incurred in periods during which the respective unit is actually used.

Area C of this handbook is devoted to multi-modal project scheduling problems, in which for each activity several alternative execution modes may be available for selection. Each execution mode defines one way to process the activity, and alternative modes may differ in activity durations, cost, resource requirements, or resource usages over time. The project scheduling problem is then complemented by a mode selection problem, which consists in choosing one execution mode for each activity. Multi-modal problems typically arise from tradeoffs between certain input factors like renewable or nonrenewable resources, durations, or cost. Other types of multi-modal problems are encountered when multi-skilled personnel has to be assigned to activities with given skill requirements or when the resource requirements are specified as workloads rather than by fixed durations and fixed resource demands.

Part VII deals with multi-modal project scheduling problems in which the activity modes represent relations between activity durations and demands for renewable, nonrenewable, or financial resources. This problem setting allows for modeling resource-resource and resource-time tradeoffs, which frequently arise in practical project management. In Chap. 21 Marek Mika, Grzegorz Waligóra, and Jan Węglarz provide a comprehensive overview of the state of the art in multi-modal project scheduling. One emphasis of the survey is on the basic multi-mode resource-constrained project duration problem MRCPSP, for which they review mixed-integer linear programming formulations, exact and heuristic solution methods, as well as procedures for calculating lower bounds on the minimum project duration. Moreover, they also revisit special cases and extensions of the basic problem as well as multi-mode problems with financial and resource-based objectives. Chapter 22, written by José Coelho and Mario Vanhoucke, presents a novel solution approach to the multi-mode resource-constrained project scheduling problem MRCPSP, which solves the mode assignment problem using a satisfiability problem solver. This approach is of particular interest since it takes advantage of the specific capabilities of these solvers to implement learning mechanisms and to combine a simple mode feasibility check and a scheduling step based on a single activity list. A capital-constrained multi-mode scheduling problem is investigated in Chap. 23 by Zhengwen He, Nengmin Wang, and Renjing Liu. The problem consists in selecting activity modes and assigning payments to project events in such a way that the project's net present value is maximized and the cash balance does not go negative at any point in time. The execution modes of the activities represent combinations of activity durations and associated disbursements. In Chap. 24 Philipp Baumann, Cord-Ulrich Fündeling, and Norbert Trautmann consider a variant of the resource-constrained project scheduling problem in which the resource usage of individual activities can be varied over time. For each activity the total work content with respect to a distinguished resource is specified, and the resource usages of the remaining resources are determined by the usage of this distinguished resource. A feasible distribution of the work content over the execution time of an activity can be interpreted as an execution mode. The authors present a priority-rule based heuristic and a mixed-integer linear programming formulation, which are compared on a set of benchmark instances.

Part VIII addresses different variants of project staffing and scheduling problems. In those problem settings, the execution of a project activity may require several skills. It then becomes necessary to assign appropriate personnel to the activities and to decide on the skills with which they contribute to each activity. Isabel Correia and Francisco Saldanha-da-Gama develop a generic mixed-integer programming formulation for project staffing and scheduling problems, which is presented in Chap. 25. The formulation captures various features like unary multi-skilled resources, which contribute with at most one skill to each activity, workload capacities of the resources, multi-unit skill requirements of the activities, and generalized precedence relations. This framework is illustrated by providing MILP models for two project staffing and scheduling problems discussed in the literature, the multi-skill project scheduling problem MSPSP and the project scheduling problem with multi-purpose resources PSMPR. In Chap. 26 Carlos Montoya, Odile Bellenguez-Morineau, Eric Pinson, and David Rivreau present a heuristic method for the MSPSP, which is based on integrating column generation and Lagrangian relaxation techniques. The MSPSP consists in assigning the multi-skilled resources to the activities so as to minimize the project duration under ordinary precedence relations between the activities. The authors develop two master problem formulations, which are heuristically solved by iteratively considering restricted versions of the master problem defined on a pool of variables. In each iteration, new variables with negative reduced cost are entered into the pool, which are identified via respective pricing problems. The required dual multipliers are obtained from solving the LP relaxation of the current restricted master problem by alternating iterations of a subgradient procedure for the Lagrangian dual and simplex iterations. Project staffing and scheduling problems of type PSMPR are discussed in Chap. 27. In difference to the MSPSP, the availability of each resource is limited by a maximum workload that can be processed in the planning horizon, and a general staffing cost function is considered. The staffing cost depends on the assignment of resources to skill requirements of the activities. Haitao Li devises an exact algorithm for the general problem with convex staffing cost. The hybrid Benders decomposition method starts from hierarchically dividing the problem into a relaxed master problem covering the assignment decisions and a feasibility subproblem modeling the scheduling decisions. Both levels are linked by top-down instructions and a bottom-up feedback mechanism adding Benders cuts to the relaxed master problem when the scheduling problem is infeasible. The feasibility of the scheduling problem is checked using a constraint programming algorithm. In Chap. 28 Cheikh Dhib, Ameur Soukhal, and Emmanuel Néron address a generalization of the MSPSP in which an activity can be interpreted as a collection of concurrent subactivities requiring a single skill each and possibly differing in durations. Moreover, it is assumed that the subactivities must be started simultaneously, but may be interrupted and resumed individually at integral points in time. The authors propose a mixed-integer linear programming formulation of the problem and describe priority-rule based solution methods, which are based on the parallel schedule-generation scheme.

Discrete time-cost tradeoff problems, which are the subject of **Part IX**, represent a type of multi-modal project scheduling problems that are frequently encountered in practice. This type of problems occur when the processing of certain activities can be sped up by assigning additional resources, leading to higher execution cost. In Chap. 29 Joseph Szmerekovsky and Prahalad Venkateshan provide a literature review on the classical discrete time-cost tradeoff problem DTCTP. Furthermore, they discuss a new integer programming formulation for a version of the DTCTP with irregular start time costs of the activities. For the special case where the start time costs represent the net present value of an activity, the formulation is compared to three alternative MILP models in an extensive computational experiment. In Chap. 30 Mario Vanhoucke studies three extensions of the DTCTP and an electromagnetic meta-heuristic algorithm to solve these problems. The setting of the DTCTP with time-switch constraints presupposes that activities can only be processed in certain time periods defined by given work/rest patterns. In addition to the direct activity costs, the objective function of the DTCTP with work continuity constraints also includes costs for the supply of resources required by groups of activities; this variant of the problem can be reduced to the basic DTCTP. Finally, the DTCTP with net present value optimization is considered.

Area D of the handbook is dedicated to project planning problems involving several individual projects. We distinguish between multi-project scheduling problems, for which the set of projects to be scheduled is assumed given, and project portfolio selection problems, dealing with the choice of the projects to be actually performed. In both scenarios, there may exist dependencies between the individual projects, for example due to precedence relations between activities of different projects or due to the joint requirements for resources.

Part X deals with the first type of multi-project problems. When scheduling concurrent projects, an important question concerns the distribution of information. In the basic multi-project scheduling problem, it is assumed that all planning data are available to a single decision maker, who may centrally schedule the entire project portfolio. On the other hand, decentralized multi-project scheduling covers the situation in which information is distributed over different decision makers, who may pursue individual targets. In this case, a central coordination mechanism is needed to resolve conflicts between the individual projects. In Chap. 31 Jos Fernando Gonçalves, Jorge Jos de Magalhes Mendes, and Mauricio Resende provide a literature overview on basic multi-project scheduling problems BMPSPS. Furthermore, they develop a biased random-key genetic algorithm for the variant of the problem in which a separable polynomial function in the tardiness, the earliness, and the flow time overrun of all projects is to be minimized subject to precedence relations and the limited availability of shared resources. The decentralized multi-project scheduling problem DRCMPSP is addressed in Chap. 32. In their contribution, Andreas Fink and Jörg Homberger discuss implications of the distributed character of the problem. In addition, they provide a classification scheme of different types of DRCMPSP, categorizing problems according to the basic problem structure, the number of decision makers, the distribution of information, and the local and global objectives. The chapter also contains an extensive discussion and classification of solution approaches presented in literature, including auction and negotiation based coordination schemes.

Part XI focuses on project portfolio selection problems. Often there are more projects on offer than resources available to carry them out. In this case project management has to choose the right project portfolio for execution. In Chap. 33 Ana Fernández Carazo considers multi-criteria problems in which the performance of a portfolio is measured according to a set of conflicting goals. First she identifies a number of key factors characterizing multi-criteria project portfolio selection problems and discusses the different ways in which those factors have been modeled in the literature. Based on this analysis, a proposal for a general project portfolio selection model is developed, which synthesizes various features of previous models. Finally, a binary nonlinear multi-criteria programming formulation of the new model is provided. Walter Gutjahr in Chap. 34 surveys models for project portfolio selection problems which include learning and knowledge depreciation effects. Different types of learning curves are reviewed and it is explained how these models have been used in the context of project staffing and scheduling problems. For the integration of skill development into project portfolio selection models, a mixed-integer nonlinear programming formulation is proposed. Moreover, analytical results for continuous project portfolio investment problems under skill development are reviewed, for which it is assumed that projects can also be partially funded.

Area E of the handbook covers the realm of project scheduling under uncertainty and vagueness, an issue that is widely recognized as being highly relevant to practical project management. Stochastic scheduling problems refer to decision situations under risk, in which quantities like activity durations or activity costs are defined as random variables with known distributions and the objective consists in optimizing the expected value of some performance measure. A solution to such a stochastic problem is commonly given by a policy that is applied when the project is executed. Robust project scheduling is concerned with the problem of finding a predictive baseline schedule that still performs well in case of disruptions or adverse scenarios. Interval uncertainty designates a situation in which only lower and upper bounds can be estimated with sufficient accuracy, but no probability distributions are known. Finally, the concept of fuzzy sets allows to model situations in which vague information, which is only available on an ordinal scale, should be taken into account.

Part XII addresses different types of stochastic project scheduling problems. Chapter 35, contributed by Wolfram Wiesemann and Daniel Kuhn, deals with the stochastic time-constrained net present value problem. Both the activity durations and the cash flows associated with the activities are supposed to be independent random variables. Having discussed the relevance and challenges of stochastic net present value problems, the authors review the state of the art for two variants of the problem. If the activity durations are assumed to be exponentially distributed, the problem can be modeled as a discrete-time Markov decision process with a constant discount rate, for which different exact solution procedures are available. Alternatively, activity durations and cash flows can be represented using discrete scenarios with given probabilities. The resulting stochastic net present value problem SNPV can be formulated as a mixed-integer linear program. Several heuristic solution approaches from literature are outlined. In Chap. 36 Evelina Klerides and Eleni Hadjiconstantinou examine the stochastic discrete time-cost tradeoff problem SDTCTP. They survey the literature on static and dynamic versions of the deadline and the budget variant of this problem. For the dynamic budget variant of SDTCTP it is shown that the problem can be formulated as a multi-stage stochastic binary program with decision-dependent uncertainty. Furthermore, the authors present effective methods for computing lower bounds and good feasible solutions, which are respectively based on a two-stage relaxation and a static mode selection policy. The resource-constrained project scheduling problem with random activity durations SRCPSP is the subject of Chap. 37. Maria Elena Bruni, Patrizia Beraldi, and Francesca Guerriero give an overview of models and methods that have been proposed for different variants of this problem. They develop a heuristic based on the parallel schedule-generation scheme, which in each iteration determines the predictive completion times of the scheduled activities by solving a chance-constrained program. The presented approach is innovative in two respects. First, the use of joint probabilistic constraints allows to relax the traditional assumption that the start time of an activity can be disturbed by at most one predecessor activity at a time. Second, similar to robust project scheduling approaches, a solution to the problem is a predictive baseline schedule that is able to absorb a large part of possible disruptions. The objective, however, still consists, for given confidence level, in finding a schedule with minimum makespan. Hence, the problem to be solved can be viewed as a dual of a robust scheduling problem. The heuristic is illustrated on a real-life construction project. Chapter 38, by Saeed Yaghoubi, Siamak Noori, and Amir Azaron, tackles a multi-criteria multi-project scheduling problem in which projects arrive dynamically according to a Poisson process. Activity durations and direct costs for carrying out activities are assumed to be independent random variables. The execution of the projects is represented as a stochastic process in a queueing network with a maximum number of concurrent projects, each activity being performed at a dedicated service station. The expected values of the activity durations and the direct costs are respectively nonincreasing and nondecreasing functions of the amount of a single resource that is assigned to the service station. The problem consists in allocating the limited capacity of the resource in such a way that the mean project completion time is minimized, the utilization of the service stations is maximized, and the probability that the total direct cost exceeds the available budget is minimum. The authors apply continuous-time Markov processes and particle swarm optimization to solve this multi-objective problem using a goal attainment technique.

Part XIII comprises two chapters on robust optimization approaches to project scheduling problems under uncertainty. The basic idea of robust project scheduling consists in establishing a predictive baseline schedule with a diminished vulnerability to disturbances or adverse scenarios and good performance with respect to some genuine scheduling objective. There are many ways in defining the robustness of a schedule. For example, a schedule may be considered robust if it maximizes the probability of being implementable without modifications. Alternatively, the robustness may refer to the genuine objective instead of the feasibility; a robust schedule then typically optimizes the worst-case performance. In difference to stochastic project scheduling, robust project scheduling approaches do not necessarily presuppose information about the probability distributions of the uncertain input parameters of the problem. In Chap. 39 Öncü Hazır, Mohamed Haouari, and

Erdal Erel discuss a robust discrete time-cost tradeoff problem in which for the activity cost associated with a given mode an interval of possible realizations is specified, but no probability distribution is assumed to be known. The authors devise a mixed-integer programming formulation for this problem. The objective function is defined to be the sum of all most likely activity mode costs plus the maximum surplus cost that may be incurred if for a given number of activities, the direct cost does not assume the most likely but the highest value. The latter number of activities may be used to express the risk attitude of the decision maker. In addition, six categories of time-based robustness measures are presented and a two-phase scheduling algorithm for placing a project buffer at minimum additional cost is outlined. Based on this algorithm, the relationship between the required budget augmentation and the average delay in the project completion time can be analyzed. The robust resource-constrained project scheduling problem with uncertain activity durations is investigated in Chap. 40 by Christian Artigues, Roel Leus, and Fabrice Talla Nobibon. Like in the preceding chapter, it is assumed that no probability distributions are available; the sets of possible realizations of activity durations may form intervals or finite sets. The problem is formulated as a minimax absolute-regret model for which the objective is to find an earliest start policy that minimizes the worst-case difference between the makespan obtained when implementing the policy and the respective optimum ex-post makespan. An exact scenario-relaxation algorithm and a scenario-relaxation based heuristic are presented for this problem.

Part XIV is devoted to project scheduling problems under interval uncertainty and to fuzzy project scheduling. In Chap. 41 Christian Artigues, Cyril Briand, and Thierry Garaix survey results and algorithms for the temporal analysis of projects for which the uncertain activity durations are represented as intervals. The temporal analysis computations provide minimum and maximum values for the earliest and latest start times of the activities and the total floats. Whereas the earliest start times can be calculated as longest path lengths like in the case of fixed activity durations, the computation of the latest start times is less simple. Two algorithms with polynomial time complexity are presented. Interestingly, the maximum total float of the activities can also be computed efficiently, whereas the computation of the minimum total floats constitutes an $\mathcal{NP}$-hard problem. The chapter elaborates on a recent branch-and-bound algorithm for the latter problem. Hua Ke and Weimin Ma in Chap. 42 study a fuzzy version of the linear time-cost tradeoff problem in which the normal activity durations are represented as fuzzy variables. The authors survey literature on time-cost tradeoff problems under uncertainty and vagueness. Using elements of credibility theory, the concepts of expected values, quantiles, and probabilistic constraints can be translated from random to fuzzy variables. Based on these concepts, three fuzzy time-cost tradeoff models are proposed, respectively, providing schedules with minimum α-quantile of the total cost, with minimum expected cost, and with maximum credibility of meeting the budget constraint. In addition, a hybrid method combining fuzzy simulations and a genetic algorithm for solving the three models is presented.

Area F addresses managerial approaches to support decision makers faced with increasingly complex project environments. Complex challenges arise, for example, when dealing with project portfolios, or when a project is performed on a client-contractor basis and the goals of both parties must be streamlined, or when risks arise from several sources and these risks are not independent from each other. These and further challenges are discussed in the two parts of Area F.

Part XV is concerned with general project management issues, covering project portfolio management, relational partnerships and incentive mechanisms, and specific challenges encountered in product development and engineering projects. In Chap. 43 Nicholas Hall contrasts the rapid growth of project activities in firms with the lack of trained project management professionals and research-based project management concepts. He proposes 11 areas for future research to reduce the gap between the great practical importance and the limited theoretical foundations of project management in these areas. Chapter 44 by Peerasit Patanakul addresses issues that arise in multi-project environments. These issues comprise the assignment of project managers to projects, organizational factors that enhance multi-project management, and alternative roles of a project management office. New product development constitutes a classical application area of project management procedures and tools. Nevertheless, managing product innovation is still a challenging task, due to the uncertainty associated with the development process and the strategic importance of its success. In Chap. 45 Dirk Pons provides guidelines from a systems engineering perspective, emphasizing on the management of human resources in the development process. Another traditional application area of project management is the construction industry. Construction projects involve two main parties: the contractor and the client receiving the project deliverables provided by the contractor. The concept of partnering tries to overcome the adversarial relation between contractor and client, which still tends to prevail in many construction projects. In Chap. 46 Hemanta Doloi examines key factors that are crucial for successful partnering and draws conclusions from a survey conducted in the Australian construction industry. Chapter 47, written by Xianhai Meng, deals with incentive mechanisms, which are frequently used to enhance project performance, especially in the construction industry. The author discusses different kinds of incentives and disincentives that are related to project goals such as time, cost, quality, and safety. A case study of a road construction project gives insight into the practical application of incentive mechanisms. Project complexity is a prominent cause for project failure. Hence, it is vitally important for managers to know about sources of complexity. In Chap. 48 Marian Bosch-Rekveldta, Hans Bakker, Marcel Hertogh, and Herman Mooi identify drivers of complexity. Based on a literature research and six case studies analyzing the complexity of engineering projects, they provide a framework for evaluating project complexity. The framework comprises technical, organizational, and external sources of project complexity.

Part XVI deals with project risk management. Since the importance of projects has grown and revenues from project work may constitute a considerable share of a firm's total income, managing project risk is vitally important as it helps to identify threats and to mitigate potential damage. In Chap. 49, Chao Fang and Franck Marle outline a framework for project risk management, which considers not only single risks separately but also interactions between risks. The authors show how interactions can be captured in a matrix-based risk network and provide a quantitative method to analyze such a network. Chapter 50 is concerned with risk management for software projects. Paul Bannerman reviews empirical research on the application of risk management in practice, the effectiveness of risk management, and factors that hinder or facilitate the implementation of risk management. He describes different perspectives on risk management in order to show the wide range of approaches and to identify avenues for further research. An important goal of risk management is to identify risks and to decide on the risks that should be mitigated. This decision is frequently based on a ranking of the identified risks. In Chap. 51 Stefan Creemers, Stijn Van de Vonder, and Erik Demeulemeester survey the different ranking methods that were proposed in the literature. In particular, they consider so-called ranking indices that provide a ranking of activities or risks based on their impact on the project objectives. They show that the ranking methods may differ in their outcome and evaluate their performance with a focus on the risk of project delay.

The last **Area G** proves evidence for the relevance of concepts developed in the preceding parts of this handbook to the practice of project management and scheduling. The area covers different domains beyond proper project scheduling and puts the concepts treated in the previous parts into the perspective of real-life project management. It includes chapters on project scheduling applications, case studies, and project management information systems.

Part XVII collects six industrial applications of resource-constrained project scheduling, where different models and methods presented in previous chapters are put into practice. In particular, test, production, and workflow scheduling problems are considered. Chapter 52, written by Jan-Hendrik Bartels and Jürgen Zimmermann, reports on the problem of scheduling destructive tests in automotive R&D projects. The planning objective consists in minimizing the number of required experimental vehicles. The problem is modeled as a multi-mode resource-constrained project scheduling problem with renewable and storage resources, in which the required stock must be built up before it can be consumed. In addition to different variants of a priority-rule based heuristic, an activity-list based genetic algorithm is proposed. Both heuristic approaches prove suitable for solving large-scale practical problem instances. In Chap. 53 Roman Čapek, Přemysl Šůcha, and Zdeněk Hanzálek describe a scheduling problem with alternative process plans, which arises in the production of wire harnesses. In such a production process, alternative process plans include production operations that can be performed in different ways, using fully or semi-automated machines. A mixed-integer linear programming model for a resource-constrained project scheduling problem with generalized precedence relations, sequence-dependent setup times, and alternative activities is presented. Furthermore, a heuristic schedule-construction procedure with an unscheduling step is proposed, which can be applied to large problem instances. Chapter 54 is concerned with the scheduling of jobs with large computational requirements in grid computing. An example of such jobs are workflow applications, which comprise several precedence-related computation tasks. A computer grid is a large-scale, geographically distributed, dynamically reconfigurable, and scalable hardware and software infrastructure. Marek Mika and Grzegorz Waligóra present three models for scheduling the computation and transmission tasks in grids, differing in their assumptions with respect to the workflow applications and computer networks. For the models with distributed resources and sequence-dependent setup times, resource allocation and scheduling algorithms are presented. For the model in which transmission tasks compete for scarce network resources it is shown how a feasible resource allocation can be determined. Chapter 55 by Haitao Li considers make-or-buy and supplier selection problems arising in conjunction with the scheduling of operations in make-to-order supply chains. A multi-mode resource-constrained project scheduling problem is formulated to minimize the total supply chain cost, in which synergies and interactions between sourcing and scheduling decisions are captured. The total supply chain cost involves the total fixed cost, cost of goods sold, and total pipeline stock cost and depends on the selected activity modes. The proposed solution algorithm draws on the hybrid Benders decomposition framework exposed in Chap. 27. The relaxed master problem (RMP) covers the assignment decisions, whereas the subproblem (SP) is concerned with the scheduling of the operations. The feasibility of an optimal RMP solution is checked by solving the respective SP. If the SP is feasible, an optimal solution has been found; otherwise, the algorithm identifies some cause of infeasibility and adds respective cuts to the RMP, which is then solved again. A numerical example is discussed to demonstrate the scope and depth of decision-support offered by the solutions of the model for purchasing and program managers. In Chap. 56 Arianna Alfieri and Marcello Urgo apply a project scheduling approach to make-to-order systems for special-purpose machinery like instrumental goods or power generation devices, in which products are assembled in the one-of-a-kind production mode. They present a resource-constrained project scheduling problem with feeding precedence relations and work content constraints and explain its application to a real-world case of machining center production. In Chap. 57 Matthew Colvin and Christos Maravelias apply multi-stage stochastic programming to the development process of new drugs. The problem consists in scheduling a set of drugs, each of which has to undergo three trials. If one trial fails, the development of the related drug is canceled. The required resources are limited and the objective is to maximize the expected net present value of the project. After an introduction to stochastic programming and endogenous observations of uncertainty, a mixed-integer multi-stage stochastic programming model is presented. Some structural properties of the problem are discussed and three solution methods including a branch-and-cut algorithm are developed.

Part XVIII presents two case studies in project scheduling. In Chap. 58 Maurizio Bevilacqua, Filippo Ciarapica, Giovanni Mazzuto, and Claudia Paciarotti combine concepts of robust project scheduling and multi-criteria project scheduling to tackle a construction project for an accommodation module of an oil rig in the Danish North Sea. To guarantee an efficient use of the resources, the project management identified the minimization of the project duration and the leveling of the manpower resources as primary goals. Using historical data from 15 past projects, the means and the standard deviations of the activity durations could be estimated with sufficient accuracy. To obtain a robust baseline schedule for the project, project buffers and feeding buffers were inserted in the schedule according to the lines of Goldratt's Critical Chain methodology. Compared to the traditional CPM method, the presented robust goal programming approach was able to reduce the project duration by 14 % and to improve the resource utilization by more than 40 %. In Chap. 59 Jiuping Xu and Ziqiang Zeng consider a multi-criteria version of the discrete time-cost tradeoff problem, which is called the discrete time-cost-environment-tradeoff problem DTCETP. They assume that normal activity durations are represented as triangular fuzzy numbers and that for each period there exists a limit on the total cost incurred by the processing and crashing of activities. This cash flow constraint can be modeled as a renewable resource whose capacity coincides with the cost limit. The capacity is taken up according to the requirements of alternative execution modes. In sum, the problem can be formulated as a fuzzy multi-criteria multi-mode resource-constrained project scheduling problem. Four objective functions are taken into account: the total project cost, the project duration, the total crashing costs of activities, and the quantified environmental impact of the project. Xu and Zeng develop an adaptive hybrid genetic algorithm for this problem and describe its application to the Jinping-II hydroelectric construction project on the Yalong River in the Sichuan-Chongqing region. Both the input data of the case study and the computed schedule are provided. The performance of the algorithm is evaluated based on a sensitivity analysis with respect to the objective weights and the results obtained with two benchmark heuristics.

Project management information systems PMIS play a crucial role in the transfer of advanced project management and scheduling techniques to professional project management. **Part XIX** addresses the question of the actual contribution of PMIS on the project performance, studies the effects of PMIS on decision making in multi-project environments, and investigates the project scheduling capabilities of commercial PMIS.

Based on a PMIS success model and a survey conducted among project managers, Louis Raymond and François Bergeron in Chap. 60 empirically assess the impact of PMIS on decision makers and project success. Their model comprises five constructs: the quality of the PMIS, the quality of the PMIS information output, the use of the PMIS, the individual impacts of the PMIS, and the impacts of the PMIS on project success. Each construct is measured using several criteria. Structural equation modeling with the partial least squares method is used to analyze the relationships between the different dimensions and to test the validity of six research hypotheses. The results obtained show that the use of PMIS in professional project management significantly contributes to the efficiency and effectiveness of individual project managers and to the overall project performance. Chapter 61 presents a related study in which Marjolein Caniëls and Ralph Bakens focus on the role of PMIS in multi-project environments, where project managers handle multiple concurrent but generally less complex projects. After a survey of the literature on multi-project management and PMIS the research model is introduced, which contains six constructs: the project overload, the information overload, the PMIS information quality, the satisfaction with PMIS, the use of PMIS information,

Table 2 Overview of project scheduling problems treated in the handbook, respective acronyms used in the literature, and three-field notations of Brucker et al. (1999)

Chaps.	Project scheduling problem	Acronym	Three-field notation
1 – 4	Resource-constrained project scheduling problem	RCPSP	$PS \mid prec \mid C_{max}$
5 – 7	Resource-constrained project scheduling problem with generalized precedence relations	RCPSP/max	$PS \mid temp \mid C_{max}$
8	Resource-constrained project scheduling problem with time-varying resource requirements and capacities	RCPSP/t	$PSt \mid prec \mid C_{max}$
9	Project scheduling problems with storage resources		$PSs \mid temp \mid C_{max}$
10	Discrete-continuous resource-constrained project scheduling problem	DCRCPSP	$PSc \mid prec \mid C_{max}$
11	Resource-constrained project scheduling problem with partially renewable resources	RCPSP/π	$PSp \mid prec \mid C_{max}$
12	Integer preemptive resource-constrained project scheduling problem with limited number of interruptions per activity	Maxnint_ PRCPSP	$PS \mid prec, l\text{-}pmtn/int \mid C_{max}$
13	Continuous preemptive resource-constrained project scheduling problem with generalized precedence relations	PRCPSP/max	$PS \mid temp, pmtn \mid C_{max}$
14	Resource-constrained project scheduling problem with discounted cash flows	RCPSPDC	$PS \mid prec, \overline{d} \mid \Sigma c_i^F \beta^{C_i}$
15	Resource availability cost problem	RACP	$PS\infty \mid prec, \overline{d} \mid \Sigma c_k \max r_{kt}$
16	Resource availability cost problems	RACP, RACPT	$PS\infty \mid prec, \overline{d} \mid \Sigma c_k \max r_{kt}$, $PS\infty \mid prec \mid \Sigma c_k \max r_{kt} + wT$
17	Resource leveling problems	RLP	$PS\infty \mid temp, \overline{d} \mid \Sigma c_k \Sigma r_{kt}^2$, $PS\infty \mid temp, \overline{d} \mid \Sigma c_k \Sigma o_{kt}$, and $PS\infty \mid temp, \overline{d} \mid \Sigma c_k \Sigma \Delta r_{kt}$
18	Resource leveling problem	RLP	$PS\infty \mid prec, \overline{d} \mid \Sigma c_k \Sigma r_{kt}^2$
19	Multi-objective time- and resource-constrained project scheduling problems	MOPSPs, MORCPSPs	$PS\infty \mid prec \mid mult$, $PS \mid prec \mid mult$
20	Multi-objective resource-constrained project scheduling	MORCPSPs	$PS \mid prec \mid mult$

(continued)

Table 2 (continued)

Chaps.	Project scheduling problem	Acronym	Three-field notation
21	Multi-modal resource-constrained project scheduling problems		$MPS \mid prec \mid f$
22	Multi-mode resource-constrained project scheduling problem	MRCPSP	$MPS \mid prec \mid C_{max}$
23	Multi-mode capital-constrained net present value problem	MNPV	$MPSs \mid prec \mid \Sigma c_i^F \beta^{C_i}$
24	Project scheduling problem with work content constraints		$PSf \mid prec \mid C_{max}$
25	Project staffing and scheduling problems		$PSS \mid temp \mid f$
26	Multi-skill project scheduling problem	MSPSP	$PSS\infty \mid prec \mid C_{max}$
27	Project scheduling with multi-purpose resources	PSMPR	$PSS \mid temp \mid staff$
28	Preemptive multi-skill project scheduling problem		$PSS \mid prec, pmtn \mid C_{max}$
29	Discrete time-cost tradeoff problem (deadline version)	d-DTCTP	$MPS\infty \mid prec, \overline{d} \mid \Sigma c_i(p_i)$
	Discrete time-cost tradeoff problem with irregular starting time costs		$MPS\infty \mid prec, \overline{d} \mid f$
30	Discrete time-cost tradeoff problem with time-switch constraints	d-DTCTP-tsc	$MPS\infty \mid prec, \overline{d}, cal \vert \Sigma c_i(p_i)$
	Discrete time-cost tradeoff problem with net present value optimization	d-DTCTP-npv	$MPS\infty \mid prec, \overline{d} \mid \Sigma c_i^F \beta^{C_i}$
31	Basic multi-project scheduling problem	BMPSP	$PS \mid mult, prec \mid f$
32	Decentralized multi-project scheduling problem	DRCMPSP	
33	Multi-criteria project portfolio selection problem		
34	Project selection, scheduling, and staffing with learning problem	PSSSLP	
35	Stochastic net present value problem	SNPV	$PS \mid prec, p_i = sto \mid \Sigma c_i^F \beta^{C_i}$
36	Stochastic discrete time-cost tradeoff problem (budget version)	b-SDTCTP	$MPS\infty \mid prec, bud, p_i = sto \mid C_{max}$
37	Stochastic resource-constrained project scheduling problem	SRCPSP	$PS \mid prec, p_i = sto \mid C_{max}$
38	Markovian multi-criteria multi-project resource-constrained project scheduling problem		$MPSm, 1, 1 \mid mult, prec, bud, p_i = sto, c_i = sto, Poi \mid mult$

(continued)

Table 2 (continued)

Chaps.	Project scheduling problem	Acronym	Three-field notation
39	Robust discrete time-cost tradeoff problem		$MPS\infty \mid prec, \overline{d}, c_i = unc \mid \Sigma c_i(p_i)$
40	(Absolute regret) Robust resource-constrained project scheduling problem	AR-RCPSP	$PS \mid prec, p_i = unc \mid rob$
41	Temporal analysis under interval uncertainty		$PS\infty \mid prec, p_i = unc \mid f$ with $f \in \{ES_i, LS_i, TF_i\}$
42	Fuzzy time-cost tradeoff problem (deadline version)		$MPS\infty \mid prec, \overline{d}, p_i = fuz \mid \Sigma c_i(p_i)$
52	Multi-mode resource-constrained project scheduling problem with storage resources		$MPSs \mid temp, \overline{d} \mid \Sigma c_k \max r_{kt}$
53	Resource-constrained project scheduling problem with generalized precedence relations, sequence dependent setup times, and alternative activities	RCPSP-APP	$PS \mid temp, s_{ij}, nestedAlt \mid C_{max}$
54	Multi-mode resource-constrained project scheduling problems	MRCPSP	$MPS \mid prec \mid C_{max}$
55	Multi-mode resource-constrained project scheduling problem		$MPS \mid prec, \overline{d} \mid mac$
56	Resource constrained project scheduling problem with feeding precedence relations and work content constraints		$PSft \mid feed \mid C_{max}$
57	Stochastic net present value problem in which the set of activities to be executed is stochastic		$PS \mid prec, act = sto \mid \Sigma c_i^F \beta^{C_i}$
58	Robust multi-criteria project scheduling problem		$PS \mid prec, p_i = sto \mid C_{max}/\Sigma r_{kt}^2$
59	Fuzzy multi-criteria multi-mode project scheduling problem	DTCETP	$MPS \mid prec, \overline{d}, bud, p_i = fuz \mid mult$

and the quality of decision making. Based on the results of a survey among project managers, several hypotheses on the relationships between the constructs are tested using the partial least square method. It turns out that project and information overload are not negatively correlated with PMIS information quality and that the quality and use of PMIS information are strongly related to the quality of decision making. In the final Chap. 62, Philipp Baumann and Norbert Trautmann experimentally assess the performance of eight popular PMIS with respect to their project scheduling capabilities. Using the more than 1.500 KSD-30, KSD-60, and KSD-120 instances of the resource-constrained project scheduling problem RCPSP from the PSPLIB library, the impact of different complexity parameters and priority rules on the resulting project durations is analyzed. The results indicate that for the project duration criterion, the scheduling performances of the software packages differ significantly and that the option of selecting specific priority rules generally leads to schedules of inferior quality as compared to PMIS that do not offer this feature.

Table 2 gives an overview of the different types of project scheduling problems treated in this book. In the literature many of those problems are commonly designated by acronyms, which are provided in the third column of the table. The last column lists the respective designators of the (extended) three-field classification scheme for project scheduling problems proposed by Brucker et al. (1999). The notation introduced there and the classification scheme, which are used in different parts of this handbook, are defined in the list of symbols, which is included in the front matter of this book.

References

Artigues C, Demassey S, Néron E (eds) (2008) Resource-constrained project scheduling: models, algorithms, extensions and applications. Wiley, Hoboken

Brucker P, Drexl A, Möhring R, Neumann K, Pesch E (1999) Resource-constrained project scheduling: notation, classification, models, and methods. Eur J Oper Res 112:3–41

Dinsmore PC, Cooke-Davies TJ (2005) Right projects done right: from business strategy to successful project implementation. Wiley, San Francisco

Demeulemeester EL, Herroelen WS (2002) Project scheduling: a research handbook. Kluwer, Dordrecht

Hartmann S, Briskorn D (2008) A survey of deterministic modeling approaches for project scheduling under resource constraints. Eur J Oper Res 207:1–14

Józefowska J, Węglarz J (eds) (2006) Perspectives in modern project scheduling. Springer, New York

Kerzner H (2013) Project management: a systems approach to planning, scheduling, and controlling. Wiley, Hoboken

Klein R (2000) Scheduling of resource-constrained projects. Kluwer, Boston

Lewis JP (1997) Fundamentals of project management. Amacom, New York

Project Management Institute, Inc. (2013) A guide to the project management body of knowledge (PMBOK®Guide). PMI, Newtown Square

Turner JR (2009) The handbook of project-based management: leading strategic change in organizations. McGraw-Hill, New York

Part I
The Resource-Constrained Project Scheduling Problem

Chapter 1
Shifts, Types, and Generation Schemes for Project Schedules

Rainer Kolisch

Abstract Schedule generation schemes are the backbone of heuristics to solve project scheduling problems. In this chapter we introduce the two schedule generation schemes for the classical resource constrained project scheduling problem, the serial and the parallel schedule generation scheme. We characterize them according to the types of schedule they generate and discuss variants of the schedule generation schemes in order to deal with extensions such as general precedence constraints and stochastic activity durations.

Keywords Makespan minimization • Project scheduling • Resource constraints • Schedule generation schemes

1.1 Introduction

In this chapter we discuss schedule generation schemes (SGSs) which are employed in order to construct feasible schedules for the resource-constrained project scheduling problem (RCPSP). Two schedule generation schemes are available in the literature, the serial SGS and the parallel SGS. Schedule generation schemes are generalizations of list scheduling as familiar from machine scheduling (see Błażewicz et al. 2007). SGSs are heuristics for generating feasible project schedules. The latter can be but are not necessarily optimal. In fact, we show that it might not be possible to obtain an optimal schedule when employing the parallel SGS. Schedule generation-schemes have become the backbone for the majority of simple and more advanced heuristics to solve the RCPSP. In the remainder of this chapter we first provide a brief introduction to the resource-constrained project scheduling problem and to schedules in Sect. 1.2. Next, in Sect. 1.3 we discuss shifts as a means of transforming schedules. Then, in Sect. 1.4 we define different types of schedules. In Sect. 1.5 we describe the two schedule generation schemes and characterize them with respect to the type of schedules they generate. Afterwards,

R. Kolisch (✉)
TUM School of Management, Technische Universität München, Munich, Germany
e-mail: rainer.kolisch@tum.de

C. Schwindt, J. Zimmermann (eds.), *Handbook on Project Management and Scheduling Vol. 1*, International Handbooks on Information Systems,
DOI 10.1007/978-3-319-05443-8_1

in Sect. 1.6 we address schedule generation schemes for the simple assembly line balancing problem as a special case of the RCPSP and for generalizations of the RCPSP with minimum and maximum time lags or stochastic activity durations. Finally, in Sect. 1.7 we provide conclusions.

1.2 The Resource-Constrained Project Scheduling Problem

The RCPSP is probably the most studied optimisation problem in project scheduling. It can be described as follows. A single project which consists of n real activities has to be scheduled subject to precedence and renewable resource constraints such that the time required for performing all activities, the makespan, is minimized. The project is depicted as an activity-on-node (AoN) network with node set $V = \{0, 1, \ldots, n+1\}$ where each node represents an activity and nodes 0 and $n+1$ are dummy activities representing the milestone "project start" and "project finish". A precedence constraint between two activities i and j, $i, j \in V$, $i \neq j$, is represented by arc (i, j). We consider simple finish-to-start precedence constraints which imply that the successor activity j must not be started before the predecessor activity is finished. N denotes the set of all arcs of a project. In order to be processed, activity j requires r_{jk} units of renewable resource k for every period of its duration p_j. The set of all renewable resources required to undertake the project is $\mathscr{R}$. Renewable resource $k \in \mathscr{R}$ has an availability of R_k in each period. Since other types of resources, such as nonrenewable, doubly constrained or partially renewable resources (see Neumann et al. 2002), are not relevant for the RCPSP, we henceforth speak only of resources, meaning renewable resources. Resource constraints necessitate that for each resource $k \in \mathscr{R}$ and each period the sum of the resource demand of processed activities must not be greater than resource availability R_k. The optimization problem is to define a start time for each activity which respects precedence and resource constraints and minimizes the makespan of the project.

A schedule $S = (S_0, S_1, \ldots, S_{n+1})$ defines for each activity $j \in V$ the start time S_j. Assuming without loss of generality that activity 0 starts at time 0 and that the duration of the activities are integer multiples of the period length, an activity always starts at the beginning of a period. An activity j with duration p_j which starts in period t is processed in periods $S_j + 1, \ldots, S_j + p_j$. We take into consideration the resource constraints of a period t at time instant $t - 1$ which defines the start of period t. In Fig. 1.1, for example, activity 1 with duration $p_1 = 2$ and resource demand $r_{11} = 1$ with respect to resource 1 is processed in periods 1 and 2; it starts at time $S_2 = 0$ and we check the availability of the capacity of periods 1 and 2 at times 0 and 1. From a given schedule S we can derive for time instant t the set of activities which are in progress $\mathscr{A}(S, t) = \{j \in V \mid S_j + 1 \leq t < S_j + p_j\}$. In Fig. 1.1 activity 1 is in progress at time instants 0 and 1.

A schedule S is precedence feasible if $S_i + p_i \leq S_j$ holds for each $(i, j) \in N$. A schedule S is resource feasible if for any resource $k \in \mathscr{R}$ and at any point in time

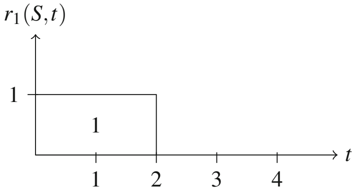


Fig. 1.1 Example of a scheduled activity

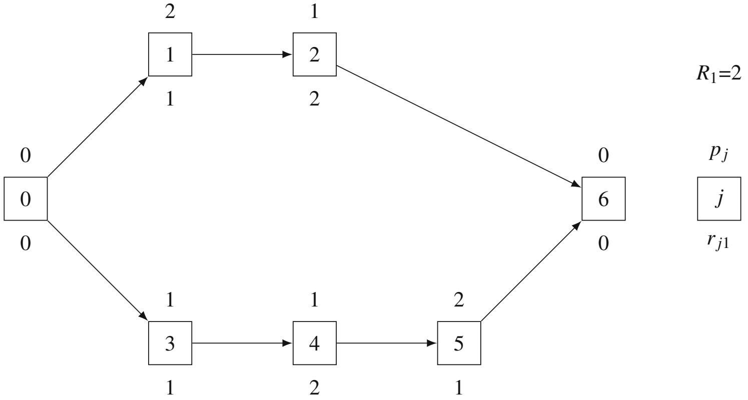


Fig. 1.2 Example instance of the RCPSP

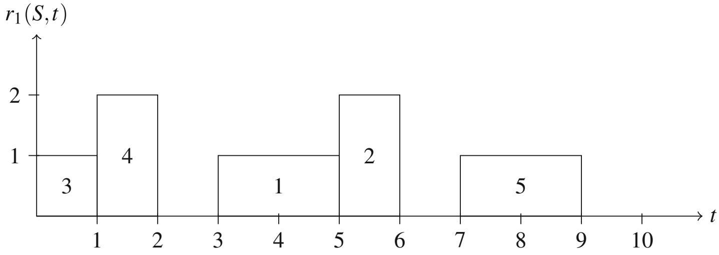


Fig. 1.3 Feasible schedule for the example instance

$t = 0, \ldots, T-1$ the capacity required by the activities in progress $r_k(S,t) = \sum_{j\in\mathscr{A}(S,t)} r_{jk}$ does not exceed the available capacity R_k. T denotes an upper bound on the project makespan. A simple upper bound for the project makespan is $\sum_{j\in V} p_j$. The optimization problem of the RCPSP is to find a precedence and resource feasible schedule for the project which minimizes the makespan S_{n+1}. The RCPSP is well-known to be $\mathscr{NP}$-hard (see Błażewicz et al. 1983). Figure 1.2 provides an example instance of the RCPSP with $n = 5$ real activities and $| \mathscr{R} |= 1$ resource with capacity $R_1 = 2$. Figure 1.3 is an example of a feasible schedule for the example instance.

An objective function $f(S)$ maps a schedule S into $f(S)$. An objective function $f(S)$ is regular if it is nondecreasing in the start time of activities, that is, if we have two schedules S and S' with $S'_j \leq S_j$ and $S'_i = S_i$ for $i \in V \setminus \{j\}$ then $f(S') \leq f(S)$ holds. The project makespan S_{n+1} as well as the sum of the activity start times $\sum_{j \in V} S_j$ are well known regular objective functions.

1.3 Shifts

A shift is a movement of an activity j currently scheduled at S_j to an earlier or later start time $S'_j \neq S_j$. With a sequence of shifts we can transform schedules. We can define the following type of shifts (see Sprecher et al. 1995).

Definition 1.1. A left shift of an activity j is an operation on a feasible schedule S which derives a feasible schedule S' such that $S'_j < S_j$ and $S'_i = S_i$ for $i \in V \setminus \{j\}$.

Remark. If the objective function is regular, then applying a left shift to a schedule S always leads to a schedule S' which dominates S with respect to the objective function.

Definition 1.2. A one-period left shift of an activity j is a left shift of activity j for which $S_j - S'_j = 1$ holds.

Definition 1.3. A local left shift of an activity j is a left shift of activity j which is obtainable by one or more successively applied one-period left shifts of activity j.

Remark. Each intermediate schedule derived by a local left shift is by definition feasible.

Definition 1.4. A global left shift of an activity j is a left shift of activity j which is not obtainable by a local left shift.

Remark. By definition, for a global left shift $S_j - S'_j > 1$ holds. Furthermore, at least one of the intermediate schedules generated in the course of the global left shift is not resource feasible.

In order to illustrate the different shifts consider the schedule in Fig. 1.4 which can be obtained from the feasible schedule provided in Fig. 1.3 performing a one-period left shift of activity 2, a one-period left shift of activity 3 and a local left shift consisting of two periods of activity 6.

1.4 Schedule Types

We distinguish three schedule types: semi-active, active, and non-delay schedules.

Definition 1.5. A semi-active schedule is a feasible schedule where none of the activities can be locally left shifted.

Fig. 1.4 Semi-active schedule for the example instance

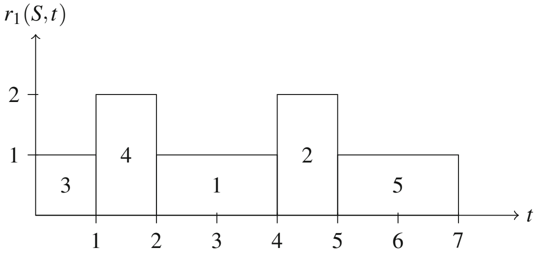


Fig. 1.5 Active and unique optimal schedule for the example instance

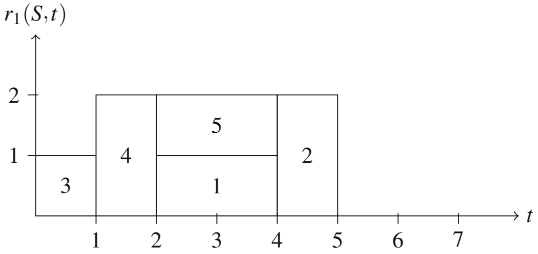


Figure 1.4 shows a semi-active schedule for the example instance. Clearly, none of the activities can be locally left shifted any more. The exact procedure of Sprecher (2000) enumerates schedules based on semi-active schedules.

Definition 1.6. An active schedule is a feasible schedule where none of the activities can be locally or globally left shifted.

Figure 1.5 provides an active schedule for the example instance which is derived from the semi-active schedule of Fig. 1.4 by undertaking a global three-period left shift of activity 6. Note that the two intermediate schedules $S' = (0, 2, 4, 0, 1, 4, 6, 2)$ and $S'' = (0, 2, 4, 0, 1, 3, 5)$ are not feasible and thus we have not performed a local left shift. The resulting schedule is the unique optimal schedule for the example instance (see Sprecher et al. 1995). Most exact solution procedures such as the one from Demeulemeester and Herroelen (1992, 1997) enumerate active schedules.

Before we turn to the definition of a non-delay schedule, we have to introduce the concept of a unit-time-duration RCPSP (UTDRCPSP). Any RCPSP can be transformed into a UTDRCPSP by splitting each activity j with duration p_j into a series of p_j activities with duration 1. The unit-time-duration activities resulting from activity j are serially ordered in the sense that there is a finish-start precedence constraint between the first and the second, the second and the third unit-time duration activity, and so on.

Definition 1.7. A non-delay schedule is a feasible schedule where the schedule of the corresponding UTDRCPSP is active.

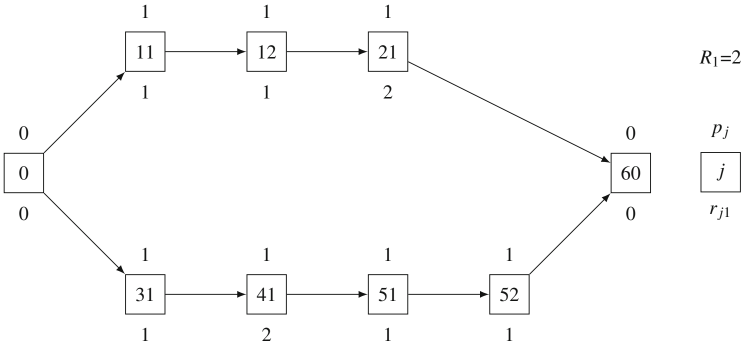


Fig. 1.6 UTDRCPSP of the example instance

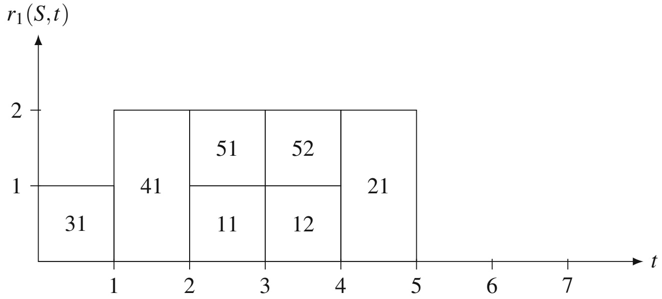


Fig. 1.7 Schedule of the UTDRCPSP corresponding to the active schedule of the RCPSP

We transform the example RCPSP-instance into the UTDRCPSP-instance shown in Fig. 1.6, where activity j is transformed into unit-time-duration activities $j1, \ldots, jp_j$.

The schedule of the UTDRCPSP corresponding to the active and optimal schedule of the RCPSP is given in Fig. 1.7. For this schedule we can globally left shift activity 11 from $S_{11} = 2$ to $S'_{11} = 0$. If we transform the resulting UTD-schedule into the schedule of the corresponding RCPSP, we obtain the non-delay schedule presented in Fig. 1.8. Note that this non-delay schedule is not optimal.

Theorem 1.1. *Let* **S** *denote the set of schedules,* **FS** *the set of feasible schedules,* **SAS** *the set of semi-active schedules,* **AS** *the set of active schedules, and* **NDS** *the set of non-delay schedules, then* $\mathbf{NDS} \subseteq \mathbf{AS} \subseteq \mathbf{SAS} \subseteq \mathbf{FS} \subseteq \mathbf{S}$ *holds.*

Obviously, the set of schedules with the smallest cardinality which contains a solution with optimal regular objective function is the active set (see Sprecher et al. 1995).

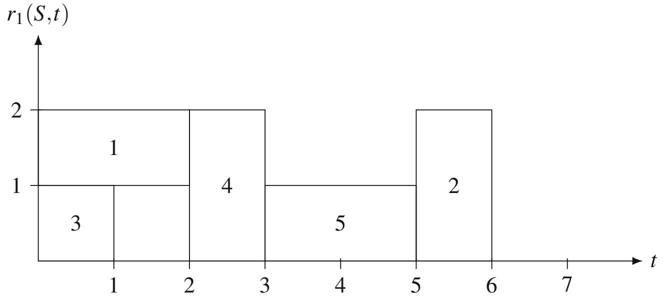


Fig. 1.8 Non-delay schedule of the example problem

1.5 Schedule Generation Schemes for the RCPSP

Schedule generation schemes (SGS) are algorithms which construct precedence and resource feasible schedules for the RCPSP. SGSs are the backbone of heuristics for the RCPSP. For both schemes we utilize an upper bound on the project makespan T and latest start times LS derived from T by backward longest path calculation (see Demeulemeester and Herroelen 2002).

1.5.1 Serial Schedule Generation Scheme

The serial schedule generation scheme performs n iterations. In each iteration one activity is scheduled at it earliest precedence and resource feasible start time. Hence, for a scheduled activity there are no further local or global left shifts possible. In order to formally describe the serial SGS, we have to introduce some additional notation. Let $\mathscr{C}_\mu$ be the set of activities which have been scheduled up to iteration μ. Employing $\mathscr{C}_\mu$ we can define the set of scheduled activities which are active at time instant t as $\mathscr{A}(\mathscr{C}_\mu, t) = \{j \in \mathscr{C}_\mu \mid S_j \leq t < S_j + p_j\}$. Now, let $\tilde{R}_k(t) = R_k - \sum_{j \in \mathscr{A}(\mathscr{C}_\mu, t)} r_{jk}$ be the remaining capacity of resource k at time t, taking into account the resource demand of the activities which have been scheduled so far. Finally, let $\mathscr{D}_\mu$ be the decision set containing all activities eligible for scheduling in iteration μ. In order to be eligible, the activity itself must not have been scheduled but all its predecessors have to be scheduled, therefore $\mathscr{D}_\mu = \{j \in V \setminus \mathscr{C}_\mu \mid Pred(j) \subseteq \mathscr{C}_\mu\}$. We now can formally state the serial SGS as follows:

```
**Initialization**: $S_0 := 0, \mathscr{C}_0 := \{0\}$.
**Iteration**: For $\mu = 1$ to $n$ do
Update $\mathscr{D}_\mu$ and $\tilde{R}_k(t)$ for all $k \in \mathscr{R}$ and $t = 0, \ldots, T-1$.
Select one $j \in \mathscr{D}_\mu$.
$\overline{ES}_j := \max_{h \in Pred(j)}\{S_h + p_h\}$.
```

Table 1.1 Earliest start times, latest start times and slack of activities

j	1	2	3	4	5
ES_j	0	2	0	1	2
LS_j	4	6	3	4	5
$LS_j - ES_j$	4	4	3	3	3

Table 1.2 Iterations of the serial SGS for the example instance

μ	1	2	3	4	5
$\mathscr{D}_\mu$	$\{1,3\}$	$\{1,4\}$	$\{1,5\}$	$\{2,5\}$	$\{2\}$
j	3	4	1	5	2

```
6: $S_j := \min\{t \mid ES_j \leq t \leq LS_j, r_{jk} \leq \tilde{R}_k(\tau)$ for all $k \in \mathscr{R}$
   and $\tau = t, \ldots, t + p_j - 1\}$.
7: $\mathscr{C}_\mu := \mathscr{C}_{g-1} \cup \{j\}$.
8: $S_{n+1} := \max_{j \in Pred(n+1)}\{S_j + p_j\}$.
```

Initialization in line 1 assigns start time 0 to the dummy start activity and initializes the set of already scheduled activities with the dummy start activity. After that, n iterations take place. In each iteration one activity of the decision set is selected and scheduled as early as possible with respect to precedence and resource constraints. In line 3 the decision set and the remaining capacity of the resources are updated. Then, in line 4 one activity j from the decision set is selected and its earliest start time $\overline{ES}_j$ is calculated in line 5 as the earliest finish time of all immediate predecessors. Note that this earliest start time is calculated taking into account the precedence and resource feasible start times of all predecessors of activity j. In contrast, classical forward pass calculation for determining earliest start times ES_j (see Demeulemeester and Herroelen 2002) only take precedence constraints into account. Starting from $\overline{ES}_j$ the earliest resource feasible start time is calculated in line 6 and afterwards j is added to the set of scheduled activities in line 7. At the end of the iteration, all n non dummy activities have been scheduled and, in line 8, the dummy end activity is assigned the maximum finish time of its predecessors as start time.

Let us illustrate the serial SGS by applying it to the example problem given in Fig. 1.2 employing the minimum slack (MSLK) priority rule. Table 1.1 provides for each activity j the precedence based early start time ES_j, the precedence based latest start time LS_j calculated by backward pass calculation from the upper bound of the project makespan $T = \sum_{j \in V} p_j = 7$ and the slack $LS_j - ES_j$. In case of ties we select the activity with the smallest number. Table 1.2 provides the decision set $\mathscr{D}_\mu$ and the selected activity j for each iteration $\mu = 1, \ldots, 5$. The resulting schedule is the active and optimal schedule given in Fig. 1.5.

An efficient implementation of the serial schedule generation scheme with a time complexity of $\mathscr{O}(n^2 \cdot \mid \mathscr{R} \mid)$ is provided in Kolisch and Hartmann (1999). The implementation utilizes the fact that when searching for the earliest resource feasible start time S_j in line 6 not every time t has to be checked. Instead, the implementation checks resource feasibility only at the finish time of activities, thereby reducing the time complexity from $\mathcal{O}(n \cdot T \cdot \mid \mathcal{R} \mid)$ to $\mathcal{O}(n^2 \cdot \mid \mathcal{R} \mid)$

Kolisch (1996) has shown that the serial scheduling scheme always generates a feasible solution if one exists and that the resulting schedule is an active schedule. This property stems from the fact that the serial schedule generation scheme schedules a selected activity at the earliest precedence and resource feasible time and, hence, that no local or global left shifts are possible.

Instead of selecting an activity from the decision set based on its priority value, a precedence feasible activity list can be employed. An activity list $\ell = (j_1, j_2, \ldots, j_n)$ is determined before the start of the algorithm. Precedence feasible means that for an activity j_i in the list each immediate predecessor activity $Pred(j_i)$ has to be before j_i on the list, that is $Pred(j_i) \subseteq \{j_1, \ldots, j_{i-1}\}$ holds (see Hartmann 1998). An activity list ℓ can be obtained by recording the sequence activities are selected when processing the serial schedule generation scheme. Also, ℓ can be generated by employing a feasible schedule and sequencing the activities in the order of increasing start times (see Debels et al. 2006). In case of equal start times, any order of the activities in question is feasible. If we record the sequence of the activities selected when applying the serial SGS with the MSLK priority rule as given in Table 1.2, we obtain the activity list $(3, 4, 1, 5, 2)$. If we derive an activity list based on the start times of the schedule given in Fig. 1.8, activity list $(3, 1, 4, 5, 2)$ results. When employing the serial SGS with an activity list ℓ, we do not need decision set $\mathcal{D}_\mu$ any more. Furthermore, we replace "Select one $j \in \mathcal{D}_\mu$" with "$j = j_\mu$" in line 4 (see Kolisch and Hartmann 1999).

The serial schedule generation scheme applied to activity lists is employed by most metaheuristics for the RCPSP (see Kolisch and Hartmann 2006). There, a solution is encoded as an activity list and by means of a schedule generation scheme mapped onto a feasible schedule. Along with the activity list, Kolisch and Hartmann (1999) present different schedule representations. Debels et al. (2006) point out that different activity list may by mapped onto the same schedule and propose a standardized random key representation where each activity j is assigned a not necessarily unique integer x_j between 1 and n. When searching for the next activity to be scheduled, the not yet scheduled precedence feasible activity j with smallest x_j is selected. For a given schedule a unique standardized random key representation can be derived by assigning x values according to the start times of the activities where activities with the same start time receive the same x value. The standardized random key representation for the optimal schedule given in Fig. 1.5 is $x = (3, 5, 1, 2, 3)$.

1.5.2 Parallel Schedule Generation Scheme

The parallel schedule generation scheme is time oriented. Each iteration μ has a unique and monotonically increasing schedule time t_μ. The algorithm starts as many precedence and resource feasible activities as possible at t_μ. Due to the time orientation, the following sets are defined as depending on t_μ. The set of completed activities $\mathscr{C}_\mu$ is comprised of all activities which have been scheduled such that the finish time is less than or equal to t_μ, that is $\mathscr{C}_\mu = \{j \in V \mid S_j + p_j \le t_\mu\}$. The set of active activities $\mathscr{A}_\mu$ is made up of the activities which have been scheduled and are in process at t_μ, that is $\mathscr{A}_\mu := \{j \in V \mid S_j \le t_\mu < S_j + p_j\}$. The remaining capacity of resource k at time t_μ is $\tilde{R}_k(t_\mu) = R_k - \sum_{j \in \mathscr{A}_\mu} r_{jk}$. Finally, we define the decision set $\mathscr{D}_\mu = \{j \in V \setminus \{\mathscr{C}_\mu \cup \mathscr{A}_\mu\} \mid Pred(j) \subseteq \mathscr{C}_\mu$ and $r_{jk} \le \tilde{R}_k(t_\mu)$ for all $k \in \mathscr{R}\}$ as all not yet completed activities whose predecessors have been completed and for whom at time t_μ there is enough remaining capacity in order to be processed. We can now provide a formal description of the parallel SGS as follows:

```
 1: Initialization: $\mu := 0, t_\mu := 0, \mathscr{C}_0 := \emptyset, S_0 := 0, \mathscr{A}_0 := \{0\}$.
 2: Iteration: While $| \mathscr{C}_\mu \cup \mathscr{A}_\mu | \le n$ do
 3:     $\mu := \mu + 1$.
 4:     $t_\mu := \min_{j \in \mathscr{A}_{\mu-1}} \{S_j + p_j\}$.
 5:     Update $\mathscr{C}_\mu, \mathscr{A}_\mu, \tilde{R}_k(t_\mu)$ for all $k \in \mathscr{R}, \mathscr{D}_\mu$.
 6:     While $\mathscr{D}_\mu \neq \emptyset$ do
 7:         Select one $j \in \mathscr{D}_\mu$.
 8:         $S_j := t_\mu$.
 9:         Update $\tilde{R}_k(t_\mu)$ for all $k \in \mathscr{R}, \mathscr{A}_\mu, \mathscr{D}_\mu$.
10: $S_{n+1} = \max_{h \in Pred(n+1)} S_h + p_h$.
```

The initialisation in line 1 sets the schedule time t_μ as well as the start of the dummy start activity to zero and assigns the dummy start activity to the active set. Each iteration has a unique counter μ and a unique schedule time t_μ which is the earliest finish time of the activities in process at iteration $\mu - 1$ (see line 4). At the new schedule time t_μ, the sets of completed and active activities, the remaining capacity of the resources and the decision set are calculated in line 5. Afterwards, in lines 6 to 9, the activities from the decision set are started in t_μ as long as there is sufficient capacity. Following the start of each activity, the capacity as well as the active set and the decision set are updated. Once all n real activities have been scheduled, the start time of the dummy activity $n + 1$ is determined in line 10.

We illustrate the parallel SGS by applying it to the example problem given in Fig. 1.2 and again employing the MSLK priority rule (see Table 1.1 for the priority values of the activities). Table 1.3 provides the decision time t_μ, the decision set $\mathscr{D}_\mu$ and the selected activity j for each iteration $\mu = 1, \ldots, 5$. The resulting schedule is the non-delay schedule given in Fig. 1.8.

Table 1.3 Iterations of the parallel SGS for the example instance

μ	1	1	2	3	4	5
t_μ	0	0	1	2	3	5
$\mathscr{D}_\mu$	$\{1, 3\}$	$\{1\}$	$\emptyset$	$\{2, 4\}$	$\{2, 5\}$	$\{2\}$
j	3	1		4	5	2

The parallel SGS can also be used with an activity list $\ell = (j_1, j_2, \ldots, j_n)$ (see Hartmann 2002). In this case, in line 7 we choose the activity from the decision set $\mathscr{D}_\mu$ which is on the foremost position of ℓ.

The parallel SGS always generates a feasible schedule if one exists, as is the case for the serial SGS. A schedule constructed by the parallel SGS belongs to the set of non-delay schedules (see Kolisch 1996). This property stems from the fact that the parallel schedule generation scheme schedules at a schedule time t_μ as many activities from the decision set as possible thereby checking only the available capacity at t_μ. Since the set of non-delay schedules is a subset of the set of active schedules, the cardinality of the set of non-delay schedules is smaller and thus fewer schedules have to be implicitly or explicitly generated when searching for a good or an optimal schedule. Experimentally it has been observed that for a simple priority rule based heuristic which generates one schedule, independent of the priority rule employed, the average objective function value when employing the parallel SGS is smaller than when employing the serial SGS. Kolisch (1996) reports an average percentage deviation from the optimal makespan when employing the priority rules LST, LFT, MTS, MSLK, GRPW and WRUP within the parallel and the serial schedule generation scheme on the PSPLIB instance set with $n = 30$ activities (see Kolisch et al. 1995) of 6.47 % compared to 9.23 %. However, when more time is spent in order to generate and examine more schedules, the parallel SGS suffers from the fact that the best schedule found might not be optimal. Kolisch (1996) reports that on the PSPLIB instances with $n = 30$ activities, which are truly resource-constrained, that is, for which the earliest start schedule is not feasible, the best non-delay schedule is for 41.27 % of the instances not optimal.

Kolisch (1996) has compared the aforementioned priority rules LST, LFT, MTS, MSLK, GRPW and WRUP in a regret-based random sampling context with sample sizes between 10 and 100. The results indicated that, for 30 activity problems, the serial SGS generates better solutions than the parallel SGS for sample sizes larger than 40. For this reason the majority of the metaheuristics which generate and evaluate up to 50,000 (not necessarily different) solutions employ the serial SGS. Some metaheuristics such as the genetic algorithm of Hartmann (2002) or the population-based approach of Kochetov and Stolyar (2003) employ the serial and the parallel SGS. However, according to the study of Kolisch and Hartmann (2006), none of the best performing metaheuristics employs only the parallel SGS.

1.6 Schedule Generation Schemes for Special Cases and Generalizations of the RCPSP

In this section we address address schedule generation schemes for the simple assembly line balancing problem as a special case of the RCPSP and for generalizations of the RCPSP with minimum and maximum time lags or stochastic activity durations.

The simple assembly line balancing problem (SALBP) with the objective function to minimize the number of stations subject to a fixed cycle time per station can be modeled as RCPSP and thus is a special case of the RCPSP (see De Reyck and Herroelen 1997). Almost all solution procedures for SALBP employ one of two construction schemes, station-oriented or task-assignment (see Scholl and Becker 2006). These two schemes are straight forward adaptations of the parallel and serial SGS, respectively.

An extension of the RCPSP with simple finish-to-start precedence constraints considers general precedence constraints with minimum and maximum time lags (see Neumann et al. 2002). Neumann et al. (2002) extend the discussion of shifts and schedules for the RCPSP with minimum and maximum time lags. They also show that the problem of finding a feasible solution becomes $\mathcal{NP}$-complete (see Neumann et al. 2002, p. 38). As a consequence, the schedule generation schemes for the RCPSP cannot be applied straight forwardly. Instead, they have to be adapted accordingly by considering the unscheduling of activities in case of an infeasible partial solution.

The stochastic RCPSP (SRCPSP) extends the determistic RCPSP in that the durations of the activities are not deterministic any more but stochastic. For the duration of each activity a known probability distribution is assumed. Due to the fact that the activity durations are stochastic, a schedule with fixed start times can no longer be employed. Instead, one has to resort to scheduling policies. For each decision time t, that is whenever an activity is finished, a scheduling policy states which activity has to be started next. This decision can only be based on the information which has become available at the decision time t, that is the information on completed activities and activities in progress (non-anticipativity constraint, see Storck 2001). A scheduling policy is depicted by an activity list giving priority to activities according to the sequence of the list. A scheduling policy is always embedded within a policy class that defines how the list is transformed into a schedule and is the counterpart to the schedule generation scheme for the deterministic case. Two well known policy classes for the SRCPSP are the resource-based and the activity-based policy class (see Ashtiani et al. 2011 for a discussion of these two classes as well as for introducing a new class).

The resource-based policy class employs the parallel SGS together with an activity list ℓ. At each decision time t, not yet scheduled activities are scanned according to the list ℓ if they can be started with respect to precedence and resource constraints at t. If no further activity from the list can be started, the decision time advances to the minimum finish time of the activities in progress. Due to the property of the parallel SGS activity j_i on the list position i can be started earlier than activity j_h on list position $h < i$. This holds true for the scenario given by the example problem (see Fig. 1.2) and activity list $\ell = (3, 4, 5, 1, 2)$ where activity 1 on list position 4 is started earlier than activities 4 and 5 on list positions 2 and 3, respectively. In terms of activity start times, this leads to different activity sequences depending on the scenario. It also leads to so-called Graham anomalies (Graham 1966) where for two scenarios the scenario with shorter activity durations results in a longer project makespan. When viewed as a function, the class of resource-based policies is thus neither monotone nor continuous (see Storck 2001).

In order to obtain a policy class which is monotone and continuous, the parallel SGS is modified by adding for each activity a side constraint $S_{j_h} \leq S_{j_i}$ for $h < i$. That is, an activity must not star earlier than any of its predecessors in the activity list. This modified policy class is termed activity-based. Despite its nice theoretic properties, the activity-based policy experimentally yields a longer expected makespan when compared to the resource-based policy (see Ashtiani et al. 2011 and Fang et al. 2012).

1.7 Conclusions

In this chapter we addressed schedule generation schemes which are the backbone of heuristics to solve project scheduling problems. We revisited the two schedule generation schemes for the classical resource-constrained project scheduling problem, the serial and the parallel schedule generation scheme. We characterized the two schemes according to their runtime and the types of schedule they generate. We then showed how the two schemes are embedded in metaheuristics. Finally, we discussed variants of the two schemes for the simple assembly line balancing problem as a special case of the RCPSP as well as for generalizations of the RCPSP with general precedence constraints or with stochastic activity durations.

References

Ashtiani B, Leus R, Aryanezhad M-B (2011) New competitive results for the stochastic resource-constrained project scheduling problem: exploring the benefits of preprocessing. J Scheduling 14:157–171

Błażewicz J, Lenstra JK, Rinnooy Kan AHG (1983) Scheduling subject to resource constraints: classification and complexity. Discrete Appl Math 5:11–24

Błażewicz J, Ecker K, Pesch E, Schmidt G, Węglarz J (2007) Handbook on scheduling. Springer, Berlin

De Reyck B, Herroelen W (1997) Assembly line balancing by resource-constrained project scheduling: a critical appraisal. Found Comput Control Eng 22:143–167

Debels D, De Reyck B, Leus R, Vanhoucke M (2006) A hybrid scatter search/electromagnetism meta-heuristic for project scheduling. Eur J Oper Res 169:638–653

Demeulemeester E, Herroelen W. (1992) A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. Manage Sci 38:1803–1818

Demeulemeester E, Herroelen W (1997) New benchmark results for the resource-constrained project scheduling problem. Manage Sci 43:1485–1492

Demeulemeester E, Herroelen W (2002) Project scheduling: a research handbook. Kluwer, Boston

Fang E, Kolisch R, Wang L, Mu C (2012) An estimation of distribution algorithm and new computational results for the stochastic resource-constrained project scheduling problem. Technical Report, TUM School of Management, Technische Universität München, München, Germany

Graham R (1966) Bounds on multiprocessing timing anomalies. Bell Syst Tech J 45:1563–1581

Hartmann S (1998) A competitive genetic algorithm for resource-constrained project scheduling. Nav Res Log 45:733–750

Hartmann S (2002) A self-adapting genetic algorithm for project scheduling under resource constraints. Nav Res Log 49:433–448

Kochetov Y, Stolyar A (2003) Evolutionary local search with variable neighborhood for the resource constrained project scheduling problem. In: Proceedings of the 3rd international workshop of computer science and information technologies, Russia

Kolisch R (1996) Serial and parallel resource-constrained project scheduling methods revisited: theory and computation. Eur J Oper Res 90:320–333

Kolisch R, Hartmann S (1999) Heuristic algorithms for the resource-constrained project scheduling problem: classification and computational analysis. In: Węglarz J (ed) Project scheduling: recent models, algorithms and applications. Kluwer, Boston, pp 147–178

Kolisch R, Hartmann S (2006) Experimental evaluation of heuristics for the resource-constrained project scheduling problem: an update. Eur J Oper Res 174:23–37

Kolisch R, Sprecher A, Drexl A (1995) Characterization and generation of a general class of resource-constrained project scheduling problems. Manage Sci 41:1693–1703

Neumann K, Schwindt C, Zimmermann J (2002) Project scheduling with time windows and scarce resources. Lecture notes in economics and mathematical systems, vol 508. Springer, Berlin

Scholl A, Becker C (2006) State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. Eur J Oper Res 168:666–693

Sprecher A (2000) Solving the RCPSP efficiently at modest memory requirements. Manage Sci 46(5):710–723

Sprecher A, Kolisch R, Drexl A (1995) Semi-active, active, and non-delay schedules for the resource-constrained project scheduling problem. Eur J Oper Res 80:94–102

Stork F (2001) Stochastic resource-constrained project scheduling. Ph.D. dissertation, Fachbereich Mathematik, TU Berlin, Berlin, Germany

Chapter 2
Mixed-Integer Linear Programming Formulations

Christian Artigues, Oumar Koné, Pierre Lopez, and Marcel Mongeau

Abstract In this chapter, (mixed-)integer linear programming formulations of the resource-constrained project scheduling problem are presented. Standard formulations from the literature and newly proposed formulations are classified according to their size in function of the input data. According to this classification, compact models (of polynomial size), pseudo-polynomial sized models, and formulations of exponential size are presented. A theoretical and experimental comparison of these formulations is then given. The complementarity of the formulations for different usages is finally discussed and directions for future work, such as hybridization with other methods, are given.

Keywords Makespan minimization • Mixed-integer linear programming formulations • Project scheduling • Resource constraints

2.1 Introduction

Given an $\mathcal{NP}$-hard optimization problem, such as the resource-constrained project scheduling problem ($PS \mid prec \mid C_{max}$ or RCPSP), a natural approach for operations research practitioners is to formulate it as a mixed-integer linear programming program (MILP), i.e., an optimization problem involving an objective function and constraints that are all linear and continuous and integer variables. In the case where the obtained formulation involves only integer variables, one shall refer to it as an

C. Artigues (✉) • P. Lopez
CNRS, LAAS, Univ de Toulouse, Toulouse, France
e-mail: artigues@laas.fr; lopez@laas.fr

O. Koné
Laboratoire de Mathématiques et Informatique, Université Nangui Abrogoua, Abidjan, Côte d'Ivoire
e-mail: mr.okone@gmail.com

M. Mongeau
lab MAIAA, ENAC, Toulouse, France
e-mail: marcel.mongeau@enac.fr

C. Schwindt, J. Zimmermann (eds.), *Handbook on Project Management and Scheduling Vol. 1*, International Handbooks on Information Systems,
DOI 10.1007/978-3-319-05443-8_2

integer linear program (ILP). Maturity of (integer) linear programming theory and tremendous progress of commercial or free MILP solvers explains the appeal of such a formulation.

Nevertheless, there can be multiple ways of formulating a combinatorial optimization program as an MILP. In a problem-independent point of view, such formulations can be distinguished according to their size (number of variables, especially the integer ones, and constraints) and by the strength of their linear programming (LP) relaxation (obtained by relaxing the integrality constraints on the variables). There is a correlation between these two characteristics. Given a combinatorial optimization problem involving n variables (such as the start times in a scheduling problem), an *extended* formulation aims at introducing extra variables so as to improve the LP relaxation. In this chapter, we will consider the *compact* models (of polynomial size), formulations of pseudo-polynomial size, and extended formulations of exponential size that have been proposed for the RCPSP.

For the RCPSP, MILP formulations are generally classified in a problem-oriented way, according to the way time and resource-sharing characteristics are modeled. This classification, in three categories, comes from fundamental polyhedral results in machine scheduling (Queyranne and Schulz 1994). *Time-indexed* (or discrete-time) formulations are, in general, purely integer and have the best LP relaxation at the expense of a large size. They are presented in Sect. 2.2. *Sequencing/natural-date* formulations on the one hand (Sect. 2.3) and *Positional-date/assignment* formulations on the other hand (Sect. 2.4) are compact MILP that generally yield weaker LP relaxations. Theoretical and experimental comparisons of the various formulations are discussed in Sect. 2.5.

For the reader's information, some of the material presented in this chapter can also be found in previous surveys on MILP formulations for the RCPSP, especially in the books by Klein (2000), Demeulemeester and Herroelen (2002), Artigues et al. (2008), and Brucker and Knust (2012).

2.2 Time-Indexed Formulations

Time-indexed, also called discrete-time, integer linear programming (ILP) formulations have been widely studied for single machine, parallel machine, and resource-constrained scheduling problems (Christofides et al. 1987; de Sousa and Wolsey 1997; Pritsker and Watters 1968; Pritsker et al. 1969; Queyranne and Schulz 1994; Sankaran et al. 1999). This is due to their relatively strong LP relaxations and to their ability for being extended to various constraints and objectives.

Time-indexed formulations are ILP formulations that involve time-indexed (or discrete-time) variables of type v_{it} indicating a particular status of activity i at time t (generally: *started*, *completed* or *in progress*).

As the number of variables is linear in function of the scheduling horizon, which can be, for some problem instances, as large as the sum of activity durations, the standard variant of this type of pure ILP formulations is of pseudo-polynomial size.

In order to be implemented with a finite number of variables, these formulations require an upper bound T on the makespan. In practice, for each activity, one defines a finite set of time periods during which the activity can start. Let $V = \{0, 1, \ldots, n, n+1\}$ denote the set of activities, where 0 and $n+1$ are fictitious start and end activities, respectively. Typically, preprocessing techniques use an upper bound on the makespan and compute for each activity $i \in V$ of processing time p_i, an earliest start time ES_i, an earliest completion time $EC_i = ES_i + p_i$, a latest start time LS_i, and a latest completion time $LC_i = LS_i + p_i$. Assuming that $(0, i) \in E$, $\forall i \in V \setminus \{0\}$, and $p_0 = 0$, ES_i can be set to the length of the longest path between 0 and i in the precedence graph $G(V, E)$ where each "precedence edge" $(i, j) \in E$ is valuated by p_i. Symmetrically, LS_i can be set to T minus the length of the longest path between i and $(n+1)$ in G.

We consider the scheduling horizon as a set $H = \{0, \ldots, T\}$ of consecutive integer values starting with $t = 0$. In the case of integer data, the set of start times can be restricted to H.

We identify time t with the time period defined by the interval $[t, t+1)$. Hence, by convention, we mean that an activity is in progress at time t if and only if its start time satisfies $S_i \leq t$ and $S_i + p_i \geq t + 1$. An activity starts at time t if and only if $S_i = t$, and an activity completes at time t if and only if $S_i + p_i = t$.

In Sect. 2.2.1 we present the time-indexed formulations based on *pulse* variables. Section 2.2.2 describes the time-indexed formulations based on *step* variables. In Sect. 2.2.3, we present a time-indexed *on-off* formulation obtained from the former formulations by non-singular transformations. As shown by Artigues (2013), this formulation is stronger than the ones previously proposed in the literature and based on the same type of variables. Section 2.2.4 features additional remarks on the strengths of other related time-indexed formulations encountered in the project scheduling literature.

To obtain stronger formulations, an integer Dantzig–Wolfe decomposition of constraints can be achieved. This is the principle of the *feasible-subset formulation* described in Sect. 2.2.5, and of the *chain-decomposition-based formulation* presented in Sect. 2.2.6.

2.2.1 The Discrete-Time Formulation Based on "Pulse" Start Variables

The more standard time-indexed formulation for the RCPSP (Christofides et al. 1987; Pritsker et al. 1969) is based on binary variable x_{it}, $\forall i \in V$, $\forall t \in H$ such that $x_{it} = 1$ if and only if activity i starts at time t. For a given activity i, all variables x_{it} are equal to 0, except for time $t = S_i$; we refer to this type of variables as *pulse variables*. The basic discrete-time formulation based on pulse start variables, noted (DT) in the sequel, then comes as follows:

$$\text{Min.} \sum_{t \in H} t x_{n+1,t} \tag{2.1}$$

$$\text{s. t.} \sum_{t \in H} t x_{jt} - \sum_{t \in H} t x_{it} \geq p_i \quad ((i,j) \in E) \tag{2.2}$$

$$\sum_{i \in V} \sum_{\tau = t - p_i + 1}^{t} r_{ik} \, x_{i\tau} \leq R_k \quad (t \in H;\ k \in \mathscr{R}) \tag{2.3}$$

$$\sum_{t \in H} x_{it} = 1 \quad (i \in V) \tag{2.4}$$

$$x_{it} = 0 \quad (i \in V;\ t \in H^{+} \setminus \{ES_i, \ldots, LS_i\}) \tag{2.5}$$

$$x_{it} \in \{0, 1\} \quad (i \in V;\ t \in \{ES_i, \ldots, LS_i\}) \tag{2.6}$$

The objective (2.1) is to minimize the makespan. Constraints (2.2) are the precedence constraints. They indeed directly translate expression $S_j \geq S_i + p_i, \forall (i,j) \in E$, observing that, according to the above-definition of the pulse variables, one has $S_i = \sum_{t \in H} t x_{it}$. Constraints (2.3) are the resource constraints, expressing that the sum of resource requirement of activities in progress at each time $t \in H$ cannot exceed the capacity of any resource $k \in \mathscr{R}$. As start times are integer, i is in progress at time t if and only if i starts at a time $\tau \in H \cap [t - p_i + 1, t]$. Constraints (2.4) state that each activity has to be started exactly once in the scheduling horizon. Constraints (2.5) set to 0 all variables out of set $H^{+} \cap [ES_i, LS_i]$, where H^{+} is defined, for ease of notation, as the extension of H by a sufficiently large set of negative integers. Finally, constraints (2.6) define the binary pulse decision variables.

A formulation yielding a stronger LP relaxation was proposed by Christofides et al. (1987). It aims at replacing constraints (2.2) by the so-called *disaggregated* precedence constraints (2.7):

$$\sum_{\tau=0}^{t-p_i} x_{i\tau} - \sum_{\tau=0}^{t} x_{j\tau} \geq 0 \quad ((i,j) \in E;\ t \in H) \tag{2.7}$$

These constraints simply model the logical relation: $S_j \leq t \Rightarrow S_i \leq t - p_i$.

The disaggregated discrete-time formulation based on pulse start variables, noted (DDT), is then obtained by replacing constraints (2.2) with constraints (2.7) in formulation (2.1–2.6). The formulation is not weaker than (DT) since constraints (2.2) are implied by constraints (2.4) together with (2.7) for $0 \leq x_{it} \leq 1$, $\forall i \in V, \forall t \in H$. Moreover, (DDT) features advantageous properties that will be explained below.

2.2.2 The Discrete-Time Formulation Based on Start "Step" Variables

We consider another formulation, based now on binary variables ξ_{it} such that $\xi_{it} = 1$ if and only if activity i starts at time t or before. For a given activity i, the variables ξ_{it} such that $t < S_i$ are all equal to 0, while the variables with indices $t \geq S_i$ are all equal to one. Hence, we refer to these type of variables as *step variables*. With these definitions, the start time can be expressed as:

$$S_i = \sum_{t \in H} t(\xi_{it} - \xi_{i,t-1}) \tag{2.8}$$

We present only the disaggregated variant of the discrete-time formulation based on step variables, noted (SDDT):

$$\text{Min.} \sum_{t \in H} t(\xi_{n+1,t} - \xi_{n+1,t-1}) \tag{2.9}$$

$$\text{s. t.} \quad \xi_{i,t-p_i} - \xi_{jt} \geq 0 \quad ((i, j) \in E;\ t \in H) \tag{2.10}$$

$$\sum_{i \in V} r_{ik}(\xi_{it} - \xi_{i,t-p_i}) \leq R_k \quad (t \in H;\ k \in \mathscr{R}) \tag{2.11}$$

$$\xi_{i,LS_i} = 1 \quad (i \in V) \tag{2.12}$$

$$\xi_{it} - \xi_{i,t-1} \geq 0 \quad (i \in V;\ t \in H) \tag{2.13}$$

$$\xi_{it} = 0 \quad (i \in V;\ t \in H^+,\ t \leq ES_i - 1) \tag{2.14}$$

$$\xi_{it} \in \{0, 1\} \quad (i \in V;\ t \in \{ES_i, \ldots, LS_i - 1\}) \tag{2.15}$$

The objective (2.9) is directly obtained by replacing the start time variable by its expression (2.8) in function of ξ. The disaggregated precedence constraints(2.10) state that if an activity j is started at time t of before, i.e., if $\xi_{jt} = 1$, then activity i has also to be started at time $t - p_i$ or before. Resource constraints (2.11) follow from the fact that an activity i is in progress at time t if and only if $\xi_{it} - \xi_{i,t-p_i} = 1$. Indeed, if $t \in [ES_i, ES_i + p_i - 1]$, we have $\xi_{i,t-p_i} = 0$ by definition, and i is in progress at time t if and only if $\xi_{it} = 1$. Otherwise, i is in progress at time t if and only if it has been started at time t but not at time $t - p_i$. Constraints (2.12) state that each activity has to be started at or before its latest start time LS_i. Constraints (2.13) define the step function, together with constraints (2.12). Note that these constraints also set to 1 all variables ξ_{it} with $t \geq LS_i$. Constraints (2.14) are just here for ease of notation as noted above, to set to 0 all variables ξ_{it} with $t < ES_i$. Finally, constraints (2.15) define the binary step variables.

Although it is presented as new by Klein (2000), the step formulation (in the aggregated form) was in fact already presented by Pritsker and Watters (1968) and theoretically studied and compared to the pulse formulation (in the disaggregated form) by de Sousa and Wolsey (1997) and Sankaran et al. (1999). This has been also underlined by Möhring et al. (2001). If we omit resource constraints in the (SDDT) formulation, and if we relax integrality constraints, i.e., considering only constraints (2.10, 2.12–2.14), and $0 \leq \xi_{it} \leq 1$, $\forall i \in V, t \in H$, de Sousa and Wolsey (1997) and Sankaran et al. (1999) observed that the constraint matrix satisfies a sufficient total-unimodularity condition. It follows from this observation that, without resource-constraints the solution of the LP relaxation of (SDDT) is $0 - 1$. So is the solution of the LP relaxation of (DDT), according to the following remark, made by de Sousa and Wolsey (1997) and Sankaran et al. (1999). The (SDDT) formulation can be obtained directly by applying the following non-singular transformation to the (DDT) formulation. For all $t \in H$, we have $x_{it} = \xi_{it} - \xi_{i\,t-1}$. Conversely, the inverse transformation defines $\xi_{it} = \sum_{\tau=0}^{t} x_{it}$ and gives the (DDT) formulation from the (SDDT) formulation. Note that, in both cases, this transformation does not change the value of the LP relaxation. An aggregated discrete-time formulation based on step variables (SDT) could also be defined this way. Formulations (DT) and (SDT) are also equivalent, for the same reason, but yield weaker relaxations as fractional solutions can be obtained by solving the LP relaxations without resource constraints (Möhring et al. 2001; Sankaran et al. 1999).

2.2.3 *The Discrete-Time Formulation Based on On/Off Variables*

Any non-singular (linear) transformation can be applied on the above-defined formulations to obtain yet another, equivalent, formulation. In this section, we consider *on/off* binary variables μ_{it} where $\mu_{it} = 1$ if activity i is in progress at time t, and $\mu_{it} = 0$ otherwise. According to our definition of the "in progress" status, an activity with zero duration cannot be in progress, so we will treat activities with zero duration separately. This on/off model is based on the following observations on the relations between the above-defined binary variables x_{it}, ξ_{it}, and μ_{it}.

As already observed while describing the resource constraints for the (DT), (DDT), (SDT), and (SDDT) models, if $p_i \geq 1$, then an activity $i \in V$ is in progress at time $t \in H$ if and only if $\xi_{it} - \xi_{i,t-p_i} = 1$ and, equivalently, if $\sum_{\tau=t-p_i+1}^{t} x_{i\tau} = 1$. So, for any activity $i \in V$ such that $p_i \geq 1$, and for any time $t \in H$, we define the non-singular transformations $\mu_{it} = \xi_{it} - \xi_{i,t-p_i}$ and $\mu_{it} = \sum_{\tau=t-p_i+1}^{t} x_{i\tau}$. To obtain the inverse transformation for ξ_{it}, we sum all $\mu_{i\tau}$ for $\tau = t - \alpha p_i$ and $\alpha = 0, \ldots, \lfloor t/p_i \rfloor$, which gives

$$\xi_{it} = \sum_{\alpha=0}^{\lfloor t/p_i \rfloor} \mu_{i,t-\alpha p_i} \tag{2.16}$$

which means that i is started at t or before if and only if it is in progress at time $t - \alpha p_i$ for some $\alpha \in \mathbb{N}$. Furthermore, as $x_{it} = \xi_{it} - \xi_{it-1}$, we obtain the inverse transformation for x_{it},

$$x_{it} = \sum_{\alpha=0}^{\lfloor t/p_i \rfloor} \mu_{i,t-\alpha p_i} - \sum_{\alpha=0}^{\lfloor (t-1)/p_i \rfloor} \mu_{i,t-\alpha p_i-1} \tag{2.17}$$

The start time S_i is then equal to

$$S_i = \sum_{t \in H} t \left(\sum_{\alpha=0}^{\lfloor t/p_i \rfloor} \mu_{i,t-\alpha p_i} - \sum_{k=0}^{\lfloor (t-1)/p_i \rfloor} \mu_{i,t-\alpha p_i-1} \right)$$

Considering now the particular cases where $p_i = 0$, we change the definitions above with $\xi_{it} = \mu_{it}$. So, by defining $K_{it} = 0$ if $p_i = 0$ and $K_{it} = \lfloor t/p_i \rfloor$ otherwise, we obtain

$$\xi_{it} = \sum_{\alpha=0}^{K_{it}} \mu_{i,t-kp_i} \text{ and } x_{it} = \sum_{\alpha=0}^{K_{it}} \mu_{i,t-kp_i} - \sum_{\alpha=0}^{K_{i,t-1}} \mu_{i,t-\alpha p_i-1}$$

We have thereby defined a non-singular transformation. Substituting variables x_{it} by variables μ_{it} in formulation (DDT), we obtain the following formulation, noted (OODDT):

$$\text{Min.} \sum_{t \in H} t(\mu_{n+1,t} - \mu_{n+1,t-1}) \tag{2.18}$$

$$\text{s.t.} \quad \sum_{\alpha=0}^{K_{i,t-p_i}} \mu_{i,t-(\alpha+1)p_i} - \sum_{\alpha=0}^{K_{jt}} \mu_{j,t-\alpha p_j} \geq 0 \quad ((i,j) \in E;\ t \in H) \tag{2.19}$$

$$\sum_{i \in V, p_i > 0} r_{ik}\, \mu_{it} \leq R_k \quad (t \in H;\ k \in \mathscr{R}) \tag{2.20}$$

$$\sum_{\alpha=0}^{K_{i,LC_i-\phi(i)}} \mu_{i,LC_i-\phi(i)-\alpha p_i} = 1 \quad (i \in V) \tag{2.21}$$

$$\sum_{\alpha=0}^{K_{it}} \mu_{i,t-\alpha p_i} - \sum_{\alpha=0}^{K_{i,t-1}} \mu_{i,t-\alpha p_i-1} \geq 0 \quad (i \in V;\ t \in H \setminus \{0\}) \tag{2.22}$$

$$\mu_{it} = 0 \quad (i \in V;\ t \in Z(i)) \tag{2.23}$$

$$\mu_{it} \in \{0,1\} \quad (i \in V;\ t \in U(i)) \tag{2.24}$$

Constraints (2.19) are the disaggregated precedence constraints, given the expression of the start time variables S_i in function of the on/off variables μ_{it}. Constraints (2.20) are the resource constraints, which have here a particularly simple expression due to the on/off variables. Let us introduce for each activity i the convenient notation: $\phi(i) = 1$ if $p_i \geq 1$ and $\phi(i) = 0$ if $p_i = 0$. With this convention, constraints (2.21) state that each activity i such that $p_i \geq 1$ has to be in progress in exactly one time period among time periods $t = LC_i - 1$, $t = LC_i - 1 - p_i$, $t = LC_i - 1 - 2p_i, \ldots$. For activities such that $p_i = 0$, the constraints simply resort to constraints (2.12), as $\mu_{it} = \xi_{it}$. Constraints (2.22) are obtained by substitution of (2.16) within constraints (2.13) of (SDDT), or, equivalently, of (2.17) within constraints $x_{it} \geq 0$ from (DDT). They ensure, together with constraints (2.21) that exactly p_i consecutive variables will be switched on, i.e., in a non-preemptive fashion (see Theorem 2.1 below). Constraints (2.23) simply set dummy variables to 0, where we define the set of zero variables: $Z(i) = H^+ \setminus \{ES_i, \ldots, LC_i - 1\}$ if $p_i \geq 1$, and $Z(i) = \{t \in H^+ \,|\, t \leq ES_i - 1\}$ when $p_i = 0$. Constraints (2.24) define the on/off binary variables, where we define the set of undetermined variables: $U(i) = \{ES_i, \ldots, LC_i - 1\}$ if $p_i \geq 1$, and $U(i) = \{ES_i, \ldots, LS_i - 1\}$ when $p_i = 0$.

As the three formulations (OODDT), (SDDT) and (DDT) can be obtained from each other via non-singular transformations, they are strictly equivalent and yield the same LP relaxation. Similarly, an aggregated formulation (OODT) could be obtained from the (SDT) and (DT) formulations.

Klein (2000) presented a variant of the on/off formulation based on the formulation of Kaplan (1998) for the preemptive RCPSP. This formulation is similar to (OODDT) with the following differences. No tasks with duration 0 are allowed. Precedence constraints are replaced by the constraints (2.25) below. Non-preemption/duration constraints (2.21–2.22) are replaced by the duration constraints (2.26) and the non-preemption constraints (2.27).

$$p_i \mu_{jt} - \sum_{q=ES_i}^{t-1} \mu_{it} \leq 0 \qquad ((i,j) \in E;\ t \in \{ES_j, \ldots, LC_i - 1\}) \tag{2.25}$$

$$\sum_{t=ES_i}^{LC_i-1} \mu_{it} = p_i \quad (i \in V) \tag{2.26}$$

$$p_i(\mu_{it} - \mu_{i,t+1}) - \sum_{q=t-p_i+1}^{t-1} \mu_{iq} \leq 1 \quad (i \in V;\ t \in \{ES_i, \ldots, LC_i - 2\}) \tag{2.27}$$

The precedence constraints (2.25) state that for an activity j to be in progress at time t, its predecessor i must have been entirely processed during interval $[ES_i, t]$. Duration constraints (2.26) are straightforward. Non-preemption constraints (2.27) model the fact that if an activity i completes at time $t + 1$, in which case the coefficient of p_i is equal to one, then the $p_i - 1$ precedent μ_{iq} variables must be switched on.

Demeulemeester and Herroelen (2002) present yet another variant. As explained below, the formulation has a slight mistake in the constraints range and we provide here a corrected version, replacing the precedence constraints (2.25) by the exclusive constraints (2.28) and (2.29) below, that are in fact stronger than Klein's precedence constraints. Let (KF) denote this formulation.

$$\mu_{jt} \leq \mu_{i,t-p_i} \quad (i \in V;\ j \in V \setminus \{i\};\ t \in H,\ t \leq EC_i - 1 + p_i) \tag{2.28}$$

$$\mu_{jt} \leq \sum_{q=ES_i+p_i-1}^{t-p_i} \mu_{iq} \quad (i \in V;\ j \in V \setminus \{i\};\ t \in H,\ t \geq EC_i + p_i) \tag{2.29}$$

Constraints (2.28) state that, to be in progress at time t, an activity j must have its predecessor, i, in progress at time $t - p_i$ for any t such that $t - p_i$ falls strictly before the earliest end time of i, $EC_i = ES_i + p_i$. Indeed, if i starts at $t - p_i$ or before (which allows j to be in progress at t), then i is necessarily in progress at time $t - p_i$, i.e., $S_i \leq t - p_i$ if and only if i is in progress at time $t - p_i$. If on the other hand, $t - p_i$ exceeds the earliest completion time of i, then constraints (2.29) state that activity j can only be in progress at time t if its predecessor, i, starts at $t - p_i$ or before, which means that i has to be in progress during at least one time period between $ES_i + p_i - 1$ and $t - p_i$.

As already mentioned, there is a slight mistake in the constraint given by Demeulemeester and Herroelen (2002) as $ES_i + p_i$ was replaced by ES_j in the range of constraints (2.28) and (2.29). In the case where $ES_i + p_i < ES_j$, this can lead to overconstraining the start time of j. Indeed, suppose two activities i and j with $(i, j) \in E$, $ES_i = 0$, $ES_j = 5$, and $p_i = 4$. At time $t = 8$, since $t - p_i = 4$ is not strictly before $EC_i = ES_i + p_i = 4$, we are in the range of constraints (2.29), so j can be in progress at time t either if i is in progress at times $t' = 4$ or $t' = 3$. If we use $ES_j = 5$ instead of $ES_i + p_i = 4$ in the constraints range, then $t - p_i$ is strictly before ES_j, which falls in the range of constraints (2.28) stating that j can be in progress at time t only if i is in progress at time $t' = 4$. This clearly overconstrains the start time of j.

We now focus on the relative strengths of the proposed (OODDT) formulation and the (KF) formulation. We restrict to instances where no activity has a zero duration, otherwise (KF) cannot be used.

Theorem 2.1. *Formulation (OODDT) is stronger than formulation (KF).*

The proof of this theorem is given in Artigues (2013).

2.2.4 Other Equivalent or Weaker Discrete-Time Formulations

Klein (2000) introduces a variant (SDDT2) of the (SDDT) formulation by introducing another step binary variable$\gamma_{it} = 1$ if i completes at time t or after. Observe that we have $\xi_{it} + \gamma_{it} - 1 = \mu_{it}$ for activities with non zero durations and $\xi_{it} + \gamma_{it} - 1 = x_{it}$ for activities with zero duration. Using these non-singular transformations we could obtain aggregated or disaggregated formulations based on γ_{it} variables and equivalent to the ones already presented. Klein (2000) introduces a formulation that is not stronger but that has an advantage when durations are decision variables. Indeed, in all formulations we presented so far, durations p_i have to be fixed parameters because they appear in the range of values for the index of variables. Mixing ξ_{it} and γ_{it} allows to get rid of this drawback. We can modify the (SDT) model by adding the following constraints, defining the γ variables and establishing the link with the ξ variables,

$$\sum_{t=ES_i}^{LC_i-1} \xi_{it} + \gamma_{it} - 1 = p_i \quad (i \in V) \tag{2.30}$$

$$\gamma_{i,EC_i-1} = 1 \quad (i \in V) \tag{2.31}$$

$$\gamma_{i,t-1} - \gamma_{i,t} \geq 0 \quad (i \in V;\ t \in H) \tag{2.32}$$

$$\gamma_{it} = 0 \quad (i \in V;\ t \in H^+,\ t \geq LC_i) \tag{2.33}$$

$$\gamma_{it} \in \{0, 1\} \quad (i \in V;\ t \in \{EC_i, \ldots, LC_i - 1\}) \tag{2.34}$$

and replacing resource constraints (2.11) by

$$\sum_{i \in V} r_{ik}(\xi_{it} + \gamma_{it} - 1) \leq R_k \quad (t \in H;\ k \in \mathscr{R}) \tag{2.35}$$

Bianco and Caramia (2013) propose a variant of the step formulation that involves the $0-1$ start variable ξ_{it} and another $0-1$ variable ξ'_{it} which equals 1 if and only if activity i is completed at t or before. Even if it is not mentioned by Bianco and Caramia (2013), we have

$$\xi'_{it} = \xi_{i,t-p_i} \quad (i \in V;\ t \in H) \tag{2.36}$$

Another variable ζ_{it} is introduced, giving the fraction of activity i that has been performed up to time t. The following constraints are defined:

$$\zeta_{i,t+1} - \zeta_{it} = \frac{1}{p_i}(\xi_{it} - \xi'_{it}) \quad (i \in V; t \in H) \tag{2.37}$$

$$\xi'_{it} \leq \zeta_{it} \leq \xi_{it} \quad (i \in V;\ t \in H) \tag{2.38}$$

$$\zeta_{it} \geq 0 \quad (i \in V;\ t \in H) \tag{2.39}$$

Note that if we introduce constraints (2.36–2.39) in the (SDDT) model, we obtain a strictly equivalent formulation, as it can be shown that for any feasible value of variable ξ_{it} in the LP relaxation of (SDDT), we can obtain values for variables ξ'_{it} immediately through constraints (2.36) and for variables ζ_{it} that satisfy (2.37–2.39).

The formulation (SDDT3) proposed by Bianco and Caramia (2013) finally replaces resource constraints (2.11) by

$$\sum_{i \in V} r_{ik}\, p_i(\zeta_{i,t+1} - \zeta_{it}) \leq R_k \quad (i \in V;\ t \in H;\ k \in \mathscr{R}) \tag{2.40}$$

Remarking that $\zeta_{i,t+1} - \zeta_{it} = \frac{1}{p_i}(\xi_{it} - \xi'_{it}) = \frac{1}{p_i}(\xi_{it} - \xi_{i,t-p_i})$, we precisely obtain resource constraints (2.11).

These formulations were proposed in the literature without any mention of the relative strengths of their LP relaxations. We remark that all the mentioned formulations are either weaker of equivalent to (DDT).

We have to acknowledge that the practical performance of a formulation, in terms of integer solving, is not necessarily related to the LP relaxation strength. It is well known that the weak formulation (DT) may outperform the strong formulation (DDT) on some instances. Bianco and Caramia (2013) showed through extensive experiments that their formulation generally outperformed (SDDT) in terms of solution time and quality. The way constraints and/or additional redundant variables are introduced and formulated influences the solver performance in terms of memory usage, preprocessing, cutting plane generation, and branching. This should not however hide the fact that, in any "new" formulation, constraints that are equivalent, via non-singular transformations, to previously proposed ones should be identified and distinguished from actual cutting-plane inequalities or stronger constraints.

For cutting plane inequalities that can be added to strengthen the (DDT) formulation, we refer to Christofides et al. (1987), de Sousa and Wolsey (1997), Cavalcante et al. (2001), Demassey et al. (2005), and Hardin et al. (2008).

2.2.5 *The Feasible-Subset Formulation*

Mingozzi et al. (1998) have introduced a time-indexed formulation for the RCPSP based on the concept of *feasible subsets*. A feasible subset is a set of activities that can be in progress simultaneously without exceeding any resource availability and that are not pairwise linked by a precedence constraint. Hence, feasible subsets can be assimilated to the antichains of the precedence network (V, E) that, in addition, satisfy resource constraints. Let $\overline{\mathscr{A}}$ denote the set of feasible subsets. The formulation proposed by Mingozzi et al. (1998) makes use of binary on/off variable y_{At}, for each feasible subset $A \in \overline{\mathscr{A}}$ and for each time period t, where $y_{At} = 1$ if all activities from subset A are in progress at time t. The formulation, noted (FSS), also involves the pulse binary variables x_{it}'s:

$$\text{Min.} \sum_{t \in H} t x_{n+1,t} \tag{2.1}$$

$$\text{s.t.} \quad \sum_{A \in \overline{\mathscr{A}}_i} \sum_{t \in H} y_{At} = p_i \quad (i \in V,\ p_i \geq 1) \tag{2.41}$$

$$\sum_{A \in \overline{\mathscr{A}}} y_{At} \leq 1 \quad (t \in H) \tag{2.42}$$

$$x_{it} - \sum_{A \in \overline{\mathscr{A}}_i} (y_{At} - y_{A,t-1}) \geq 0 \quad (i \in V;\ t \in H) \tag{2.43}$$

$$\sum_{t \in H} x_{it} = 1 \quad (i \in V) \tag{2.4}$$

$$\sum_{t \in H} t x_{jt} - \sum_{t \in H} t x_{it} \geq p_i \quad ((i,j) \in E) \tag{2.2}$$

$$y_{At} = 0 \quad (A \in \overline{\mathscr{A}};\ t \in H^+ \setminus \cap_{i \in A}\{ES_i, \ldots, LS_i\}) \tag{2.44}$$

$$x_{it} = 0 \quad (i \in V;\ t \in H^+ \setminus \{ES_i, \ldots, LS_i\}) \tag{2.5}$$

$$x_{it} \in \{0,1\} \quad (i \in V;\ t \in \{ES_i, \ldots, LS_i\}) \tag{2.6}$$

$$y_{At} \in \{0,1\} \quad (A \in \overline{\mathscr{A}};\ t \in \cap_{i \in A}\{ES_i, \ldots, LS_i\}) \tag{2.45}$$

where $\overline{\mathscr{A}}_i \subseteq \overline{\mathscr{A}}$ is the set of all feasible subsets that contain activity i. We only describe the constraints involving the y variables. Constraints (2.41) state that the feasible subsets containing activity i must be in progress during exactly p_i time periods. Constraints (2.42) express that at most one feasible subset can be in progress at each time period. Constraints (2.43) link variables x and y. Constraints (2.44) prevent a feasible subset from being in progress outside of the time window of one of its activities by setting the corresponding variables to 0. Constraints (2.45) define the binary on/off variables for feasible subsets.

It must be noted that in practice the set of feasible subsets can be exponentially large. Formally, there is a bijection between $\overline{\mathscr{A}}$ and the following set of solutions of a multiple knapsack problem with incompatibility constraints:

$$\sum_{i \in V} r_{ik} a_i \leq R_k \quad (k \in \mathscr{R})$$

$$a_i + a_j \leq 1 \quad ((i,j) \in TE)$$

$$a_i \in \{0,1\} \quad (i \in V)$$

where a_i is a binary variable indicating whether activity i belongs to the feasible subset and TE is the transitive closure of the precedence constraints E. With this remark, the (FSS) formulation can be defined as an integer Dantzig–Wolfe decomposition of the resource constraints on the (DT) formulation (Vanderbeck 2000).

It has to be remarked that since the y variables are on/off variables, we have $\mu_{it} = \sum_{A \in \overline{\mathscr{A}}_i} y_{At}$ for any activity i such that $p_i \geq 1$. Hence, a stronger formulation than the (FSS) formulation can be obtained by substitution of variables μ_{it} and removal of resource constraints (2.20) in the (OODDT) formulation. Since it is an integer Dantzig–Wolfe decomposition of (OODDT), the obtained formulation is stronger, at the expense of an exponential increase of the number of variables. Column-generation-based lower bounding techniques have been proposed on relaxed variants of this formulation (see Chap. 3 of this handbook and Baptiste and Demassey 2004; Brucker and Knust 2012).

2.2.6 The Chain-Decomposition Formulation

Kimms (2001) introduced a model for resource-constrained project scheduling with financial objectives. The model is based on the decomposition of the precedence constraints into chains. We present here an adapted version of the formulation for the makespan objective. Let $\mathscr{P}$ denote the set of considered chains. The set $\mathscr{P}$ is such that each precedence constraint is included in one and only one chain, and such that, for each chain $P \in \mathscr{P}$, any arc $(i, j) \in P$ corresponds to a precedence constraint in E. The formulation (ND) below uses a binary variable β_{sP} equal to one if and only if schedule s is selected for chain P. Indeed, given a chain and an upper bound on the makespan, it is in theory possible to consider all integer schedules $\mathscr{S}_P$ respecting the precedence constraints of chain P. However, there is clearly an exponential number of variables β_{sP}. A binary parameter a_{itsP} (given input data) is equal to one if and only if activity i completes at t in schedule s of chain P. Let C_{sP} give the makespan of schedule s for chain P. As an activity can be involved in more than one precedence constraint, there may be more than one chain in which each activity appears. Let $P(i)$ denote the chain of smallest index (assuming that chains are numbered arbitrarily) in which i appears. The (ND) formulation is given as follows:

$$\text{Min. } C_{max} \tag{2.46}$$

$$C_{max} \geq \sum_{s\in\mathscr{S}_P} C_{sP}\,\beta_{sP} \quad (P \in \mathscr{P}) \tag{2.47}$$

$$\sum_{s\in\mathscr{S}_P} \beta_{sP} = 1 \quad (P \in \mathscr{P}) \tag{2.48}$$

$$\sum_{t=EC_i}^{LC_i} t \left(\sum_{s\in\mathscr{S}_{P(i)}} a_{itsP(i)}\,\beta_{sP(i)} - \sum_{s\in\mathscr{S}_P} a_{itsP}\,\beta_{sP} \right) \quad (i \in V;\ P \in \mathscr{P} \setminus P(i)) \tag{2.49}$$

$$\sum_{i\in V} \sum_{\tau=t}^{t+p_i-1} \sum_{s\in\mathscr{S}_{P(i)}} r_{ik}\,a_{i\tau sP(i)}\,\beta_{sP(i)} \leq R_k \quad (k \in \mathscr{R};\ t \in H) \tag{2.50}$$

$$\beta_{sP} \in \{0, 1\} \quad (P \in \mathscr{P};\ s \in \mathscr{S}_P) \tag{2.51}$$

Constraints (2.47) state that the makespan must be larger than the length of the schedule selected for all chains. Constraints (2.48) express that exactly one schedule must be selected for each chain. Constraints (2.49) synchronize the chains in the sense that the completion time of each activity must be the same in all chains in which it appears (here, chain $P(i)$ is used as a reference). Constraints (2.50) are the resource constraints.

As the number of variables is exponential, this formulation cannot be used directly. Kimms (2001) proposed a column-generation procedure to compute an upper bound of the net present value (NPV) maximization objective. Note that if (FSS) can be seen as a Dantzig–Wolfe decomposition of resource constraints, (ND) can be in a complementary manner seen as a (partial) Dantzig–Wolfe decomposition of precedence constraints.

2.3 Sequencing and Natural-Date Formulations

Sequencing and *natural-date* formulations are MILP that involve at least two categories of variables. For each activity i, a continuous natural-date variable S_i gives the start time of activity i, while for each pair of activities $i \in V, j \in V \setminus \{i\}$, a binary sequencing variable z_{ij} is defined such that $z_{ij} = 1$ if the completion of activity i is scheduled before starting activity j, more precisely, if $S_j \geq S_i + p_i$. As remarked by Queyranne and Schulz (1994), this formulation is issued from the earliest studies of the linear ordering polytope (where in this case activities cannot overlap) and yielded the *disjunctive* MILP formulation of the job-shop scheduling problem, studied among others by Applegate and Cook (1991). The formulation is also conceptually equivalent to the disjunctive graph representation of the job-shop problem (Adams et al. 1988).

Note that there are at most $n(n-1)/2$ sequencing variables and n natural-date variables, which yields a polynomial number of variables.

2.3.1 The Minimal-Forbidden-Set-Based Formulation

For the RCPSP, Alvarez-Valdés and Tamarit (1993) proposed a first formulation of this category, only based on the sequencing variables. The formulation is based on a dual concept of the feasible-subset concept: the *minimal forbidden set*. A minimal forbidden set is a set of activities that exceed the availability of at least one resource while any proper subset satisfies all resource constraints. There is also in general an exponential number of minimal forbidden sets. The minimal forbidden sets of cardinality Q are the integer points of the following polytope:

$$\sum_{i \in V} b_{iq} = 1 \quad (q = 1, \ldots, Q) \tag{2.52}$$

$$\sum_{q=1}^{Q} b_{iq} \leq \hat{a}_i \quad (i \in V) \tag{2.53}$$

$$\sum_{k \in \mathscr{R}} \sigma_k \geq 1 \tag{2.54}$$

$$\sum_{i \in V} r_{ik}\, \hat{a}_i \geq (R_k + 1)\sigma_k \quad (k \in \mathscr{R}) \tag{2.55}$$

$$\sum_{i \in V} \hat{a}_i = Q \tag{2.56}$$

$$\sum_{i \in V} r_{ik}(\hat{a}_i - b_{iq}) \leq R_k \quad (k \in \mathscr{R};\ q = 1, \ldots, Q) \tag{2.57}$$

$$\hat{a}_i + \hat{a}_j \leq 1 \quad ((i, j) \in TE) \tag{2.58}$$

$$\hat{a}_i \in \{0, 1\} \quad (i \in V) \tag{2.59}$$

$$b_{iq} \in \{0, 1\} \quad (i \in V;\ q = 1, \ldots, Q) \tag{2.60}$$

$$\sigma_k \in \{0, 1\} \quad (k \in \mathscr{R}) \tag{2.61}$$

where $\hat{a}_i$ is the binary variable (2.59) indicating if activity i belongs to the forbidden set, b_{iq} is a binary variable (2.60) used to ensure that each subset of the minimal forbidden set is a feasible set and σ_k is a binary variable (2.61) indicating whether the set is forbidden with respect to resource k availability. A forbidden set of cardinality Q has $|Q|$ inclusion-maximal subsets, each being obtained by removing one activity. By definition of a minimal forbidden set, each such maximal subset must be a feasible set, as stated by constraint (2.57) explained below. For a maximal subset q, the unique activity i such that $b_{iq} = 1$, as stated by Constraints (2.52), identifies the subset obtained by removing i from the forbidden set. Constraints (2.53) ensure that the removed activity belongs to the forbidden set and that each activity identifies at most one subset. There must be at least one resource such that the set is forbidden and this is ensured by constraints (2.54, 2.55). The cardinality Q of the forbidden set is enforced by constraint (2.56). Constraints (2.57) state that each of the maximal subset q must be feasible, as the unique non zero b_{iq} subtracted from the corresponding $\hat{a}_i$ makes the l.h.s. equal to the total resource requirement of the subset. Constraints (2.58) prevent two activities linked by a precedence constraint to belong to the same forbidden set. Constraints (2.59–2.61) define the binary variables. Note that Stork and Uetz (2005) proposed an algorithm to enumerate all minimal forbidden sets based on the equivalence of this enumeration with the generation of all circuits of an independence system.

Let $\mathscr{F}$ denote the set of all minimal forbidden sets as defined above. With this concept, Alvarez-Valdés and Tamarit (1993) propose the following formulation (FS):

$$\text{Min. } S_{n+1} \tag{2.62}$$

$$\text{s.t.} \quad z_{ij} + z_{ji} \leq 1 \quad (i, j \in V,\ i < j) \tag{2.63}$$

$$z_{ij} + z_{jh} - z_{ih} \leq 1 \quad (i, j, h \in V,\ i \neq j \neq h) \tag{2.64}$$

$$z_{ij} = 1 \quad ((i,j) \in E) \tag{2.65}$$

$$S_j - S_i - M_{ij} z_{ij} \geq p_i - M_{ij} \quad (i,j \in V,\ i \neq j) \tag{2.66}$$

$$\sum_{i,j \in F, i \neq j} z_{ij} \geq 1 \quad (F \in \mathscr{F}) \tag{2.67}$$

$$z_{ij} \in \{0,1\} \quad (i,j \in V,\ i \neq j) \tag{2.68}$$

$$ES_i \leq S_i \leq LS_i \quad (i \in V) \tag{2.69}$$

The objective is to minimize the makespan (2.62). Constraints (2.63) and (2.64) forbid the existence of cycles of length 2 and $\geq$ 3 in the sequencing decisions, respectively. Constraints (2.65) enforce that the sequencing decisions are compatible with the precedence constraints. Constraints (2.66) link the sequencing decisions with the start time variables. The constant M_{ij} must be a valid upper bound for $p_i + S_j - S_i$ and can be set consequently to $M_{ij} = p_i + LS_j - ES_i$. As underlined by Queyranne and Schulz (1994) among others, the presence of such big-M constraints yield poor LP relaxations. Constraints (2.67) state that for each minimal forbidden set $F \in \mathscr{F}$, there is at least one sequencing decision that prevents all activities of F from being in progress simultaneously. Constraints (2.68) and (2.69) define the binary sequencing variables and the natural-date variables, respectively.

Due to the forbidden-set structure, formulation (FS) contains an exponential number of constraints (2.67) and has, to our knowledge, never been used directly in practice.

2.3.2 *The Flow-Based Formulation*

Artigues et al. (2003) introduced a formulation involving in addition, for each pair of activities $i \in V$, $j \in V \setminus \{i\}$ and for each resource $k \in \mathscr{R}$, a continuous flow variable ϕ_{ij}^k giving the amount of resource k *transferred* from i to j. The flow variables allow to get rid of the forbidden-set constraints (2.67), replacing them by the following constraints:

$$\phi_{ij}^k - \min(\tilde{r}_{ik}, \tilde{r}_{jk}) z_{ij} \leq 0 \quad (i,j \in V,\ i \neq j;\ k \in \mathscr{R}) \tag{2.70}$$

$$\sum_{j \in V \setminus \{i\}} \phi_{ij}^k = \tilde{r}_{ik} \quad (i \in V \setminus \{n+1\}) \tag{2.71}$$

$$\sum_{i \in V \setminus \{j\}} \phi_{ij}^k = \tilde{r}_{jk} \quad (j \in V \setminus \{0\}) \tag{2.72}$$

$$0 \leq \phi_{ij}^k \leq \min(\tilde{r}_{ik}, \tilde{r}_{jk}) \quad (i,j \in V,\ i \neq n+1,\ j \neq 0,\ i \neq j;\ k \in \mathscr{R}) \tag{2.73}$$

Constraints (2.70) link the flow variables and the sequencing variables. Constraints (2.71) are the outflow constraints stating that each activity i, except activity $n + 1$, must send $\tilde{r}_{ik}$ resource units to other activities, for each resource $k \in \mathscr{R}$. Constraints (2.72) stipulate the number of units of each resource k that must be received by each activity j. Note that we set value of the parameter $\tilde{r}_{ik}$ to r_{ik} if $0 < i < n + 1$, and to R_k if $i = 0$ or $i = n + 1$. Constraints (2.73) define the continuous flow variables.

The formulation (FL) is obtained by combining the objective (2.62) and constraints (2.63–2.66, 2.68, 2.69, 2.70–2.73). It involves a polynomial number of variables and constraints. Despite its poor LP relaxation, it is an interesting alternative to the time-indexed formulations to solve exactly or approximately instances of small size featuring large time horizons (Koné et al. 2011). Valid inequalities were proposed by Demassey et al. (2005). The (FS) and (FL) formulations are also of interest for stochastic (see Chap. 37 in the second volume of this handbook) and robust (see Chap. 40 in the second volume of this handbook) recourse-based approaches considering uncertain processing times. Sequencing and flow variables can be used as scenario-independent first level variables while the start times are the recourse variables.

2.4 Positional-Date and Assignment Formulations

Positional-date and *assignment* formulations (also called *event-based* formulations) also involve two categories of variables. The time horizon is pre-decomposed into a set of *positions* or *events* $\mathscr{E}$. For each event $e \in \mathscr{E}$, a positional-date continuous variable t_e gives the time at which event e occurs. Originally, event-based formulations were proposed for machine scheduling (see Sect. 5 in Queyranne and Schulz 1994), and for batch scheduling in the process industry. In the latter category, we refer to a survey by Mouret et al. (2011) and also to the formulations by Zapata et al. (2008) and Kyriakidis et al. (2012) that are explicit adaptations of process scheduling formulations of the (multi-mode) RCPSP.

Events correspond to start or end times of activities. In any left-shifted schedule for the RCPSP, with finish-to-start precedence relations, with zero time lag, the start time of an activity is either 0 or it coincides with the end time of some other activity. Furthermore, it can be straightforwardly shown that the set of left-shifted (or semi-active) schedules is dominant. Consequently, the number of events can be restricted to the number of activities plus one. Let $\mathscr{E} = \{0, 1, \ldots, n\}$ be the index set of the events. Event-based formulations do not require the use of dummy activities. Consequently, the number of activities to be considered is n (instead of $n + 2$ for all the preceding formulations).[1] Event-based formulations, as well as (FL), have

[1] However, to simplify the presentation, we use set V, which includes the dummy activities, in the formulations.

also the advantage of coping with instances containing some non-integer activity processing times. More importantly, for instances with long-enough scheduling horizon, event-based models involve fewer variables compared to the models indexed by time.

Zapata et al. (2008) propose, for a multi-mode RCPSP, a first formulation involving three categories of binary assignment variables: a pulse start variable a_{ie}^{+} which indicates whether activity i starts at event e, a pulse end variable a_{ie}^{-}, which indicates whether activity i completes at event e and an on/off variable $\overline{a}_{ie}$ which indicates whether activity i is in progress at event e. In fact, there is no need to consider simultaneously all three types of variables to model the RCPSP. However, it can be easily seen that it is not possible to model the problem with a single pulse start variable a_{ie}^{+} (or end variable a_{ie}^{-}). Indeed, knowing that an activity starts (or ends) at event e does not automatically identify the events that overlap with its processing window since time slots between events are variable. We propose below a start/end formulation (involving variables a_{ie}^{+} and a_{ie}^{-}) and then an on/off formulation (involving only one type of variables, the on/off variables $\overline{a}_{ie}$'s. We also refer to the process-scheduling-based formulation by Kyriakidis et al. (2012) that in fact amounts to considering both start and on/off variables.

2.4.1 The Start/End Event-Based Formulation

The start/end event-based formulation (SEE) makes use of binary variables a_{ie}^{-} and a_{ie}^{+}. It was initially proposed by Koné et al. (2011) and corrected by Artigues et al. (2013). It also involves continuous event date variables t_e, $e \in \mathscr{E}$, and continuous event resource-usage variables b_{ek}, $e \in \mathscr{E}$, $k \in \mathscr{R}$.

$$\text{Min. } t_n \tag{2.74}$$

$$\text{s. t. } \quad t_0 = 0 \tag{2.75}$$

$$t_{e+1} - t_e \geq 0 \quad (e \in \mathscr{E} \setminus \{n\}) \tag{2.76}$$

$$t_f - t_e - p_i\, a_{ie}^{+} + p_i(1 - a_{if}^{-}) \geq 0 \quad (i \in V;\ e, f \in \mathscr{E},\ e < f) \tag{2.77}$$

$$\sum_{e \in \mathscr{E}} a_{ie}^{+} = 1 \quad (i \in V) \tag{2.78}$$

$$\sum_{e \in \mathscr{E}} a_{ie}^{-} = 1 \quad (i \in V) \tag{2.79}$$

$$\sum_{v=0}^{e} a_{iv}^{-} + \sum_{v=e}^{n} a_{iv}^{+} \leq 1 \quad (i \in V;\ e \in \mathscr{E}) \tag{2.80}$$

$$\sum_{e'=e}^{n} a_{ie'}^{-} + \sum_{e'=0}^{e-1} a_{je'}^{+} \leq 1 \quad ((i, j) \in E;\ e \in \mathscr{E}) \tag{2.81}$$

$$b_{0k} - \sum_{i \in V} r_{ik}\, a^+_{i0} = 0 \quad (k \in \mathscr{R}) \tag{2.82}$$

$$b_{ek} - b_{e-1,k} + \sum_{i \in V} r_{ik}(a^-_{ie} - a^+_{ie}) = 0 \quad (e \in \mathscr{E} \setminus \{0\};\ k \in \mathscr{R}) \tag{2.83}$$

$$b_{ek} \leq R_k \quad (e \in \mathscr{E};\ k \in \mathscr{R}) \tag{2.84}$$

$$ES_i\, a^+_{ie} \leq t_e \leq LS_i\, a^+_{ie} + LS_{n+1}(1 - a^+_{ie}) \quad (i \in V;\ e \in \mathscr{E}) \tag{2.85}$$

$$(ES_i + p_i)a^-_{ie} \leq t_e \quad (i \in V;\ e \in \mathscr{E}) \tag{2.86}$$

$$t_e \leq (LS_i + p_i)a^-_{ie} + LS_{n+1}(1 - a^-_{ie}) \quad (i \in V;\ e \in \mathscr{E}) \tag{2.87}$$

$$ES_{n+1} \leq t_n \tag{2.88}$$

$$a^+_{ie}, a^-_{ie} \in \{0, 1\} \quad (i \in V;\ e \in \mathscr{E}) \tag{2.89}$$

$$t_e \geq 0 \quad (e \in \mathscr{E}) \tag{2.90}$$

$$b_{ek} \geq 0 \quad (e \in \mathscr{E};\ k \in \mathscr{R}) \tag{2.91}$$

The objective function (2.74) consists in minimizing the completion time t_n of an activity processed last. The single constraint (2.75) indicates that the the first event starts at time 0, while constraints (2.76) stipulates a convention for the ordering of the events. Inequalities (2.77) ensure that if $a^+_{ie} = a^-_{if} = 1$, i.e., i starts at event e and completes at event f, then $t_f \geq t_e + p_i$ holds. For all other combinations of values for a^+_{ie} and a^-_{if} we have either $t_f \geq t_e$ or $t_f \geq t_e - p_i$, which are covered by (2.76). Constraints (2.78) and (2.79) guarantee that each activity starts and ends exactly once. Constraints (2.80) state that an activity cannot simultaneously start at event e or after and finish at event e or before.

Constraints (2.81) ensure that the precedence constraints are respected: If a predecessor i of j ends at event e or later, i.e., if $\sum_{e'=e}^{n} a^-_{ie} = 1$, then $\sum_{e'=0}^{e-1} a^+_{je'}$ must be zero, i.e., j cannot start before event e. Equalities (2.82) set the initial quantities of resources k needed immediately after time 0. Equalities (2.83) describe the recursion for calculating the b_{ek} values for the other events. More precisely, the quantity of resource k needed immediately after time t_e is equal to the quantity of resource k needed immediately after time t_{e-1} plus the quantity of resource k needed by the activities starting at time t_e minus the quantity of resource k needed by the activities completing at time t_e. Constraints (2.84) limit the quantity of resource k needed immediately after time t_e to the availability of resource k. Inequalities (2.85) and (2.86) ensure that an activity cannot start neither before its earliest starting time nor after its latest starting time. Furthermore, (2.88) says that the project cannot end before the earliest start time of the dummy finishing activity $n + 1$.

The (SEE) formulation involves a polynomial number of variables and constraints.

2.4.2 The On/Off Event-Based Formulation

The on/off event-based formulation, noted (OOE), proposed by Koné et al. (2011) involves the above-defined on/off binary variables, $\overline{a}_{ie}$. It also makes use of the continuous event date variables, t_e, $e \in \mathscr{E}$.

$$\text{Min. } C_{max} \tag{2.92}$$

$$\text{s.t. } C_{max} \geq t_e + (\overline{a}_{ie} - \overline{a}_{i(e-1)})p_i \quad (e \in \mathscr{E} \setminus \{0\};\ i \in V) \tag{2.93}$$

$$t_0 = 0 \tag{2.94}$$

$$t_{e+1} \geq t_e \quad (e \in \mathscr{E} \setminus \{n\}) \tag{2.95}$$

$$t_f \geq t_e + (\overline{a}_{ie} - \overline{a}_{i,e-1} - \overline{a}_{if} + \overline{a}_{i,f-1} - 1)p_i \quad ((e, f, i) \in \mathscr{E}^2 \times V,\ f > e \neq 0) \tag{2.96}$$

$$\sum_{e'=0}^{e-1} \overline{a}_{ie'} \leq e(1 - \overline{a}_{ie} + \overline{a}_{i,e-1})) \quad (e \in \mathscr{E} \setminus \{0\}) \tag{2.97}$$

$$\sum_{e'=e}^{n} \overline{a}_{ie'} \leq (n - e + 1)(1 + \overline{a}_{ie} - \overline{a}_{i,e-1}) \quad (e \in \mathscr{E} \setminus \{0\}) \tag{2.98}$$

$$\sum_{e \in \mathscr{E}} \overline{a}_{ie} \geq 1 \quad (i \in V) \tag{2.99}$$

$$\overline{a}_{ie} + \sum_{e'=0}^{e} \overline{a}_{je'} \leq 1 + (1 - \overline{a}_{ie})e \quad (e \in \mathscr{E};\ (i, j) \in E) \tag{2.100}$$

$$\sum_{i \in V} r_{ik} \overline{a}_{ie} \leq R_k \quad (e \in \mathscr{E};\ k \in \mathscr{R}) \tag{2.101}$$

$$ES_i\, \overline{a}_{ie} \leq t_e \quad (e \in \mathscr{E};\ i \in V) \tag{2.102}$$

$$t_e \leq LS_i(\overline{a}_{ie} - \overline{a}_{i,e-1}) + LS_{n+1}(1 - (\overline{a}_{ie} - \overline{a}_{i,e-1})) \quad (e \in \mathscr{E} \setminus \{0\};\ \forall i \in V) \tag{2.103}$$

$$t_e \geq 0 \quad (e \in \mathscr{E}) \tag{2.104}$$

$$\overline{a}_{ie} \in \{0, 1\} \quad (i \in A;\ e \in \mathscr{E}) \tag{2.105}$$

Constraints (2.93) link the makespan to the event dates: $C_{max} \geq t_e + p_i$ if i is in progress at event e but not at event $e - 1$, i.e., if i starts at event e. Constraints (2.94,2.95) set the event sequencing. Constraints (2.96) link the binary optimization variables $\overline{a}_{ie}$ to the continuous optimization variables t_e and ensure that, if activity i starts immediately after event e and ends at event f, then the date of event f is at least equal to the date of event e plus the processing time of activity i ($t_f \geq t_e + p_i$). The validity of these constraints follows the same logic as for constraints (2.77). Constraints (2.97) and (2.98), called *contiguity constraints*, ensure non-preemption (the events after which a given activity is being processed must be adjacent)—for a formal proof, we refer to Koné et al. (2011). Constraints (2.99) ensure that each activity is processed at least once during the project. Constraints (2.100) describe each precedence constraint $(i, j) \in E$, modeling the expression $(\overline{a}_{ie} = 1) \implies (\sum_{e'=0}^{e} \overline{a}_{je} = 0)$ for each event e. Constraints (2.101) are the resource constraints limiting the total demand of activities in progress at each event. Constraints (2.102) and (2.103) set the start time of any activity i between its earliest start time, ES_i, and its latest start time, LS_i.

The on/off event-based formulation involves also a polynomial number of variables and constraints but only half the number of variables of the (SEE) formulation. (OOE) would also need only n events instead of $n+1$ events for (SEE) although we did not include this feature to simplify the presentation. Recently, Koné et al. (2013) provided an on/off event-based formulation for the RCPSP with consumption and production of resources.

2.4.3 More Event-Based Formulations

As mentioned above, event-based formulations have been significantly studied in the process-engineering literature. Kyriakidis et al. (2012) present a MILP formulation based on the Resource Task Network (RTN) representation of batch processes. They decompose the time horizon into time slots of variable lengths. The concept of time slot is equivalent to the concept of event (an event is one of the boundaries of a time slot). Suppose we identify a time slot by its start event. The model uses start variables a_{ie}^{+} and a new on/off binary variable, $\tilde{a}_{ie}$, slightly different from $\bar{a}_{ie}$ presented above as $\tilde{a}_{ie}$ indicates whether the activity is active at $e-1$ and at e. In other words, $\tilde{a}_{ie}$ is equal to 1 for each event at which activity i is active except the event at which i is started. Formally we have the equivalence

$$\tilde{a}_{ie} = \bar{a}_{ie} - a_{ie}^{+} \quad (i \in V;\ e \in \mathscr{E})$$

Kyriakidis et al. (2012) also use a new continuous variable for time slot duration, τ_e. For an event $e < |\mathscr{E}|$, the time slot duration variable equals $t_{e+1} - t_e$.

We call this formulation (SOOE). To link the time slot duration variables to the activity duration variables, they introduce and linearize the non-linear constraint (2.106) below in replacement of constraints (2.77) of the (SEE) formulation and constraints (2.96) of the (OOE) formulation.

$$\sum_{e \in \mathscr{E}} (a_{ie}^{+} + \tilde{a}_{ie})\tau_e = p_i \quad (i \in V;\ e \in \mathscr{E}) \tag{2.106}$$

Contiguity constraints also take the following form:

$$a_{ie}^{+} + \tilde{a}_{ie} \geq \tilde{a}_{i,e+1} \quad (i \in V;\ e \in \mathscr{E}) \tag{2.107}$$

which is simpler than constraints (2.97) and (2.98) of the (OOE) formulation. These constraints (2.107) could in turn be translated into

$$\bar{a}_{ie} \geq \bar{a}_{i,e+1} - a^+_{ie} \quad (i \in V;\ e \in \mathscr{E})$$

Last, an interesting unified modeling of precedence and resource constraints via resource production and consumption constraints is used. Precedence and resource constraints are replaced by excess resource balance and excess resource limitation constraints similar to constraints (2.82–2.83) of the (SEE) model:

$$b_{ek} = b_{e,k-1} + \sum_{i \in I_k} \left(r^-_{ik}\, a^+_{ie} + r^+_{ik} (\tilde{a}_{i,e-1} + a^+_{i,e-1} - \tilde{a}_{ie}) \right) \quad (k \in \overline{\mathscr{R}};\ e \in \mathscr{E}) \tag{2.108}$$

$$R^{min}_k \leq b_{ek} \leq R^{max}_k \quad (k \in \overline{\mathscr{R}};\ e \in \mathscr{E}) \tag{2.109}$$

The set $\overline{\mathscr{R}}$ contains the set of renewable resources and a fictitious resource for each precedence constraint. For a renewable resource $k \in \overline{\mathscr{R}} \cap \mathscr{R}$, we have $r^-_{ik} = r^+_{ik} = r_{ik}$. Furthermore, the parameter R^{min}_k is set to 0 and R^{max}_k is set to R_k. A fictitious event $e = -1$ such that $b_{-1k} = R_k$ for each resource $k \in \mathscr{R}$ has to be introduced. Constraints (2.108) are equivalent to resource balance constraints (2.83) as the factor $\tilde{a}_{i,e-1} + a^+_{i,e-1} - \tilde{a}_{ie}$ is equal to 1 if activity i ends at event e. If $k \in \overline{\mathscr{R}} \setminus \mathscr{R}$, it corresponds to a precedence constraints (i, j). We then set $r^-_{jk} = 1$, $r^+_{ik} = 1/|\{l \in V \mid (l, j) \in E\}|$, $r^-_{ik} = r^+_{jk} = 0$, $R^{min}_k = 0$, $R^{max}_k = 1$, and $b_{-1k} = 0$. The successor j, which consumes units of k at its start time will not be able to start until one resource unit has been produced, i.e., when all its predecessors have been completed.

This model is interesting as it can be extended to more general resource models, especially to problems involving resource production and consumption. However, it contains twice as many binary variables as the (OOE) model.

2.5 Synthesis of Theoretical and Experimental Comparisons

As already mentioned throughout the chapter, the various ILP formulations can be compared in terms of their relative LP relaxation strengths. Another way of comparing the formulations is through experimental evaluation of integer solving procedures, mostly via ILP solvers. Koné et al. (2011) performed an experimental comparison of (DT, DDT, FL, SEE, OOE) on various benchmark instances, both in terms of LP relaxations and integer solving.

We must first mention that, without any additional valid inequalities, the LP relaxations of the compact formulations presented in this chapter are all very poor. Koné et al. (2011) report that the lower bounds obtained by solving the LP relaxations of the flow-based (FL) and event-based (SEE, OOE) formulations never exceed the critical path length in the precedence graph. For the extended (FS) formulation, we are not aware of experiments assessing the quality of its LP-relaxation directly except that it served as a basis for generating constraint propagation-based cutting planes in Demassey et al. (2005), but the forbidden set constraints were included for only minimal forbidden sets of cardinality equal to 2 or 3.

Discrete-time formulations are known to yield better LP relaxations, keeping in mind that aggregated formulations (DT, SDT, OODT) have weaker relaxations than disaggregated formulations (DDT, SDDT, OODDT) but involves fewer constraints. The (FSS) extended feasible-subset-based time-indexed formulation features by construction a better LP relaxation than the (DDT) formulation and is consequently the strongest formulation presented in this chapter in terms of LP relaxation. As a counterpart, it involves an exponential number of variables (see also Chap. 3 of this handbook for lower bounds based on relaxations of this formulation). No theoretical nor experimental study has been carried out to our knowledge for the chain-decomposition formulation (ND).

All discrete-time formulations involve a pseudo-polynomial number of variables and constraints, which can significantly slow down the solving process, especially for large problems and/or problems involving large time horizons. According to Koné et al. (2011), the discrete-time formulations outperform the flow-based and event-based formulations for integer solving on small to medium-sized instances (from 30 to 60 activities) with small scheduling horizons. However, for instances involving large scheduling horizons the flow-based and event-based formulations (especially OOE) in turn outperform the discrete-time formulations for integer solving, although the results of these ILP formulations in terms of optimality gap is not particularly good. The instances tested were separated into two categories with respect to resource considerations, as introduced by Baptiste and Le Pape (2000). The first category is made of highly "disjunctive" instances, where, due to precedence or resource constraints, an important number of activities cannot be pairwise scheduled in parallel. The second category consists of highly "cumulative" instances where, on the contrary, many activities can be scheduled in parallel. Koné et al. (2011) report that the flow-based formulation tends to be better than the event-based formulation for highly disjunctive instances, while the reverse statement applies for highly cumulative instances.

Bianco and Caramia (2013) present an experimental comparison of the pulse discrete-time formulation (DT), the step disaggregated discrete-time formulation (SDDT), the forbidden-set-based formulation (FS), the feasible-subset-based formulation (FSS) and their variant of the disaggregated step discrete-time formulation (SDDT3). They show that, on the tested instances with 60, 90, and 120 activities from the PSPLIB,[2] (SDDT3) generally outperforms the other formulations in terms of solution time and quality. For the instances with 90 or 120 activities, they report that the (SDDT) also performs well. For the smaller instances, the (FS) and (FSS)

[2] www.om-db.wi.tum.de/psplib/main.html.

formulations, for which all variables and constraints are explicitly generated, are competitive. However, on the larger instances, too much time is needed to load the model. Column-generation techniques for (FSS) and cutting-plane algorithms for (FS) could be used to improve this result.

2.6 Conclusions

We conclude by noting that recent success in solving the RCPSP was obtained by SAT-inspired techniques or, more precisely, constraint-programming solvers incorporating no-good learning (see, e.g., Horbach 2010; Schutt et al. 2009, 2011, and Chap. 7 of this handbook). These methods use SAT encodings close to the time-indexed and event-based formulations. As suggested by Artigues et al. (2013), this can give rise to successful hybrid methods exploiting the strengths of both conflict-based clause learning of SAT techniques and LP relaxations.

References

Adams J, Balas E, Zawack D (1988) The shifting bottleneck procedure for job shop scheduling. Manage Sci 34(3):391–401

Alvarez-Valdés R, Tamarit JM (1993) The project scheduling polyhedron: dimension, facets and lifting theorems. Eur J Oper Res 67:204–220

Applegate D, Cook W (1991) A computational study of the job-shop scheduling problem. ORSA J Comput 3(2):149–156

Artigues C (2013) A note on time-indexed formulations for the resource-constrained project scheduling problem. Technical Report 13206, LAAS, CNRS, Toulouse

Artigues C, Michelon P, Reusser S (2003) Insertion techniques for static and dynamic resource constrained project scheduling. Eur J Oper Res 149(2):249–267

Artigues C, Demassey S, Néron E (2008) Resource-constrained project scheduling: models, algorithms, extensions and applications. ISTE Ltd, London; Wiley, Hoboken

Artigues C, Brucker P, Knust S, Koné O, Lopez P, Mongeau M (2013) A note on “Event-based MILP models for resource-constrained project scheduling problems”. Comp Oper Res 40(4):1060–1063

Baptiste P, Demassey S (2004) Tight LP bounds for resource constrained project scheduling. OR Spectr 26(2):251–262

Baptiste P, Le Pape C (2000) Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems. Constraints 5(1–2):119–39

Bianco L, Caramia M (2013) A new formulation for the project scheduling problem under limited resources. Flex Serv Manuf J 25:6–24

Brucker P, Knust S (2012) Complex scheduling. Springer, Berlin

Cavalcante CCB, de Souza CC, Savelsbergh MWP, Wang Y, Wolsey LA (2001) Scheduling projects with labor constraints. Discrete Appl Math 112(1–3):27–52

Christofides N, Alvarez-Valdés R, Tamarit J (1987) Project scheduling with resource constraints: a branch and bound approach. Eur J Oper Res 29:262–273

de Souza CC, Wolsey LA (1997) Scheduling projects with labour constraints. Relatório Técnico IC-P7-22. Instituto de Computação, Universidade Estadual de Campinas

Demassey S, Artigues C, Michelon P (2005) Constraint propagation-based cutting planes: an application to the resource-constrained project scheduling problem. INFORMS J Comput 17(1):52–65

Demeulemeester E, Herroelen W (2002) Project scheduling: a research handbook. Kluwer, Dordrecht

Hardin JR, Nemhauser GL and Savelsbergh MW (2008). Strong valid inequalities for the resource-constrained scheduling problem with uniform resource requirements. Discrete Optim 5(1):19–35

Horbach A (2010) A Boolean satisfiability approach to the resource-constrained project scheduling problem. Ann Oper Rer 181:89–107

Kaplan LA (1998) Resource-constrained project scheduling with preemption of jobs. Unpublished Ph.D. dissertation, University of Michigan, USA

Kimms A (2001) Mathematical programming and financial objectives for scheduling projects. Kluwer, Dordrecht

Klein R (2000) Scheduling of resource-constrained projects. Kluwer, Dordrecht

Koné O, Artigues C, Lopez P, Mongeau M (2011) Event-based MILP models for resource-constrained project scheduling problems. Comput Oper Res 38(1):3–13

Koné O, Artigues C, Lopez P, Mongeau M (2013) Comparison of mixed integer linear programming models for the resource-constrained project scheduling problem with consumption and production of resources. Flex Serv Manuf J 25(1–2):25–47

Kyriakidis TS, Kopanos GM, Georgiadis MC (2012) MILP formulations for single- and multi-mode resource-constrained project scheduling problems. Comput Chem Eng 36:369–385

Mingozzi A, Maniezzo V, Ricciardelli S, Bianco L (1998) An exact algorithm for the resource-constrained project scheduling problem based on a new mathematical formulation. Manage Sci 44:714–729

Möhring RH, Schulz AS, Stork F, Uetz M (2001) On project scheduling with irregular starting time costs. Oper Res Lett 28:149–154

Mouret S, Grossmann IE, Pestiaux P (2011). Time representations and mathematical models for process scheduling problems. Comput Chem Eng 35(6):1038–1063

Pritsker A, Watters L (1968) A zero-one programming approach to scheduling with limited resources. The RAND Corporation, RM-5561-PR

Pritsker A, Watters L, Wolfe P (1969) Multi-project scheduling with limited resources: a zero-one programming approach. Manage Sci 16:93–108

Queyranne M, Schulz A (1994) Polyhedral approaches to machine scheduling. Technical Report 408/1994, Technische Universität Berlin, Berlin, Germany

Sankaran JK, Bricker DL, Juang SH (1999) A strong fractional cutting-plane algorithm for resource-constrained project scheduling. Int J Ind Eng 6(2):99–111

Schutt A, Feydy T, Stuckey PJ, Wallace MG (2009) Why cumulative decomposition is not as bad as it sounds. In: Gent IP (ed) Proceedings of principles and practice of constraint programming: CP 2009, Lecture notes in computer science, vol 5732. Springer, Berlin, pp 746–761

Schutt A, Feydy T, Stuckey PJ, Wallace MG (2011) Explaining the cumulative propagator. Constraints 16(3):250–282

Stork F, Uetz M (2005) On the generation of circuits and minimal forbidden sets. Math Prog 102(1):185–203

Vanderbeck F (2000). On Dantzig-Wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm. Oper Res 48(1):111–128

Zapata JC, Hodge BM, Reklaitis GV (2008) The multimode resource constrained multiproject scheduling problem: alternative formulations. Aiche J 54(8):2111–2119

Chapter 3
Lower Bounds on the Minimum Project Duration

Sigrid Knust

Abstract In this chapter methods to calculate lower bounds on the minimum project duration (i.e. the makespan C_{max}) of the basic resource-constrained project scheduling problem $PS \mid prec \mid C_{max}$ are presented. We distinguish between constructive and destructive lower bounds.

Keywords Lower bounds • Makespan minimization • Project scheduling • Resource constraints

3.1 Introduction

In this chapter methods to calculate lower bounds for the resource-constrained project scheduling problem $PS \mid prec \mid C_{max}$ are presented (cf. also Sect. 3.7 in Brucker and Knust 2012). On the one hand, lower bounds are useful to estimate the quality of heuristic solutions (if no exact solution values are available). If LB is a lower bound for an instance and UB is the solution value given by some heuristic, then $UB - LB$ is an upper bound for the distance between the optimal solution value and UB. Furthermore, $\frac{UB-LB}{LB}$ is an upper bound for the relative error. On the other hand, lower bounds are also needed in connection with exact branch-and-bound algorithms in order to reduce the search space. Often, a trade-off between computation times and quality exists.

In general, two types of lower bounds can be distinguished: constructive and destructive bounds. *Constructive* lower bounds are usually provided by directly solving relaxations of the problem, in which some constraints (which make the problem hard) are relaxed. *Destructive* lower bounds are based on considering the corresponding decision version (feasibility problem) of the optimization problem: Given a threshold value T, does a feasible schedule with $C_{max} \leq T$ exist? If we can prove that such a schedule does not exist, then $T + 1$ is a valid lower bound value for the optimization problem supposing that all data are integral. To contradict

S. Knust (✉)
Institute of Computer Science, University of Osnabrück, Osnabrück, Germany
e-mail: sknust@uos.de

C. Schwindt, J. Zimmermann (eds.), *Handbook on Project Management and Scheduling Vol. 1*, International Handbooks on Information Systems,
DOI 10.1007/978-3-319-05443-8_3

(destruct) a threshold value T, again relaxations may be used. If we can state infeasibility for a relaxed problem, obviously the original problem is also infeasible. To find the best lower bound we search for the largest T, where infeasibility can be proved.

The remainder of this chapter is organized as follows. In Sect. 3.2 several constructive lower bounds are described, while in Sect. 3.3 destructive lower bounds are presented. Some conclusions can be found in Sect. 3.4.

3.2 Constructive Lower Bounds

In this section we present some constructive lower bounds which are based on solving different relaxations of problem $PS \mid prec \mid C_{max}$.

A simple constructive lower bound is obtained if we relax all resource constraints and only take into account the precedence constraints (i.e. consider the relaxed problem $PS\infty \mid prec \mid C_{max}$). The optimal makespan of this relaxation is equal to the length of a longest (critical) path in the activity-on-node network, which can be calculated in $\mathcal{O}(|E|)$ time. Usually, this lower bound is denoted by LB_0.

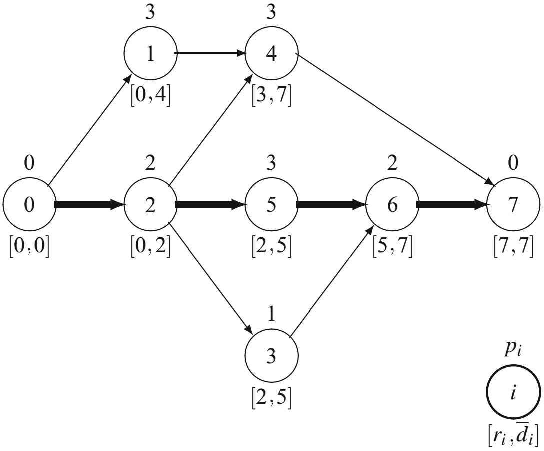
This value was strengthened by Stinson et al. (1978) partially taking into account some resource conflicts as follows. If a schedule with $C_{max} = LB_0$ exists, activity i has to be completed before the deadline $\overline{d}_i := LB_0 + p_i - d_{i,n+1}$, where $d_{i,n+1}$ denotes the length of a longest path from activity i to the dummy end activity $n+1$ in the activity-on-node network (which is a lower bound for the length of the time period between the completion time of i and the optimal makespan). Furthermore, i cannot be started before its release date $r_i := d_{0i}$ corresponding to the length of a longest path from the dummy start activity 0 to activity i. After having identified one critical path P_0, for each activity i not belonging to P_0, let l_i be the maximal length of an interval contained in $[r_i, \overline{d}_i]$ in which i can be processed with the activities from P_0 simultaneously without violating the resource constraints. If $l_i < p_i$ holds, no feasible schedule with $C_{max} = LB_0$ exists and in order to get a feasible schedule the time window of activity i has to be enlarged by at least $p_i^+ := \max\{p_i - l_i, 0\}$ time units. Thus,

$$LB_S := LB_0 + \max_{i \notin P_0} \{p_i^+\}$$

defines a valid lower bound value. For each activity not in P_0 we have to check, whether the resources left by the activities from P_0 are sufficient or not. This can be done in $\mathcal{O}(n^2|\mathcal{R}|)$ time.

Example 3.1. Consider the instance with $n = 6$ activities and two resources with capacities $R_1 = 3$, $R_2 = 1$ shown in Fig. 3.1 (taken from Brucker and Knust 2012).

The critical path $P_0 = (0 \to 2 \to 5 \to 6 \to 7)$ has the length $LB_0 = 7$. In Fig. 3.2 the partial schedule for all critical activities from P_0 is drawn, furthermore, the time windows for the remaining activities $i \notin P_0$ are shown. For these activities



i	1	2	3	4	5	6
p_i	3	2	1	3	3	2
r_{i1}	2	2	0	2	1	2
r_{i2}	1	1	1	1	0	1
r_i	0	0	2	3	2	5
$\overline{d}_i$	4	2	5	7	5	7

Fig. 3.1 A project with $n = 6$ and time windows for $LB_0 = 7$, taken from Brucker and Knust (2012)

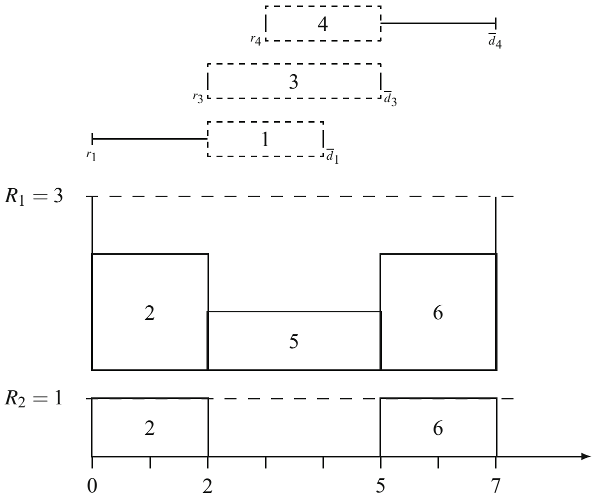


Fig. 3.2 Partial schedule for P_0

we have $l_1 = 2$ since activity 1 can be processed in parallel with P_0 in the interval $[2, 4]$, $l_3 = 3$ since 3 can be processed in the interval $[2, 5]$, and $l_4 = 2$.

Thus, $p_1^+ = 3 - 2 = 1$, $p_3^+ = \max\{1 - 3, 0\} = 0$, $p_4^+ = 3 - 2 = 1$, and

$$LB_S = LB_0 + \max_{i \notin P_0} \{p_i^+\} = 7 + 1 = 8.$$

□

Note that in LB_S only conflicts between activities from P_0 and activities $i \notin P_0$ are taken into account, conflicts among activities $i \notin P_0$ are not regarded. An extension of LB_S was suggested in Demeulemeester (1992) where a critical path together with a second node-disjoint path in the network is considered and a lower bound is determined using dynamic programming.

A simple resource-based lower bound can be determined in $\mathcal{O}(n|\mathcal{R}|)$ time by considering each resource separately. For each renewable resource $k \in \mathcal{R}$ the value $\left\lceil \sum\limits_{i=1}^{n} r_{ik}\, p_i / R_k \right\rceil$ defines a lower bound on the optimal makespan because $\sum\limits_{i=1}^{n} r_{ik}\, p_i$ cannot be greater than the availability $R_k \cdot C_{max}$ of resource k in the interval $[0, C_{max}]$. Thus, the maximum among all resources gives the lower bound

$$LB_1 := \max_{k \in \mathcal{R}} \left\lceil \sum_{i=1}^{n} r_{ik}\, p_i / R_k \right\rceil .$$

A more complicated resource-based lower bound based on Lagrangian relaxation was proposed in Möhring et al. (2003). Here, the resource constraints are relaxed, but violations of them are penalized in the objective function. Lower bounds based on parallel machine scheduling problems have for example been applied in Carlier and Néron (2000) (see also Néron et al. 2006). For this purpose, a subset of $\mu + 1$ activities is calculated which cannot all be processed simultaneously due to the resources. Therefore, at most μ activities can be in progress at the same time, which leads to a parallel machine problem with μ machines. Other constructive lower bounds can be obtained by solving continuous relaxations of integer linear programming formulations (cf. Chap. 2 of this handbook or Artigues et al. 2010). Unfortunately, these relaxations often provide only poor lower bounds.

In the following we consider a relaxation of the RCPSP allowing preemption and partially relaxing the precedence constraints proposed by Mingozzi et al. (1998) (cf. also Chap. 2 of this handbook). This relaxation can be formulated as a linear program where the columns correspond to so-called feasible subsets (or antichains). Later on this formulation was also used to solve the preemptive RCPSP (see Damay et al. 2007 and Chap. 13 of this handbook).

A *feasible antichain* (or feasible subset) A is a subset of activities which may be processed simultaneously with respect to the resource constraints (i.e. $\sum\limits_{i \in A} r_{ik} \leq R_k$ for all $k \in \mathcal{R}$) and among the activities in A no precedence relations exist (i.e. $(i, j) \notin E, (j, i) \notin E$ for all $i, j \in A$). This means that the precedence relations are partially relaxed, since instead of requiring that an activity i has to precede activity j, it is only forbidden to schedule them simultaneously (i.e. j may also be scheduled before i).

Unfortunately, the number of all feasible antichains is very large since it grows exponentially with the number n of activities. In order to reduce this large number, the set of all feasible antichains may be restricted to a smaller subset of so-called non-dominated ones. An antichain A is called *dominated* if it is a proper subset

$A \subset A'$ of another feasible antichain A', otherwise it is called *non-dominated*. In the following we denote the set of all non-dominated feasible antichains by $\mathscr{A}$.

With each subset $A \in \mathscr{A}$ we associate an incidence vector $a^A \in \{0,1\}^n$ defined by

$$a_i^A := \begin{cases} 1, & \text{if activity } i \in A \\ 0, & \text{otherwise.} \end{cases}$$

Furthermore, let $z_A \geq 0$ be a variable denoting the total number of time units when the antichain A is processed. Then the preemptive relaxation can be formulated as the following linear program:

$$\text{Min.} \sum_{A \in \mathscr{A}} z_A \tag{3.1}$$

$$\text{s.t.} \quad \sum_{A \in \mathscr{A}} a_i^A z_A \geq p_i \quad (i = 1, \ldots, n) \tag{3.2}$$

$$z_A \geq 0 \qquad (A \in \mathscr{A}) \tag{3.3}$$

In (3.1) we minimize the makespan $C_{max} = \sum\limits_{A \in \mathscr{A}} z_A$ which is equal to the sum of all processing times of the antichains. The constraints (3.2) ensure that all activities i are processed for at least p_i time units. Here, due to the fact that only non-dominated antichains are considered, we must allow that an activity i may be processed longer than its processing time p_i. It is easy to see that the optimal makespan of this preemptive relaxation cannot be less than LB_0 since due to the precedence constraints all activities belonging to a critical path must be contained in different feasible antichains.

Example 3.2. Consider the instance in Fig. 3.3 with $n = 6$ activities, one resource with capacity $R_1 = 4$ (taken from Brucker and Knust 2012). Here, the length of a critical path is $LB_0 = 4$.

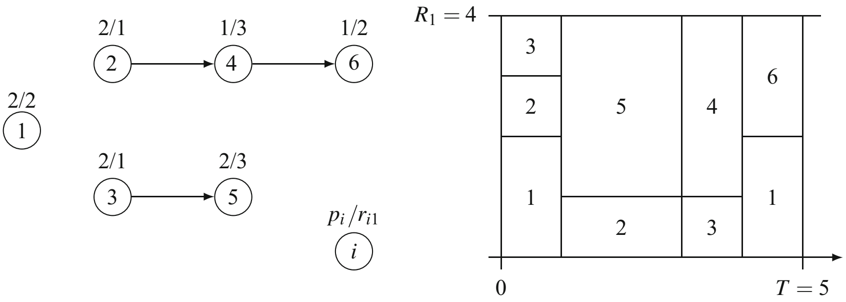


Fig. 3.3 A project and a corresponding optimal preemptive schedule, taken from Brucker and Knust (2012)

For this instance we have 14 feasible antichains: $\{1\}$, $\{1,2\}$, $\{1,2,3\}$, $\{1,3\}$, $\{1,6\}$, $\{2\}$, $\{2,3\}$, $\{2,5\}$, $\{3\}$, $\{3,4\}$, $\{3,6\}$, $\{4\}$, $\{5\}$, $\{6\}$. Among these sets five are non-dominated: $\{1,2,3\}$, $\{1,6\}$, $\{2,5\}$, $\{3,4\}$, $\{3,6\}$.

An optimal preemptive schedule for the relaxation with makespan 5 is shown in Fig. 3.3. Note that all activities $i \neq 2$ are processed for exactly p_i time units, only activity 2 is processed longer for $p_2 + 1 = 3$ time units. This schedule corresponds to the LP solution $z_{\{1,2,3\}} = 1$, $z_{\{1,6\}} = 1$, $z_{\{2,5\}} = 2$, $z_{\{3,4\}} = 1$, $z_{\{3,6\}} = 0$. □

Unfortunately, the number of all non-dominated feasible antichains still grows exponentially with the number n of activities. For $n = 60$ we have approximately 300,000, for $n = 90$ even 8,000,000 columns. For this reason, in Mingozzi et al. (1998) the dual linear program was considered and solved heuristically (which gives weaker bounds than the original LP formulation). However, using the technique of *delayed column generation*, it is not necessary to generate and store all columns. In the following we describe a delayed column generation approach proposed by Baar et al. (1999). In this approach, after solving the LP with the current working set of columns to optimality, we have to look for feasible columns which are able to improve the objective value when entering the basis. If y_i $(i = 1, \dots, n)$ are the values of the dual variables associated with the current basic solution, we have to find a feasible column $a \in \{0,1\}^n$ with $\sum\limits_{i=1}^{n} y_i \cdot a_i > 1$, i.e. a has to satisfy

$$\sum_{i=1}^{n} y_i \cdot a_i > 1 \tag{3.4}$$

$$\sum_{i=1}^{n} r_{ik} \cdot a_i \leq R_k \quad (k \in \mathcal{R}) \tag{3.5}$$

$$a_i + a_j \leq 1 \quad ((i,j) \in E) \tag{3.6}$$

$$a_i \in \{0,1\} \quad (i = 1, \dots, n) \tag{3.7}$$

Conditions (3.5) and (3.6) ensure that a column a corresponds to a feasible antichain (respecting the resource and precedence constraints). The problem of finding a column a maximizing the value $\sum\limits_{i=1}^{n} y_i \cdot a_i$ may be regarded as a multi-dimensional knapsack problem, which can efficiently be solved by a branch-and-bound algorithm (for details see Baar et al. 1999). If for an optimal solution a of this knapsack problem the condition $\sum\limits_{i=1}^{n} y_i \cdot a_i > 1$ holds, we have found an improving column satisfying (3.4). Otherwise, no improving column exists and the whole LP is solved to optimality. Then the column generation process is stopped.

Unfortunately, for problems with a larger number of activities the quality of this lower bound worsens. The reason is that the possibility of preemption allows us to split the activities into many pieces and to put them appropriately into different feasible antichains yielding low objective values. Therefore, in the next section we strengthen this lower bound by additionally taking into account time windows and using a destructive approach.

3.3 Destructive Lower Bounds

In this section we present destructive lower bounds for problem $PS \mid prec \mid C_{max}$. In order to calculate lower bounds in a destructive way, for a given threshold value T we must prove that no feasible schedule with $C_{max} \leq T$ exists (cf. Klein and Scholl 1999). If infeasibility is shown, then $T + 1$ is a valid lower bound value. To find the best lower bound we search for the largest T, where infeasibility can be proved. This can be organized in incremental steps or by binary search. An incremental procedure starts with a valid lower bound value $T = LB$ and increases T in each step by an appropriate value $\Delta \geq 1$ until it is not possible to state infeasibility. When applying binary search, in each step we consider an interval $[L, U]$ in which we search for a valid lower bound value. Initially, we start with a valid upper bound U^0 and a valid lower bound L^0. In each iteration we solve the feasibility problem for $T = \lfloor \frac{U+L}{2} \rfloor$. If we can prove that no feasible solution with $C_{max} \leq T$ exists, we increase the lower bound L to the value $T + 1$ and repeat the procedure with the interval $[T + 1, U]$. Otherwise, if we do not succeed in proving infeasibility for the threshold T, we replace U by T and repeat the procedure in the interval $[L, T]$. The binary search procedure terminates as soon as $L \geq U$ holds after at most $\mathcal{O}(\log(U^0 - L^0))$ steps. Then L equals the largest lower bound value which can be calculated in this way.

Several tests were proposed to detect infeasibility. Many of them are based on time windows $[r_i, \overline{d}_i]$ for the activities which can be derived from the activity-on-node network by longest path calculations. If $[r_i, \overline{d}_i]$ is a time window in which activity i has to be processed completely for p_i time units, obviously $r_i + p_i \leq \overline{d}_i$ must be satisfied (otherwise no feasible schedule exists). Furthermore, we know that i cannot finish before time $EC_i = r_i + p_i$ and cannot start later than time $LS_i = \overline{d}_i - p_i$. Thus, if $EC_i > LS_i$ holds, activity i has to be processed in the interval $[LS_i, EC_i]$ in any feasible schedule (this interval is also called core time of i). If by partially scheduling all mandatory parts of activities during their core times a resource capacity is exceeded, obviously no feasible schedule exists.

In the so-called "*disjunctive interval consistency tests*" (see Dorndorf et al. 1999 or Baptiste et al. 2001) it is checked whether the capacity of certain intervals is sufficient for sets of pairwise incompatible activities (so-called disjunctive sets). A subset $I \subseteq \{1, \ldots, n\}$ of (non-dummy) activities with $|I| \geq 2$ is called a *disjunctive set* if for all $i, j \in I$ with $i \neq j$ a precedence relation $(i, j) \in E$ or $(j, i) \in E$ exists or if i and j cannot be processed simultaneously due to the resource constraints (i.e. if $r_{ik} + r_{jk} > R_k$ for a resource $k \in \mathcal{R}$ holds). For example, for a disjunctive resource (i.e. a renewable resource k with capacity $R_k = 1$), all activities needing this resource constitute a disjunctive set.

A first infeasibility test can be derived from the following result: If there is a subset $J \subseteq I$ with

$$\max_{\mu \in J}\{\overline{d}^{\mu}\} - \min_{\nu \in J}\{r^{\nu}\} < \sum_{i \in J} p_i$$

then obviously no feasible schedule exists since all jobs from the subset J have to be processed in the interval $[\min_{\nu \in J}\{r^{\nu}\}, \max_{\mu \in J}\{\overline{d}^{\mu}\}]$, which does not have the capacity $\sum_{i \in J} p_i$.

Additional precedence relations and smaller time windows for activities of a disjunctive set I may be derived from the following general result:

Theorem 3.1. *Let I be a disjunctive set and $J', J'' \subset J \subseteq I$ with $J' \cup J'' \neq \emptyset$. If*

$$\max_{\substack{\nu \in J \setminus J' \\ \mu \in J \setminus J'' \\ \nu \neq \mu}} (\overline{d}^{\mu} - r^{\nu}) < \sum_{i \in J} p_i \tag{3.8}$$

holds, then in J an activity from J' must start first or an activity from J'' must end last in any feasible schedule.

Based on this theorem, the interval consistency tests shown in Table 3.1 have been proposed (cf. Brucker and Knust 2012 or Baptiste et al. 2001). In the table also the complexity of the best known implementation is listed.

In order to use these techniques for lower bound calculations, usually, a test is not applied only once, but in several iterations until no more constraints can be deduced. This process is also called *constraint propagation*. Due to additional precedence relations and possibly strengthened time windows in later iterations infeasibility may be detected for a given threshold value T.

In case that we cannot prove infeasibility by constraint propagation alone, we may use the time windows and try to prove that no preemptive schedule with $C_{max} \leq T$ exists such that all activities are processed within their time windows and all resource constraints are respected. This can efficiently be done by using an extension of the linear programming formulation from the previous section proposed by Brucker and Knust (2000) where additionally time windows for the activities are taken into account.

Table 3.1 Summary of disjunctive interval consistency tests

Test	$J \setminus J'$	$J \setminus J''$	Conclusion	Complexity
Input	$J \setminus \{i\}$	J	$i \rightarrow J \setminus \{i\}$	$\mathcal{O}(\lvert I\rvert \log \lvert I\rvert)$
Output	J	$J \setminus \{i\}$	$J \setminus \{i\} \rightarrow i$	$\mathcal{O}(\lvert I\rvert \log \lvert I\rvert)$
Input-or-output	$J \setminus \{i\}$	$J \setminus \{j\}$	$i \rightarrow J \setminus \{i\} \vee J \setminus \{j\} \rightarrow j$	$\mathcal{O}(\lvert I\rvert^3)$
Input negation	$\{i\}$	$J \setminus \{i\}$	$i \nrightarrow J \setminus \{i\}$	$\mathcal{O}(\lvert I\rvert \log \lvert I\rvert)$
Output negation	$J \setminus \{i\}$	$\{i\}$	$J \setminus \{i\} \nrightarrow \{i\}$	$\mathcal{O}(\lvert I\rvert \log \lvert I\rvert)$

For the LP-formulation let $\vartheta_0 < \vartheta_1 < \ldots < \vartheta^\tau$ be the ordered sequence of all different r_i- and $\overline{d}_i$-values. For $\lambda = 1, \ldots, \tau$ we consider the intervals $I^\lambda := [\vartheta_{\lambda-1}, \vartheta^\lambda]$ of length $\vartheta^\lambda - \vartheta_{\lambda-1}$. With each interval I^λ we associate a set V^λ of all activities i which can partially be scheduled in this interval, i.e. with $r_i \leq \vartheta_{\lambda-1} < \vartheta^\lambda \leq \overline{d}_i$. Let $\mathscr{A}^\lambda$ be the set of all feasible antichains consisting of activities from the set V^λ (here, again a reduction to the set of non-dominated antichains can be done) and denote again by $a^A \in \{0,1\}^n$ the incidence vector corresponding to antichain A. Furthermore, for $\lambda = 1, \ldots, \tau$ let $z_{A,\lambda}$ be a variable denoting the number of time units when antichain $A \in \mathscr{A}^\lambda$ is processed in interval I^λ. Then the preemptive feasibility problem may be written as follows:

$$\sum_{\lambda=1}^{\tau} \sum_{A \in \mathscr{A}^\lambda} a_i^A z_{A,\lambda} \geq p_i \qquad (i = 1, \ldots, n) \tag{3.9}$$

$$\sum_{A \in \mathscr{A}^\lambda} z_{A,\lambda} \leq \vartheta^\lambda - \vartheta_{\lambda-1} \qquad (\lambda = 1, \ldots, \tau) \tag{3.10}$$

$$z_{A,\lambda} \geq 0 \qquad (\lambda = 1, \ldots, \tau; A \in \mathscr{A}^\lambda) \tag{3.11}$$

Due to restrictions (3.9) all activities i are processed for at least p_i time units. Conditions (3.10) ensure that the number of time units scheduled in interval I^λ does not exceed the length $\vartheta^\lambda - \vartheta_{\lambda-1}$ of this interval.

By introducing artificial variables u^λ for $\lambda = 1, \ldots, \tau$ in conditions (3.10), the feasibility problem can be formulated as the following linear program:

$$\text{Min.} \sum_{\lambda=1}^{\tau} u^\lambda \tag{3.12}$$

$$\text{s.t.} \sum_{\lambda=1}^{\tau} \sum_{A \in \mathscr{A}^\lambda} a_i^A z_{A,\lambda} \geq p_i \qquad (i = 1, \ldots, n) \tag{3.13}$$

$$-\sum_{A \in \mathscr{A}^\lambda} z_{A,\lambda} + u^\lambda \geq -\vartheta^\lambda + \vartheta_{\lambda-1} \qquad (\lambda = 1, \ldots, \tau) \tag{3.14}$$

$$z_{A,\lambda} \geq 0 \qquad (\lambda = 1, \ldots, \tau; A \in \mathscr{A}^\lambda) \tag{3.15}$$

$$u^\lambda \geq 0 \qquad (\lambda = 1, \ldots, \tau) \tag{3.16}$$

A solution for the preemptive feasibility problem exists if and only if the linear program has the optimal solution value zero, i.e. if all values of the artificial variables become zero.

Example 3.3. Consider again Example 3.2 from the previous subsection. In Fig. 3.4 additionally the time windows $[r_i, \overline{d}_i]$ for $T = 6$ are shown.

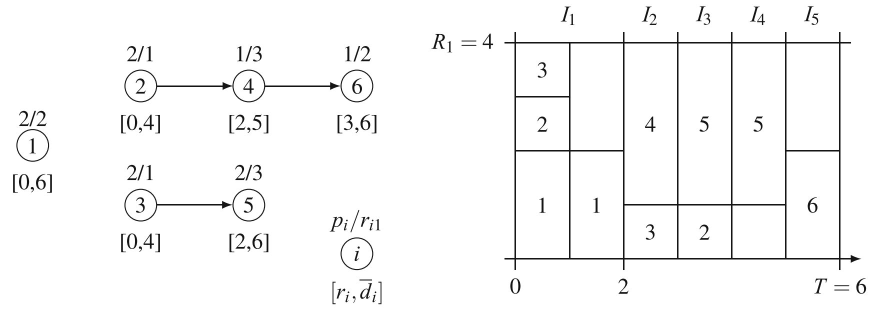


Fig. 3.4 A feasible preemptive schedule for $T = 6$, taken from Brucker and Knust (2012)

For this instance we have $\tau = 5$ intervals I^λ with the following sets V^λ:

$$\begin{aligned} I_1 &= [0,2] : V_1 = \{1,2,3\}, \\ I_2 &= [2,3] : V_2 = \{1,2,3,4,5\}, \\ I_3 &= [3,4] : V_3 = \{1,2,3,4,5,6\}, \\ I_4 &= [4,5] : V_4 = \{1,4,5,6\}, \\ I_5 &= [5,6] : V_5 = \{1,5,6\}. \end{aligned}$$

In Fig. 3.4 a feasible preemptive schedule is shown corresponding to the solution

$$z_{\{1,2,3\},1} = 1,\ z_{\{1\},1} = 1,\ z_{\{3,4\},2} = 1,\ z_{\{2,5\},3} = 1,\ z_{\{5\},4} = 1,\ z_{\{6\},5} = 1$$

which may be derived from the non-dominated solution

$$z_{\{1,2,3\},1} = 2,\ z_{\{3,4\},2} = 1,\ z_{\{2,5\},3} = 1,\ z_{\{5\},4} = 1,\ z_{\{1,6\},5} = 1$$

by eliminating parts of activities i which are processed longer than p_i time units.

Furthermore, it is easy to see that for $T = 5$ no feasible schedule respecting the time windows exists. Thus, we get a lower bound value of 6, which improves the lower bound from the previous subsection by one unit. □

Again, the linear programming formulation contains an exponential number of variables, but can be solved efficiently with column generation techniques (see Brucker and Knust 2000). In Baptiste and Demassey (2004) the LP-formulation was strengthened by adding additional valid inequalities (so-called "*cuts*"). In the following three different types of inequalities are described.

For an interval $[a, b] \subseteq [0, T]$ and an activity i we calculate a lower bound $P_i(a, b)$ for the amount of time where i has to be processed in $[a, b]$ in any feasible schedule. The value $P_i(a, b)$ is given by the minimum of

- the interval length $b - a$,
- $p_i^+ := \max\{0, p_i - \max\{0, a - r_i\}\}$, which equals the required processing time in $[a, \overline{d}_i]$ if i is started at time r_i, and

- $p_i^- := \max\{0, p_i - \max\{0, \overline{d}_i - b\}\}$, which equals the required processing time in $[r_i, b]$ if i is completed at time $\overline{d}_i$.

Obviously, the minimum of these three values is a lower bound for the processing time of activity i in the interval $[a, b] \cap [r_i, \overline{d}_i]$.

For the LP-formulation each interval $[\vartheta_{\lambda_1}, \vartheta_{\lambda_2}]$ with $0 \leq \lambda_1 < \lambda_2 \leq \tau$ may be considered. We may add the so-called "energetic cuts"

$$\sum_{\lambda=\lambda_1+1}^{\lambda_2} \sum_{A \in \mathscr{A}^\lambda} a_i^A z_{A,\lambda} \geq P_i(\vartheta_{\lambda_1}, \vartheta_{\lambda_2}) \quad (i = 1, \ldots, n;\ 0 \leq \lambda_1 < \lambda_2 \leq \tau) \tag{3.17}$$

For the formulation using all feasible antichains (and not only the non-dominated ones) some other valid inequalities may be derived from the observation that in a non-preemptive schedule (where additionally all starting times are integral), an activity cannot be processed simultaneously in two intervals $[\vartheta_{\lambda^\nu - 1}, \vartheta_{\lambda^\nu}[$ $(\nu = 1, 2)$ which have a distance of at least $p_i - 1$ time units. For example, an activity i with processing time $p_i = 5$ cannot overlap with the intervals $[1, 4[$ and $[8, 10[$ simultaneously. Furthermore, activity i can be processed for at most $\max\{4-1, 10-8\} = 3$ time units in $[1, 4[\cup[8, 10[$.

More generally, for each activity i we consider subsets of the given intervals $I^\lambda = [\vartheta_{\lambda-1}, \vartheta^\lambda[$ in which i can be processed and which have a distance of at least $p_i - 1$ time units. For activity i let $\Psi_i \subseteq \{1, \ldots, \tau\}$ be a subset of interval indices with $I^\lambda = [\vartheta_{\lambda-1}, \vartheta_t[\subseteq [r_i, \overline{d}_i]$ for all $\lambda \in \Psi_i$ and $\vartheta_{\lambda'-1} - \vartheta^\lambda \geq p_i - 1$ for all indices $\lambda < \lambda' \in \Psi_i$. Then we may state that i can be processed in at most one interval of Ψ_i, i.e. the maximal length of an interval in Ψ_i is an upper bound for the total processing time of i in all intervals belonging to Ψ_i.

Thus, we may add the "non-preemptive cuts"

$$\sum_{\lambda \in \Psi_i} \sum_{A \in \mathscr{A}^\lambda} a_i^A z_{A,\lambda} \leq \max_{\lambda \in \Psi_i} \{\vartheta^\lambda - \vartheta_{\lambda-1}\} \quad (i = 1, \ldots, n;\ \Psi_i \subseteq \{1, \ldots, \tau\}) \tag{3.18}$$

There are many possible subsets $\Psi_i \subseteq \{1, \ldots, \tau\}$ for an activity i representing non-overlapping intervals with distance at least $p_i - 1$. Baptiste and Demassey (2004) suggested to construct a set $\Psi_{i,\lambda}$ for each i and each $\lambda \in \{1, \ldots, \tau\}$ with $I^\lambda \subseteq [r_i, \overline{d}_i]$ and add all the corresponding inequalities.

Finally, the third type of inequalities tries to take into account the precedence constraints. For each activity $i = 1, \ldots, n$ we introduce an additional variable M_i representing the *midpoint* of activity i (i.e. the average of its starting and its completion time, $M_i = \frac{S_i + C_i}{2}$). Then the precedence relations $(i, j) \in E$ may be expressed as $S_j \geq C_i$, which implies $C_j - C_i \geq p_j$ and $S_j - S_i \geq p_i$. By adding the last two inequalities and dividing the result by two we get

$$M_j - M_i \geq \frac{p_j + p_i}{2} \quad \text{for all } (i, j) \in E \tag{3.19}$$

The midpoint variables M_i may be linked to the $z_{A,\lambda}$-variables by the additional inequalities

$$\sum_{\lambda=1}^{\tau}(\vartheta_{\lambda-1}+0.5) \sum_{A\in\mathscr{A}^\lambda} a_i^A z_{A,\lambda} \leq M_i\, p_i \leq \sum_{\lambda=1}^{\tau}(\vartheta^{\lambda}-0.5) \sum_{A\in\mathscr{A}^\lambda} a_i^A z_{A,\lambda} \quad (i = 1,\ldots,n)$$

If these "precedence cuts" as well as conditions (3.19) are added to the LP, additional (infeasible) solutions are excluded, i.e. the lower bound is strengthened (see Baptiste and Demassey 2004).

3.4 Conclusions

In recent years large progress has been made in calculating good lower bounds for the RCPSP (cf. Néron et al. 2006). Different concepts have been proposed and computationally tested on large sets of benchmark instances (e.g. from the PSPLIB, cf. Kolisch and Sprecher 1997). Currently, the best lower bounds can be obtained with a destructive approach combining intensive constraint propagation (cf. Baptiste and Demassey 2004) with the linear programming formulation of Brucker and Knust (2000) tightened by some cuts introduced in Baptiste and Demassey (2004). Recently, some improved lower bound values for the PSPLIB instances were reported by Schutt et al. (2011) using constraint programming.

Lower bounds for extended versions of the RCPSP can for example be found in Bianco and Caramia (2011) for the problem $PS\,|\,temp\,|\,C_{max}$ with generalized precedence relations (cf. also Chap. 5 of this handbook) or in Brucker and Knust (2003) for the multi-mode problem $MPS\,|\,temp\,|\,C_{max}$ with generalized precedence relations.

References

Artigues C, Demassey S, Néron E (2010) Resource-constrained project scheduling: models, algorithms, extensions and applications. Wiley, Hoboken

Baar T, Brucker P, Knust S (1999) Tabu search algorithms and lower bounds for the resource-constrained project scheduling problem. In: Voss S, Martello S, Osman I, Roucairol C (eds) Meta-heuristics: advances and trends in local search paradigms for optimization. Kluwer, Boston, pp 1–18

Baptiste P, Demassey S (2004) Tight LP bounds for resource constrained project scheduling. OR Spectr 26(2):251–262

Baptiste P, Le Pape C, Nuijten W (2001) Constraint-based scheduling: applying constraint programming to scheduling problems. International series in operations research & management science, vol 39. Kluwer, Boston

Bianco L, Caramia M (2011) A new lower bound for the resource-constrained project scheduling problem with generalized precedence relations. Comput Oper Res 38(1):14–20

Brucker P, Knust S (2000) A linear programming and constraint propagation-based lower bound for the RCPSP. Eur J Oper Res 127(2):355–362

Brucker P, Knust S (2003) Lower bounds for resource-constrained project scheduling problems. Eur J Oper Res 149(2):302–313

Brucker P, Knust S (2012) Complex scheduling. Springer, Berlin

Carlier J, Néron E (2000) A new LP-based lower bound for the cumulative scheduling problem. Eur J Oper Res 127(2):363–382

Damay J, Quilliot A, Sanlaville E (2007) Linear programming based algorithms for preemptive and non-preemptive RCPSP. Eur J Oper Res 182(3):1012–1022

Demeulemeester E (1992) Optimal algorithms for various classes of multiple resource constrained project scheduling problems. Ph.D. dissertation, Katholieke Universiteit Leuven, Leuven, Belgium

Dorndorf U, Huy TP, Pesch E (1999) A survey of interval capacity consistency tests for time- and resource-constrained scheduling. In: Węglarz J (ed) Project scheduling. Kluwer, Boston, pp 213–238

Klein R, Scholl A (1999) Computing lower bounds by destructive improvement: an application to resource-constrained project scheduling. Eur J Oper Res 112(2):322–346

Kolisch R, Sprecher A (1997) PSPLIB - a project scheduling problem library: OR software - ORSEP operations research software exchange program. Eur J Oper Res 96(1):205–216. http://www.om-db.wi.tum.de/psplib/

Mingozzi A, Maniezzo V, Ricciardelli S, Bianco L (1998) An exact algorithm for project scheduling with resource constraints based on a new mathematical formulation. Manage Sci 44(5):714–729

Möhring R, Schulz A, Stork F, Uetz M (2003) Solving project scheduling problems by minimum cut computations. Manage Sci 49(3):330–350

Néron E, Artigues C, Baptiste P, Carlier J, Damay J, Demassey S, Laborie P (2006) Lower bounds for resource constrained project scheduling problem. In: Perspectives in modern project scheduling. Springer, New York, pp 167–204

Schutt A, Feydy T, Stuckey P, Wallace M (2011) Explaining the cumulative propagator. Constraints 16(3):250–282

Stinson J, Davis E, Khumawala B (1978) Multiple resource-constrained scheduling using branch and bound. AIIE Trans 10(3):252–259

Chapter 4
Metaheuristic Methods

Anurag Agarwal, Selcuk Colak, and Selcuk Erenguc

Abstract Given the $\mathcal{NP}$-hard nature of the Resource Constrained Project Scheduling Problem (RCPSP), obtaining an optimal solution for larger instances of the problem becomes computationally intractable. Metaheuristic approaches are therefore commonly used to provide near-optimal solutions for larger instances of the problem. Over the past two decades, a number of different metaheuristic approaches have been proposed and developed for combinatorial optimization problems in general and for the RCPSP in particular. In this chapter, we review the various metaheuristic approaches such as genetic algorithms, simulated annealing, tabu search, scatter search, ant colonies, the bees algorithm, neural networks etc., that have been applied to the RCPSP. One metaheuristic approach called the NeuroGenetic approach is described in more detail. The NeuroGenetic approach is a hybrid of a neural-network based approach and the genetic algorithms approach. We summarize the best results in the literature for the various metaheuristic approaches on the standard benchmark problems J30, J60, J90, and J120 from PSPLIB (Kolisch and Sprecher, Eur J Oper Res 96:205–216, 1996).

Keywords Makespan minimization • Metaheuristics • NeuroGenetic approach • Project scheduling • Resource constraints

A. Agarwal (✉)
Department of Information Systems and Decision Sciences, University of South Florida,
Sarasota, FL, USA
e-mail: agarwala@usf.edu

S. Colak
Department of Business, Cukurova University, Adana, Turkey
e-mail: scolak@cu.edu.tr

S. Erenguc
Department of Information Systems and Operations Management, University of Florida,
Gainesville, FL, USA
e-mail: selcuk.erenguc@warrington.ufl.edu

C. Schwindt, J. Zimmermann (eds.), *Handbook on Project Management and Scheduling Vol. 1*, International Handbooks on Information Systems,
DOI 10.1007/978-3-319-05443-8_4

4.1 Introduction

Due to the combinatorial nature of the Resource Constrained Project Scheduling Problem (RCPSP), obtaining optimal solutions using exact methods, for problems with over 50 or so activities, becomes intractable and hence impractical. Metaheuristic approaches such as genetic algorithms, simulated annealing, tabu search, scatter search etc., are therefore used to provide near-optimal solutions within reasonable computation times. Over the last two decades, a number of metaheuristic approaches have been proposed and applied to various optimization problems in general and to the RCPSP in particular. In this chapter, we will review and discuss these metaheuristic approaches.

In the standard RCPSP problem, the one that is most commonly studied and for which standard benchmark problems exist, the objective is to minimize the makespan. Other characteristics of the problem include: (1) preemption of activities is not allowed, (2) the resources are renewable from period to period, (3) there is a single mode of resource usage, (4) the quantities of available resources are known and fixed for the duration of the project, and (5) resource requirements and processing times for each activity are known and fixed a priori. Benchmark problems for this standard RCPSP problem exist in PSPLIB (Kolisch and Sprecher 1996). Many researchers have demonstrated the effectiveness of their approaches on these benchmark problems.

The rest of the chapter is organized as follows. Before discussing the various metaheuristics, for the sake of completeness, we will briefly review, in Sect. 4.2, some important concepts about heuristic methods in general. These concepts are critical to the understanding of metaheuristics. In Sect. 4.3, we will review the various metaheuristic approaches in the literature, which have been applied to the RCPSP. In Sect. 4.4 we will describe in some detail, one metaheuristic called the NeuroGenetic approach. Computational results are presented in Sect. 4.5 and the conclusions are discussed in Sect. 4.6.

4.2 Single-Pass Heuristics Methods

A variety of single-pass heuristics based on different priority rules, such as, "minimum latest-finish-time next" (min LFT) or "longest processing time first" (LPT) etc. have been proposed in the literature for the RCPSP. The priority rule determines a unique activity list. An activity list, for a given problem, is basically a list of all the activities of that problem listed in a certain precedence feasible order. The order is determined by the priority rule. Different priority rules produce different activity lists. Given an activity list, a schedule can be generated using a schedule-generation scheme. While single-pass heuristics are extremely fast with $\mathscr{O}(n \cdot \ln(n))$ complexity, the optimality gaps are considered quite unsatisfactory.

To further reduce the optimality gaps, several improvement techniques have been proposed in the literature. These improvement techniques include (1) applying different schedule-generation schemes (Serial and Parallel), (2) using forward and backward scheduling, and (3) applying double justification schemes. These improvement methods supplement single-pass heuristic approaches to reduce the optimality gaps without significant extra computational time. We will briefly review these improvement methods next.

4.2.1 Serial vs. Parallel Schedule-Generation Scheme

Given an activity list ℓ, we can use either the serial schedule-generation scheme (S-SGS) or the parallel schedule-generation scheme (P-SGS) to generate a schedule. In S-SGS, activities in ℓ are considered one by one in serial order, and scheduled at the earliest clock time at which that activity becomes precedence and resource feasible. In P-SGS, a clock is maintained and at each unit of clock time, the set of activities that is precedence and resource feasible is determined. If this set contains only one activity, that activity is scheduled. If the set contains more than one activity, then priority is given to the activity that appears first in the activity list. These two schedule-generation schemes often give different schedules. We can apply both these schemes to a problem and use the schedule with the shorter makespan. For more details, please refer to Kolisch (1996) or Chap. 1 of this handbook.

4.2.2 Forward and Backward Scheduling

An RCPSP graph $G = (V, E)$, where V represents nodes and E represents arcs or precedence constraints, can be viewed either as a forward problem or a backward problem. In the backward problem, the last activity node is regarded as the first activity node and vice versa. Essentially, the entire activity graph is viewed backwards. Using the same schedule-generation scheme and the same priority rule, the forward and backward schedules for the same problem often give different makespans. The schedule with the shorter makespan can be used as the best solution. This method was first proposed by Li and Willis (1992). For the same activity list, applying forward and backward scheduling, in conjunction with serial and parallel schedule-generation schemes can produce as many as four different solutions and we can use the best of these four as our final solution.

4.2.3 Double Justification Scheme

Once a schedule has been generated using some schedule-generation scheme and some priority rule, it is often possible to reduce the empty spaces on the Gantt chart by scanning the Gantt chart in the reverse direction and shifting the activities to the right to fill the empty spaces. This shifting of activities packs the Gantt chart more densely, thus reducing the makespan. This procedure of shifting activities to the right is called backward justification. Once we perform the backward justification process, we can then perform a forward justification in which activities are scanned in the forward direction and activities shifted to the left to fill the empty spaces on the new Gantt chart. Applying both these procedures is called forward–backward improvement (FBI) or double justification (Valls et al. 2005). This approach can be applied to refine any solution obtained through any schedule-generation scheme.

Single-Pass heuristics in conjunction with the improvement techniques discussed above provide very good solutions. However, the optimality gaps are still unsatisfactory. By using metaheuristics, the optimality gap can be reduced further, although at the expense of more computation time. In the next section we will discuss the various metaheuristic approaches.

4.3 Metaheuristic Methods

Over the past 20 years, several metaheuristic approaches such as genetic algorithms, simulated annealing, tabu search etc., have become very popular for solving combinatorial optimization problems in general and the RCPSP in particular. While a single-pass heuristic generates solutions in a deterministic manner, a metaheuristic generates multiple solutions in a somewhat random manner within the problem's search space in the hopes of finding the optimal or a near-optimal solution. Different types of metaheuristics employ different search strategies to produce new solutions, but they all strive to find increasingly better solutions as the search progresses. At some point the search must stop, using some stopping criteria, in the interest of computation time. The stopping criteria can either be in terms of a predetermined CPU time or in terms of the number of solutions evaluated or in terms of achieving a certain minimum gap from a known lower bound solution (for a minimization problem).

Many of the metaheuristic approaches have been inspired by processes observed in nature. For example the genetic algorithms were inspired by the process of evolution of species. The simulated annealing approach was inspired by the annealing process of metals and the ant colony optimization approach was inspired by observing the foraging behavior of colonies of ants. Before we discuss each of the metaheuristic approaches we will briefly discuss the concept of global vs. local search, also sometimes referred to as diversification and intensification.

Every search method faces two main challenges. The first challenge is—how to converge the search to the best solution in a given neighborhood in the search space (local search). The second challenge is—how to direct the search towards the best neighborhood amongst many neighborhoods in the overall search space (global search). A technique that effectively addresses both these challenges is likely to be more successful in generating good solutions. Unfortunately, search strategies that help with local search are often in conflict with those that help with global search, and vice versa. Metaheuristics have to therefore device strategies to balance these two conflicting goals. In general, strategies devised for global search are termed diversification strategies, whereas those designed for local search are termed intensification strategies.

All metaheuristic approaches must address how these conflicting goals are balanced. Sometimes, a hybrid of two metaheuristics is employed if one metaheuristic happens to be intrinsically better at local search, such as simulated annealing, and the other intrinsically better at global search, such as genetic algorithms. One of the challenges in global search is how to avoid getting stuck in a non-optimal neighborhood. Strategies should be developed to sense the state of being stuck in a local neighborhood and to have an escape plan. So essentially, all metaheuristics must have elements of local search, global search, and a strategy to avoid getting stuck in one neighborhood. We will now briefly describe each of the metaheuristic approaches. Detailed explanation can be found in the cited references.

We list below, the various metaheuristic approaches that, to the best of our knowledge, have been applied to the RCPSP:

1. Genetic Algorithms
2. Scatter Search
3. Electromagnetism-Like Search
4. Shuffled Frog Leaping Algorithm
5. Simulated Annealing
6. Tabu Search
7. Ant-Colony Optimization
8. Bees Algorithm
9. Multi-Pass Sampling
10. Filter and Fan
11. Neural-Networks based search
12. The Neuro Genetic Approach

We next provide a brief description of each of these approaches with references to articles using these approaches.

4.3.1 Genetic Algorithms

The Genetic algorithms (GAs) approach, proposed by Goldberg (1989), is by far the most popular metaheuristic approach for optimization problems in general and the RCPSP in particular. As previously stated, GAs are inspired by the phenomenon of evolution of species observed in nature. In the evolution process, successive generations of populations of a species attempt to improve upon their previous generations through certain genetic and survival-of-the-fittest processes. For this reason, GAs are also called population-based metaheuristics. When applied to an optimization problem, GAs employ similar processes to produce improved sets of solutions (generations of population) as the search progresses from one generation to the next.

When applying GAs to the RCPSP, a set of activity lists acts as the population. From a given population of activity lists, a new population is produced through the reproduction process, involving an appropriate crossover mechanism. The survival-of-the-fittest process is employed by being selective about the choice of parent activity lists used for producing a new offspring for the next generation of population. An activity list, when used for genetic algorithms, is also called a chromosome in which each activity is considered a gene. For a given chromosome (or activity list), the best possible schedule can be generated using the S-SGS or the P-SGS, forward or backward scheduling, and FBI. In GAs, diversification is achieved through crossover mechanisms that involve two or more crossover points. Intensification is generally achieved using some local search strategy that is not necessarily population based. If the search gets stuck in a local neighborhood, the process of mutation is applied to escape from the neighborhood. In mutation, the position of one gene is randomly changed, as long as it maintains its precedence feasible state.

Within the umbrella of Genetic Algorithm procedures, many variations are possible depending on the strategies used to produce new populations. Several mechanisms affect the creation of new populations, including (1) the choice of parent chromosomes, (2) the crossover mechanism, and (3) mutation strategies. The GA approach has been found to be particularly effective for global search although somewhat weak for local search. Researchers have tried to supplement GAs with some non-GA search strategies such as path relinking to fine tune solutions obtained by GAs. Genetic algorithms have been applied by Hartmann (1998, 2002), Valls et al. (2004, 2008), Alcaraz and Maroto (2001), Debels and Vanhoucke (2007), Gonçalves et al. (2011), Khanzadi et al. (2011), Sebt et al. (2012), and Zamani (2013). Amongst all the metaheuristics, GAs have produced some of the best results in the literature for the RCPSP. One of the variations of Genetic Algorithms is Scatter Search which we discuss next.

4.3.2 Scatter Search

Scatter Search (SS) can be considered a variation of Genetic Algorithms, in that they both work with populations of chromosomes. In SS, the main difference is that there is heavy emphasis on diversification. In SS, in addition to the pool of chromosomes used in each generation, a reference set of chromosomes (*RefSet*) is also maintained. The *RefSet* is further divided into two subsets. The first subset (*RefSet1*) is a collection of high-quality solutions and the second subset (*RefSet2*) is a collection of diverse solutions. A new pool of solutions is generated by crossing pairs of chromosomes in *RefSet1* and also by crossing a chromosome from *RefSet1* by a chromosome in *RefSet2*.

This approach is also called a two-tier design, which is maintained throughout the search process. To ensure diversity, the diverse solutions must be sufficiently distant from the high-quality solutions. A good SS approach must specify (1) the diversification generation method to generate the initial pool of chromosomes, (2) the improvement method, which is usually the double justification method, (3) the reference set update method to ensure that in the new generations of chromosomes, adequate diversification is maintained, and (4) the solution combination method to generate new solutions using appropriate crossover techniques. Debels et al. (2006), Ranjbar et al. (2009), and Mobini et al. (2009) have applied the scatter search approach for the RCPSP. Scatter search metaheuristic has also produced some of the best results in the RCPSP literature.

4.3.3 Electromagnetism-Like Search

The Electromagnetism-Like (EM) Search can be thought of as another variation of Genetic Algorithms because it also works with populations of chromosomes. The EM Search strategy comes into play when generating new populations of chromosomes. EM Search is inspired by Coulomb's *Inverse Square Law*, found in Electromagnetism theory, which describes the electrostatic interaction between electrically charged particles. The force between charged particles is inversely proportional to the square of the distance between them. Birbil and Fang (2003) first introduced the idea of EM Search applied to global optimization. Debels et al. (2006) applied the EM search approach to the RCPSP, along with Scatter Search. When applied to the RCPSP, each chromosome is regarded as a particle with a certain charge, where the charge is a function of the objective function value. Each particle exerts a certain force, either attraction or repulsion, with other particles. The principle behind the algorithm is that inferior particles (or activity lists) will repel or prevent a move in their direction and superior particles will attract or facilitate move in their direction. This strategy ensures that subsequent generations improve upon previous generations. For details of how the charges are defined and how the forces are calculated, please refer to Debels et al. (2006).

4.3.4 Shuffled Frog Leaping Algorithm

The Shuffled Frog Leaping Algorithm (SFLA) can also be regarded as a variation of Genetic Algorithms, as it also works with populations of activity lists or chromosomes. The algorithm was first proposed by Eusuff et al. (2006) and first applied to the RCPSP by Fang and Wang (2012). In the SFLA, an initial population of chromosomes of activity lists is formed by a set of randomly generated solutions called virtual frogs. Fang and Wang generate the initial population using the regret-based biased random sample method based on the min LFT priority rule. Then double justification improvement is applied to each solution in the population. The chromosomes are then sorted in descending order of the objective function value. The virtual frog with the best objective function value in the entire population is denoted by P_G. The population is then partitioned into λ subsets of frogs called *memeplexes*. Each *memeplex*, containing μ frogs, is considered as having its own culture and evolves independently. Each *memeplex* is further partitioned into several *submemeplexes*, each with ν virtual frogs. The resource-based crossover is applied to the best frog (P_B) and the worst frog (P_W) in a *submemeplex* to produce a child. If the child is worse than P_W, then P_W and P_G are crossed over. If the new child is still worse than P_W, a random frog is generated to replace P_W. For the new child, permutation based local search (PBLS) and double justification improvements are applied. After a certain number of iterations of crossover and local search, the whole population is shuffled and partitioned again to perform the next generation of evolution. For further details, please refer to Fang and Wang (2012).

4.3.5 Simulated Annealing

The Simulated Annealing (SA) metaheuristic has been inspired by the process of annealing used in metallurgy for hardening metals. In the SA algorithm, a current solution is maintained at all times. Neighborhood solutions from the current solution are evaluated iteratively and if a better solution is found, it becomes the new current solution. Occasionally, with a certain acceptance probability, a worse solution replaces the current solution as a mechanism to escape from a local neighborhood. If the probability $\pi = \mathrm{e}^{(-\Delta/Temp)}$ is less than a random number $u \in [0, 1]$, a worse solution is accepted. Here, Δ is the difference in the objective function value of the current solution and the new solution. So, the smaller this difference, the higher the acceptance probability and vice versa. *Temp* is a temperature parameter. So, the higher the value of *Temp*, the higher is the acceptance probability and vice versa. A cooling schedule for *Temp* is generally deployed in which you start with a high value and progressively lower the value as the search progresses. A higher value of *Temp* encourages diversification because it increases the probability of accepting a worse solution, which in turn allows search in other neighborhoods, while a lower *Temp* value encourages intensification. Note that if we never accept a worse solution, we will necessarily stay in the same neighborhood. A good neighbor-generation-scheme must be specified for a SA algorithm, one that allows improvements within the local neighborhood, yet allows opportunities to escape a local neighborhood to evaluate diversified neighborhoods.

The SA metaheuristic is stronger for local search than for a global search. Boctor (1996), Cho and Kim (1997), Bouleimen and Lecocq (2003), and Bouffard and Ferland (2006) have applied simulated annealing to the RCPSP.

4.3.6 *Tabu Search*

Tabu Search (TS) employs intelligent use of memory to help exploit useful past experience in search. Memory is essentially a list of previously visited solutions. Several types of lists are maintained, each for a different purpose. A short-term tabu list includes recently visited solutions. Its purpose is to help avoid cycling within the same neighborhood. If a new solution is in this short-term list, it implies that the current neighborhood should not be explored further and the search should diversify to another neighborhood by using a different starting point. A list of poor solutions is also maintained so that if the search leads to a neighborhood of poor solutions, its presence can be detected and the search directed away from the current neighborhood—as if it is taboo to be found in some neighborhoods. A list of good quality solutions is also maintained to help identify good neighborhoods, in which search can be intensified. Neighborhood search is performed similar to the neighborhood search in SA using a neighbor-generation-scheme. Appropriate diversification and intensification strategies are devised to guide the search. Tabu search based metaheuristics for the RCPSP are proposed by Pinson et al. (1994), Baar et al. (1997), Thomas and Salhi (1998), and Nonobe and Ibaraki (2002).

4.3.7 *Ant Colony Optimization*

Ant colony optimization (ACO) metaheuristic is inspired by the observed foraging behavior of ant colonies in which ants discharge a chemical substance called pheromone along the path between its colony and the food source. The smell of pheromone signals to the other ants in the colony about the existence of previously followed paths. A stronger smell signals that a larger number of ants have been on that path more recently, suggesting that food might be found on that path. After some elapsed time, all ants in a colony figure out the shortest path towards food and they travel on the shortest possible path, thus optimizing their collective efforts. When applied to the RCPSP, an ant selects the activity order in the activity list. The ant uses heuristic information (η_{ij}) and pheromone information (τ_{ij}) as indicators of goodness of placing an activity at a particular position in the activity list. The idea behind using heuristic information is similar to the idea of using a priority rule. The pheromone information stems from former ants that have found good solutions. Some function of η_{ij} and τ_{ij} determines the probability of choosing the next activity in the activity list from amongst the set of eligible activities. Merkle et al. (2002) proposed an ant-colony approach to the RCPSP.

Escaping from a neighborhood is achieved by accepting worse solutions after a predetermined number of searches in the current neighborhood. A suggested intensification strategy by Merkel et al. (2002) is a 2-OPT move. The 2-OPT move has nothing to do with ACO though, and can be applied in conjunction with any metaheuristic. Tseng and Chen (2006) use a hybrid of ACO and Genetic algorithms and call their algorithm ANGEL.

4.3.8 Bees Algorithm/Artificial Bee Colony/Bee Swarm Optimization

The Bees Algorithm (BA), also known as Artificial Bee Colony (ABC) approach or Bee Swarm Optimization (BSO) is inspired by the observed foraging behavior of bees. The foraging behavior of bees happens to be quite different from the foraging behavior of ants, hence the BA is quite different from the ACO. Foraging in a bee colony begins by sending a few scout bees to search for good flower patches. These scout bees search various flower patches randomly and return to the hive and convey the information about good flower patches through a "waggle dance". They convey the distance, the direction, and the quality rating of a flower patch through their dance. After the dance, some follower bees follow the scout bees who reported the highest quality flower patches. This mechanism allows the bee colony to efficiently gather food. Inspired by this behavior of the bees, in the Bees Algorithm, an initial set of random solutions in the search space serves as scout bees. Each solution in the initial set is evaluated for its quality. Bees reporting the highest quality are chosen as "selected bees" and the neighborhood chosen by the selected bees become targets for local neighborhood search. The scouting mechanism becomes more focused on good neighborhoods for intensification. Sadeghi et al. (2011) and Ziarati et al. (2011) report results on the RCPSP problem, using the Bees Algorithm. Jia and Seo (2013) also use the Bees Algorithm for the RCPSP.

4.3.9 Multi-Pass Sampling

The term "sampling" in the multi-pass sampling approach simply means that a sample of all possible schedules is evaluated. In that sense, all metaheuristic approaches are essentially multi-pass sampling approaches. But in the multi-pass sampling approach as applied to the RCSPS, (Kolisch and Drexl 1996), a sample of solutions is generated and evaluated using certain random strategies designed to bias the solutions towards better solutions. Kolisch and Drexl (1996) talk about three types of sampling strategies—(1) random sampling, (2) biased random sampling, and (3) parameterized regret-based biased sampling. A different mapping function is used for each type of sampling strategy that determines the probability of an activity being selected next in the activity list. Before the advent of the more advanced metaheuristic approaches, multi-pass sampling approaches gave some of the best results for the RCPSP.

4.3.10 Filter and Fan

The Filter and Fan (F&F) search is more of a tree search than a metaheuristic, although it has elements of randomness that is characteristic of metaheuristics. It builds a search tree where the root node is the solution obtained by an iterative local search approach starting from a random schedule. The first level of the tree is constructed using certain moves that were captured during the descent process in the local search. The next level of the tree is similarly determined from the local search. The method stops branching as soon as an improved solution is found or the maximum number of levels is reached, or if no more candidate moves remain to be evaluated. In case a global improvement is found during the tree search, the new best solution is made the starting solution for another run of the local search procedure. For details, please refer to Ranjbar (2008).

4.3.11 Neural-Networks Based Search

The Neural-Networks based (NN-based) search was first proposed for a scheduling problem by Agarwal et al. (2003) and was first applied to the RCPSP by Colak et al. (2006). In this approach, a chosen priority rule (such as min LFT or LPT) and a chosen SGS, i.e., serial or parallel, is applied iteratively. The best solution, after a certain number of iterations, is saved as the final solution. In each iteration, the activity list ℓ is different. Since only one priority rule is used, the question is how is a new activity list generated? In this approach, for a set of activities $V = (0, 1, \ldots, n+1)$ we define a weight vector $W = (w_0, w_1, \ldots, w_{n+1})$. Suppose Q represents the vector of parameters used in the chosen priority rule. For example, if the chosen priority rule is say Min LFT, then let q_i represent the LFT for activity i and $Q = (q_0, q_1, \ldots, q_{n+1})$ represents the vector of latest finish times of all activities. If the chosen heuristic is say LPT, then q_i in Q represents the processing time for activity i. Let Q_w represent a vector of weighted parameters $(w_0 \cdot q_0, w_1 \cdot q_1, \ldots, w_{n+1} \cdot q_{n+1})$. If we assume a unit weight vector W, then $Q_w = Q$. For the first iteration we obtain ℓ using Q. For subsequent iterations, we use Q_w to obtain a different ℓ. After each iteration, W is updated using a weight update strategy to give a new Q_w, which in turn generates a new ℓ, which produces a new solution.

This NN-based approach is basically a local search approach because the perturbed vector Q_w produces a perturbed activity list in the local neighborhood of the original activity list. The approach is called NN-based because of its similarity with the traditional neural networks in which a weight vector is used as the perturbation mechanism. If a good priority rule and a good SGS are used to produce the initial solution, the local search around this original solution produces very competitive results as shown in Colak et al. (2006). In the next section, we will describe the NeuroGenetic approach in some detail.

4.4 The NeuroGenetic Approach

In recent years, there has been a trend towards hybrid metaheuristics. The NeuroGenetic (NG) approach is one such hybrid approach. It's a hybrid of the neural-networks based approach and the genetic algorithms approach. The GA approach has shown remarkable success in solving the RCPSP and is one of the most preferred approaches for this problem. Colak et al. (2006) proposed a NN-based approach which also gave very competitive results for this problem. Although both GA and NN-based approaches give some of the best known results in the literature, the two approaches are very different from each other in terms of search strategies. While the GA approach is very effective for global search, the NN-based approach is basically a nondeterministic local-search technique. In the NG approach, GAs provide the diversification in search while NNs provide the intensification.

It may be noted that while GA is a solution-space based approach, NN-based approach is a problem-space based approach. In a solution-space based approach, the solution is perturbed from one iteration to the next, using some mechanism (such as crossover and mutation in Genetic Algorithms). Tabu Search, Simulated Annealing, GA, and Ant-Colony Optimization all belong to the class of solution-space based approaches. In a problem-space approach, the problem parameters are perturbed before moving to the next iteration, while using the same heuristic. The NN-based approach provides a framework for applying a problem-space based approach. With the help of a weight vector which is modified after each iteration, weighted problem parameters are used instead of the original parameters. A suitable weight modification strategy guides the search. The makespan for the new schedule is still calculated using the original problem parameters. The NG approach may also be regarded as a hybrid of a solution-space based approach and a problem-space based approach.

In the NG approach, the GA iterations and NN iterations are interleaved, i.e., the search alternates between GA iterations and NN iterations. When switching from the GA to the NN approach, a small set of good quality solutions obtained thus far by the GA approach are fed to the NN approach, which tries to improve upon those solutions locally (intensification). When returning to the GA approach, a set of good solutions obtained thus far by the NN approach becomes part of the current GA population. Interleaving requires that switching back and forth between the two techniques be technically feasible. Switching between NN and GA approaches is somewhat challenging, given that the two approaches work quite differently in terms of problem encoding. For more details on how the switching is done, please see Agarwal et al. (2011).

4.5 Computational Results

In this section, we will summarize the best results in the literature for various metaheuristics discussed in this chapter. We discuss the results for the NeuroGenetic approach in some detail. We will present the results of running the NN approach and the GA approach individually and then of running the NeuroGenetic approach. These techniques were run on the well-known benchmark problem instance sets from PSPLIB (Kolisch and Sprecher 1996) and http://www.om-db.wi.tum.de/psplib/main.html. The sets J30, J60, and J90 consist of 480 problem instances with

Table 4.1 Average percentage deviations from optimal solutions for J30 and from critical-path based lower bound for J60, J90, and J120 for NN, GA, and Neurogenetic Approaches

		Number of schedules evaluated	
Approach	Dataset	1,000	5,000
NN approach alone	J30	0.25	0.11
GA	J30	0.19	0.15
Neurogenetic	J30	0.13	0.10
NN approach alone	J60	11.72	11.39
GA	J60	11.66	11.52
Neurogenetic	J60	11.51	11.29
NN approach alone	J90	11.21	11.10
GA	J90	11.31	11.11
Neurogenetic	J90	11.17	11.06
NN approach alone	J120	34.94	34.57
GA	J120	35.11	34.95
Neurogenetic	J120	34.65	34.15

4 resource types and 30, 60, and 90 activities, respectively. The set J120 consists of 600 problem instances with 4 resource type and 120 activities. For a good review of results we refer the readers to Icmeli et al. (1993), Ozdamar and Ulusoy (1995), Demeulemeester and Herroelen (1997), Herroelen et al. (1998), Herroelen et al. (1998), Hartmann and Kolisch (2000), Kolisch and Hartmann (2006).

Table 4.1 shows the results of NN, GA, and NG approaches for 1,000 and 5,000 solutions for each of the four datasets in terms of the average percentage deviations from optimal solutions for J30 and from critical-path based lower bound solutions for J60, J90, and J120. For each dataset, NG performs better than NN or GA alone. Tables 4.2 through 4.5 show the results obtained by some of the top performing metaheuristics in the literature for 1,000 and 5,000 schedules, respectively. Again, for the J30 problems in Table 4.2, the average percentage deviations from optimal solutions are shown. For J60, J90, and J120 problems (Tables 4.3, 4.4, and 4.5), the average percentage deviations from the critical-path based lower bound are presented.

4.6 Conclusions

Over the past 20 years, a large number of metaheuristic approaches have been proposed for the resource constrained project scheduling problem. Many of the metaheuristics have been inspired by observation of certain processes found in nature. These processes include the genetic processes for evolution of species, ant colony's food foraging behavior, bee colony food foraging behavior, electromagnetism, behavior of neurons in a neural networks etc. The various metaheuristics

Table 4.2 Average percentage deviations from the optimal solutions for J30

Algorithm	Reference	# of schedules	
		1,000	5,000
Scatter search	Mobini et al. (2009)	0.05	0.02
Filter and fan	Ranjbar (2008)	0.09	0.00
Hybrid scatter search	Ranjbar et al. (2009)	0.10	0.03
GA, TS, path relinking	Kochetov and Stolyar (2003)	0.10	0.04
Decomposition based GA	Debels and Vanhoucke (2007)	0.12	0.04
Neurogenetic	Agarwal et al. (2011)	0.13	0.10
Bees algorithm	Sadeghi et al. (2011)	0.15	0.09
ACO and GA (ANGEL)	Tseng and Chen (2006)	0.22	0.09
Sampling—LFT	Tormos and Lova (2003)	0.23	0.14
GA	Alcaraz et al. (2004)	0.25	0.06
HNA	Colak et al. (2006)	0.25	0.11
Sampling—LFT	Tormos and Lova (2001)	0.25	0.15
GA—hybrid	Valls et al. (2008)	0.27	0.06
Scatter search/EM	Debels et al. (2006)	0. 27	0.11
GA (biased random key)	Gonçalves et al. (2011)	0.32	0.02
GA	Alcaraz and Maroto (2001)	0.33	0.12
Bees algorithm	Jia and Seo (2013)	0.34	0.17
GA	Valls et al. (2005)	0.34	0.20

Table 4.3 Avg percentage deviations from critical path based lower bounds for J60

Algorithm	Reference	# of schedules	
		1,000	5,000
GA	Khanzadi et al. (2011)	10.10	9.54
Filter and fan	Ranjbar (2008)	10.66	10.56
SS, path relinking	Mobini et al. (2009)	11.12	10.74
Decomposition based GA	Debels and Vanhoucke (2007)	11.31	10.95
GA	Zamani (2013)	11.33	10.94
Shuffled frog-leaping	Fang and Wang (2012)	11.44	10.87
Neurogenetic	Agarwal et al. (2011)	11.51	11.29
GA (biased random key)	Gonçalves et al. (2011)	11.56	10.57
GA—hybrid	Valls et al. (2008)	11.56	11.10
Hybrid scatter search	Ranjbar et al. (2009)	11.59	11.07
HNA	Colak et al. (2006)	11.72	11.39
Scatter search	Debels et al. (2006)	11.73	11.10
GA	Alcaraz et al. (2004)	11.89	11.19
Bees algorithm	Sadeghi et al. (2011)	11.93	11.48
ACO and GA (ANGEL)	Tseng and Chen (2006)	11.94	11.27
Sampling—LFT	Tormos and Lova (2003)	12.04	11.72
GA	Valls et al. (2005)	12.21	11.27
Bees algorithm	Jia and Seo (2013)	12.35	11.96

Table 4.4 Avg percentage deviations from critical path based lower bounds for J90

		# of schedules	
Algorithm	Reference	1,000	5,000
GA—hybrid	Valls et al. (2008)	NA	10.46
Filter and fan	Ranjbar (2008)	10.52	10.11
Decomposition based GA	Debels and Vanhoucke (2007)	10.80	10.35
GA	Khanzadi et al. (2011)	11.02	10.75
Neurogenetic	Agarwal et al. (2011)	11.17	11.06
Scatter search/EM	Debels et al. (2006)	11.30	10.59

Table 4.5 Avg percentage deviations from critical path based lower bounds for J120

		# of schedules	
Algorithm	Reference	1,000	5,000
Filter and fan	Ranjbar (2008)	32.96	31.42
Decomposition based GA	Debels and Vanhoucke (2007)	33.55	32.18
GA	Zamani (2013)	34.02	32.89
GA—hybrid	Valls et al. (2008)	34.07	32.54
Scatter search, path relinking	Mobini et al. (2009)	34.49	32.61
Neurogenetic	Agarwal et al. (2011)	34.65	34.15
Shuffled frog leaping	Fang and Wang (2012)	34.83	33.20
HNA—FBI	Colak et al. (2006)	34.94	34.57
Scatter search	Debels et al. (2006)	35.22	33.10
GA—FBI	Valls et al. (2005)	35.39	33.24
GA (biased random key)	Gonçalves et al. (2011)	35.94	32.76
Sampling—LFT	Tormos and Lova (2003)	35.98	35.30
Sampling—LFT	Tormos and Lova (2001)	36.32	35.62
ACO and GA hybrid (ANGEL)	Tseng and Chen (2006)	36.39	34.49
GA	Alcaraz et al. (2004)	36.53	33.91
Bees algorithm	Jia and Seo (2013)	36.84	35.79

include Genetic Algorithms, Neural Networks, Ant Colony Optimization, Bees Algorithm, Simulated Annealing, Shuffled Frog-Leaping Algorithm, Scatter Search, Electromagnetism-Like Algorithms, Tabu Search, and Filter and Fan approach.

In this chapter we have reviewed and discussed the various metaheuristic approaches used for the RCPSP and explained in some detail one metaheuristic approach called the NeuroGenetic approach. The top performing metaheuristics are mostly GA based approaches. Metaheuristics that are not GA based, that perform well are Filter and Fan, Neural Networks based metaheuristics, Multi-pass Sampling, and Bee Colony Algorithms. Simulated Annealing, Tabu Search, and Ant Colony Optimization have not performed quite as well in comparison to other metaheuristics.

References

Agarwal A, Jacob VS, Pirkul H (2003) Augmented neural networks for task scheduling. Eur J Oper Res 151(3):481–502

Agarwal A, Colak S, Erenguc SS (2011) A neurogenetic approach for the resource-constrained project scheduling problem. Comput Oper Res 38(1):44–50

Alcaraz J, Maroto C (2001) A robust genetic algorithm for resource allocation in project scheduling. Ann Oper Res 102:83–109

Alcaraz J, Maroto C, Ruiz R (2004) Improving the performance of genetic algorithms for the RCPS problem. In: Proceedings of the ninth international workshop on project management and scheduling, Nancy, pp 40–43

Baar T, Brucker P, Knust S (1997) Tabu search algorithms for resource-constrained project scheduling problems. In: Voss S, Martello S, Osman I, Roucairol C (eds) Metaheuristics: advances and trends in local search paradigms for optimization. Kluwer, Boston, pp 1–18

Birbil SI, Fang SC (2003) An electromagnetism-like mechanism for global optimization. J Global Optim 25:263–282

Boctor FF (1996) Resource-constrained project scheduling simulated annealing. Int J Prod Res 34(8):2335–2351

Bouffard V, Ferland JA (2007) Improving simulated annealing with variable neighborhood search to solve the resource-constrained scheduling problem. J Sched 10:375–386

Bouleimen K, Lecocq H (2003) A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. Eur J Oper Res 149:268–281

Cho JH, Kim YD (1997) A simulated annealing algorithm for resource-constrained project scheduling problems. J Oper Res 48(7):736–744

Colak S, Agarwal A, Erenguc SS (2006) Resource-constrained project scheduling problem: a hybrid neural approach. In: Węglarz J, Jozefowska J (eds) Perspectives in modern project scheduling. Springer, New York, pp 297–318

Debels D, Vanhoucke M (2007) A decomposition-based genetic algorithm for the resource-constrained project-scheduling problem. Oper Res 55(3):457–469

Debels D, De Reyck B, Leus R, Vanhoucke M (2006) A hybrid scatter search/electromagnetism meta-heuristic for project scheduling. Eur J Oper Res 169(2):638–653

Demeulemeester E, Herroelen W (1997) New benchmark results for the resource-constrained project scheduling problem. Manag Sci 43(11):1485–1492

Eusuff M, Lansey K, Fasha F (2006) Shuffled frog-leaping algorithm: a memetic metaheuristic for discrete optimization. Eng Optim 38(2):129–154

Fang C, Wang L (2012) An effective shuffled frog-leaping algorithm for resource-constrained project scheduling problem. Comput Oper Res 39:890–901

Goldberg DE (1989) Genetic algorithms in search optimization and machine learning. Addison Wesley, New York

Gonçalves JF, Resende MGC, Mendes JJM (2011) A biased random-key genetic algorithm with forward-backward improvement for the resource constrained project scheduling problem. J Heuristics 17:467–486

Hartmann S (1998) A competitive genetic algorithm for the resource-constrained project scheduling. Nav Res Log 45:733–750

Hartmann S (2002) A self-adapting genetic algorithm for project scheduling under resource constraints. Nav Res Log 49(5):433–448

Hartmann S, Kolisch R (2000) Experimental evaluation of state of-the-art heuristics for the resource-constrained project scheduling problem. Eur Oper Res 127:394–407

Herroelen W, Demeulemeester E, De Reyck B (1998) Resource-constrained project scheduling: a survey of recent developments. Comput Oper 25(4):279–302

Icmeli O, Erenguc SS, Zappe CJ (1993) Project scheduling problems: a survey. Int J Oper Prod Man 13(11):80–91

Jia Q, Seo Y (2013) Solving resource-constrained project scheduling problems: conceptual validation of FLP formulation and efficient permutation-based ABC computation. Comput Oper Res 40(8):2037–2050

Khanzadi M, Soufipour R, Rostami M (2011) A new improved genetic algorithm approach and a competitive heuristic method for large-scale multiple resource-constrained project-scheduling problems. Int J Ind Eng Comput 2:737–748

Kochetov Y, Stolyar A (2003) Evolutionary local search with variable neighborhood for the resource constrained project scheduling problem. In: Proceedings of the 3rd international workshop of computer science and information technologies, Russia

Kolisch R (1996) Serial and parallel resource–constrained project scheduling methods revisited: theory and computation. Eur J Oper Res 90:320–333

Kolisch R, Drexl A (1996) Adaptive search for solving hard project scheduling problems. Nav Res Log 43(1):23–40

Kolisch R, Hartmann S (2006) Experimental investigation of heuristics for resource–constrained project scheduling: an update. Eur J Oper Res 174(1):23–37

Kolisch R, Sprecher A (1996) PSPLIB – a project scheduling problem library. Eur J Oper Res 96:205–216

Li K, Willis R (1992) An iterative scheduling technique for resource-constrained project scheduling. Eur J Oper Res 56(3):370–379

Mobini M, Rabbani M, Amalnik MS, Razmi J, Rahimi-Vahed AR (2009) Using an enhanced scatter search algorithm for a resource-constrained project scheduling problem. Soft Comput 13:597–610

Merkle D, Middendorf M, Schmeck H (2002) Ant colony optimization for resource-constrained project scheduling. IEEE Trans Evol Comput 6:333–346

Nonobe K, Ibaraki T (2002) Formulation and tabu search algorithm for the resource constrained project scheduling problem. In: Ribeiro CC, Hansen P (eds) Essays and surveys in metaheuristics. Kluwer, Boston, pp 557–588

Ozdamar L, Ulusoy G (1995) A survey on the resource-constrained project scheduling problem. IIE Trans 27:574–586

Pinson E, Prins C, Rullier F (1994) Using tabu search for solving the resource-constrained project scheduling problem. In: Proceedings of the 4th international workshop on project management and scheduling, Leuven, pp 102–106

Ranjbar M (2008) Solving the resource constrained project scheduling problem using filter-and-fan approach. Appl Math Comput 201:313–318

Ranjbar M, De Reyck B, Kianfar F (2009) A hybrid scatter search for the discrete time/resource trade-off problem in project scheduling. Eur J Oper Res 193:35–48

Sadeghi A, Kalanaki A, Noktehdan A, Samghabadi AS, Barzinpour F (2011) Using bees algorithm to solve the resource constrained project scheduling problem in PSPLIB. In: Zhou Q (ed) ICTMF 2011. CCIS, vol 164, pp 486–494

Sebt MH, Alipouri Y, Alipouri Y (2012) Solving resource-constrained project scheduling problem with evolutionary programming. J Oper Res Soc 62:1–9

Thomas PR, Salhi S (1998) A tabu search approach for the resource constrained project scheduling problem. J Heuristics 4:123–139

Tormos P, Lova A (2001) A competitive heuristic solution technique for resource constrained project scheduling. Ann Oper Res 102:65–81

Tormos P, Lova A (2003) An efficient multi-pass heuristic for project scheduling with constrained resources. Int J Prod Res 41(5):1071–1086

Tseng LY, Chen SC (2006) A hybrid metaheuristic for the resource-constrained project scheduling problem. Eur J Oper Res 175:707–721

Valls V, Ballestín F, Quintanilla MS (2004) A population-based approach to the resource-constrained project scheduling problem. Ann Oper Res 131:305–324

Valls V, Ballestín F, Quintanilla MS (2005) Justification and RCPSP: a technique that pays. Eur J Oper Res 165(2):375–386

Valls V, Ballestín F, Quintanilla MS (2008) A hybrid genetic algorithm for the resource constrained project scheduling problem. Eur J Oper Res 185:495–508

Zamani R (2013) A competitive magnet-based genetic algorithm for solving the resource-constrained project scheduling problem. Eur J Oper Res 229:552–559

Ziarati K, Akbari R, Zeighami V (2011) On the performance of bee algorithms for resource-constrained project scheduling problem. Appl Soft Comput 11:3720–3733

Part II
The Resource-Constrained Project Scheduling Problem with Generalized Precedence Relations

Chapter 5
Lower Bounds and Exact Solution Approaches

Lucio Bianco and Massimiliano Caramia

Abstract Generalized precedence relations are temporal constraints in which the starting/finishing times of a pair of activities have to be separated by at least or at most an amount of time denoted as time lag (minimum time lag and maximum time lag, respectively). This chapter is devoted to project scheduling with generalized precedence relations with and without resource constraints. Attention is focused on lower bounds and exact algorithms. In presenting existing results on these topics, we concentrate on recent results obtained by ourselves. The mathematical models and the algorithms presented here are supported by extensive computational results.

Keywords Generalized precedence relations • Makespan minimization • Project scheduling • Resource constraints

5.1 Introduction

Generalized Precedence Relations (GPRs) (Elmaghraby and Kamburowski 1992) are temporal constraints in which the starting/finishing times of a pair of activities have to be separated by at least or at most an amount of time denoted as "time lag" (minimum time lag and maximum time lag, respectively). GPRs can be classified into *Start-to-Start* (*SS*), *Start-to-Finish* (*SF*), *Finish-to-Start* (*FS*) and *Finish-to-Finish* (*FF*) relations.

A minimum time lag constraint $(SS_{ij}^{min}(\delta), SF_{ij}^{min}(\delta), FS_{ij}^{min}(\delta), FF_{ij}^{min}(\delta))$ specifies that activity j can start (finish) only if its predecessor i has started (finished) at least δ time units before. Analogously, a maximum time lag constraint $(SS_{ij}^{max}(\delta), SF_{ij}^{max}(\delta), FS_{ij}^{max}(\delta), FF_{ij}^{max}(\delta))$ imposes that activity j can be started (finished) at most δ time slots beyond the starting (finishing) time of activity i.

The introduction of GPRs has been stimulated by many practical applications. An example may be given by a company that must supply a client with a certain number

L. Bianco (✉) • M. Caramia
Department of Enterprise Engineering, University of Rome "Tor Vergata", Roma, Italy
e-mail: bianco@dii.uniroma2.it; caramia@dii.uniroma2.it

C. Schwindt, J. Zimmermann (eds.), *Handbook on Project Management and Scheduling Vol. 1*, International Handbooks on Information Systems,
DOI 10.1007/978-3-319-05443-8_5

of products which must be also assembled within 100 days; this relationship can be modeled as $SF_{ij}^{max}(100)$, which says that the assembly process (activity j) must finish at most 100 days after the starting time of the assembly (activity i) of the products. Another example may be borrowed from building construction companies that during the planning process often have to face with planning problems in which pairs of activities i, j have to be scheduled in such a way that the beginning of activity j must be at least δ units of time after the starting time of activity i. These situations can be represented by a constraint of the type $SS_{ij}^{min}(\delta)$.

GPRs can be represented in a so-called *standardized* form by transforming them, e.g., into minimum *Start-to-Start* precedence relationships by means of the so-called Bartusch et al.'s transformations (Bartusch et al. 1988). Thus, when applied to a given *Activity-on-Node* (AoN) network with GPRs, such transformations lead to a *standardized activity network* where, to each arc (i, j), a label δ_{ij} representing the time lag between the two activities i and j is associated (De Reyck 1998). If more than one time lag δ_{ij} between i and j exists, only the largest δ_{ij} is considered. In GPR networks, depending on the transformations, temporal relationships may produce cycles while preserving project feasibility, as it happens with Bartusch et al.'s transformations.

When no resource constraints are concerned the minimum completion time can be calculated in polynomial time by computing the longest path from the source to the sink in such a network. In fact it is well known that, in presence of GPRs and not positive cycles, such longest path can be computed by means of the algorithm of Floyd and Warshall (see, e.g., Ahuja et al. 1993) which complexity is $\mathcal{O}(n^3)$ where n is the number of network nodes (i.e., the number of real activities of the project). A variant of this algorithm, proposed by Bellman–Ford–Moore (see, Ahuja et al. 1993), can be also utilized. It has a complexity $\mathcal{O}(n \cdot m)$ where n is the number of nodes (i.e., the number of real activities of the project) and m the number of arcs of the network. Bianco and Caramia (2010) proposed a new formulation of the Resource Unconstrained Project Scheduling Problem (RUPSP) with GPRs which permits to compute the minimum project duration in $\mathcal{O}(m)$ time where m is the number of precedence relations. This formulation will be described in Sect. 5.2. When resource constraints are involved the problem is denoted as Resource Constrained Project Scheduling Problem (RCPSP) with GPRs. It is strongly $\mathcal{NP}$-hard and also the easier problem of detecting whether a feasible solution exists is $\mathcal{NP}$-complete (Bartusch et al. 1988).

To the best of our knowledge, the exact procedures presented in the literature for such a problem are the branch-and-bound algorithms by Bartusch et al. (1988), Demeulemeester and Herroelen (1997b), De Reyck (1998), Schwindt (1998), Fest et al. (1999), Dorndorf et al. (2000). The paper by Bartusch et al. (1988) reports a limited computational experience on a case study. The paper by Demeulemeester and Herroelen (1997b) is conceived to work on minimum time lags only. The other three algorithms work with both minimum and maximum time lags. In particular, De Reyck and Herroelen (1998) present results on projects with 30 activities and percentages of maximum time lags of 10 and 20 % with respect to the total number of generalized precedence relations. The branch-and-bound algorithm by Schwindt (1998) is characterized by delaying activities by adding special precedence constraints, i.e., disjunctive precedence constraints, between sets of activities rather than pair of activities as done by the above mentioned approaches. The algorithm by Fest et al. (1999) differs from that of Schwindt (1998) by making use of release dates for certain activities in place of disjunctive precedence constraints used in the previous approach. The algorithm by Dorndorf et al. (2000) uses constraint-propagation techniques which check whether certain start times can be excluded from the computation since they cannot produce a feasible, active or optimal schedule.

Also lower bounds are available for this problem. In particular, two classes of lower bounds are well known in the literature, i.e., constructive and destructive lower bounds. The first class is formed by those lower bounds associated with relaxations of the mathematical formulation of the problem (for instance, the critical path lower bound and the basic resource based lower bound; see, e.g., Demeulemeester and Herroelen 2002). Destructive lower bounds, instead, are obtained by means of an iterated binary search based routine as reported, e.g., in Klein and Scholl (1999). Also De Reyck and Herroelen (1998) proposed a lower bound for the RCPSP-GPRs denoted with lb3-gpr that is the extension of the lower bound lb3 proposed by Demeulemeester and Herroelen (1997a) for the RCPSP.

Additional literature on RCPSP with GPRs can be found in the books of Dorndorf (2002), Neumann et al. (2003).

In Sect. 5.2 a new mathematical formulation for the RUPSP with GPRs is illustrated (Bianco and Caramia 2010). In Sect. 5.3.1 and in Sect. 5.3.2 a new lower bound and a new mathematical formulation with an exact solution algorithm proposed by Bianco and Caramia (2011a, 2012) are described, respectively.

5.2 A New Formulation of the Resource Unconstrained Project Scheduling Problem with GPRs

It is well known in project scheduling that PERT/CPM methods can be applied under two assumptions. The first one is that resources are available in infinite amounts, and the second is that the precedence relationships between two activities are only of the Finish-to-Start type with time lags equal to zero; this implies that an activity can start only as soon as all its predecessors have finished.

In this context, one can define an acyclic network whose nodes are the activities and arcs are the precedence constraints (AoN network), and compute the minimum project completion time as the length of the critical path, i.e., the longest path from the initial activity to the final activity in such an activity network (see, e.g., Moder et al. 1983; Radermacher 1985).

The computation of the critical path can be accomplished by means of the well-known forward pass recursion algorithm (see, e.g., Kelley 1963), that is a classical label setting algorithm for longest path calculation. The computational complexity of this algorithm is $\mathscr{O}(m)$, where m is the number of arcs of the network;

when the graph has a high density, the complexity tends to $\mathcal{O}(n^2)$, where n is the number of real activities.

When GPRs are present, the standardized network, as mentioned in the introduction may contain cycles.

This implies that the RUPSP with GPRs and minimum completion time objective:

- cannot be solved by the classical forward and backward recursion algorithm;
- can be solved by computing the longest path on the standardized network from the source node to the sink node, whose length is the sum of the time lags associated with its arcs, using an algorithm working on "general" networks with a worst-case complexity $\mathcal{O}(n \cdot m)$ (see, e.g., Ahuja et al. 1993);
- does not admit feasible solutions when the topology of the network and the arc-length function induce one or more directed cycles of positive length (see, e.g., Demeulemeester and Herroelen 2002).

In the following, the new network model for RUPSP with GPRs, mentioned before, is described. It is based on a new formulation such that the standardized network obtained is always acyclic since the precedence relationships between each pair of activities are only of the Finish-to-Start type with zero time lags.

5.2.1 A Network Formulation of RUPSP with GPRs

For the sake of completeness, before presenting such a new formulation, we show how project scheduling with GPRs and minimum makespan (without resource constraints) is formulated in the literature. First GPRs are all converted into Start-to-Start minimum time lag relationships by means of the Bartusch et al.'s transformation (Bartusch et al. 1988) where δ_{ij} is the generic time lag between activities i and j; then, denoting with $n+1$ the (dummy) sink node of the project network and with E the set of pairs of activities constrained by GPRs, we have

$$
\begin{aligned}
\text{Min. } & S_{n+1} \\
\text{s.t. } & S_j \geq S_i + \delta_{ij} \quad ((i,j) \in E) \\
& S_i \geq 0 \quad (i \in V)
\end{aligned}
$$

where S_i is the starting time of activity i, and V is the set of activities to be carried out.

Let us consider now an acyclic AoN network, representing a given project, where the labels on the nodes denote the activity durations while the labels on the arcs encode time lags. An example is the network in Fig. 5.1, where node 0 and node 5 are dummy nodes (the source and the sink of the network, respectively), i.e., they have zero durations.

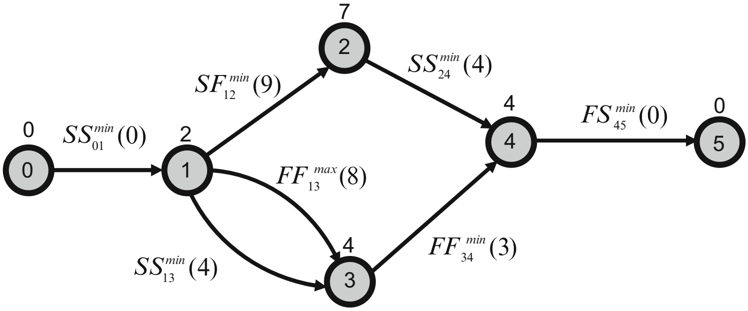


Fig. 5.1 A network with GPRs

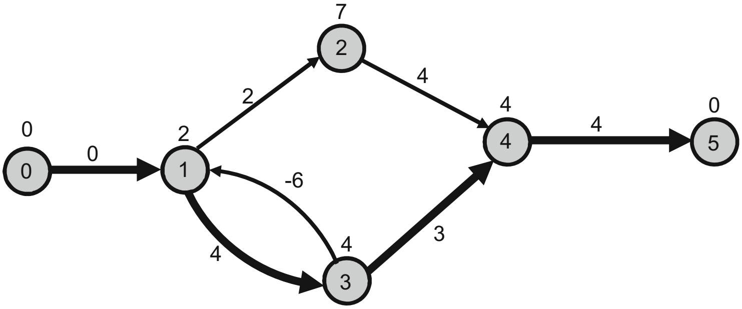


Fig. 5.2 The AoN standardized network and its critical path (*bold arcs*)

If all the GPRs are transformed into SS^{min}_{ij} type precedence relationships by means of the Bartusch et al.'s transformations, we obtain the standardized network in Fig. 5.2, where the label on a generic arc (i, j) is the Start-to-Start minimum time lag between i and j, and the label on the generic node i is the duration p_i of the corresponding activity i (note that the network contains a cycle).

Let us now examine whether an acyclic AoN network with GPRs can be transformed into an AoN acyclic network with all constraints of the Finish-to-Start type and zero time lag. To this aim let us consider all GPRs between two activities i and j. They have the following form:

$$S_i + SS^{min}_{ij} \le S_j \le S_i + SS^{max}_{ij}$$

$$S_i + SF^{min}_{ij} \le C_j \le S_i + SF^{max}_{ij}$$

$$C_i + FS^{min}_{ij} \le S_j \le C_i + FS^{max}_{ij}$$

$$C_i + FF^{min}_{ij} \le C_j \le C_i + FF^{max}_{ij}$$

where $S_i(S_j)$ denotes the starting time, and $C_i(C_j)$ the finishing time of activity $i(j)$. Moreover, $SS^{min}_{ij}, SF^{min}_{ij}, FS^{min}_{ij}, FF^{min}_{ij}$ are the minimum time lags, while

$SS_{ij}^{max}, SF_{ij}^{max}, FS_{ij}^{max}, FF_{ij}^{max}$ are the maximum time lags, each one corresponding to the different type of constraint between activities i and j.

These relations can be transformed in terms of FS_{ij} constraints by means of the following transformation rules:

$$S_i + SS_{ij}^{min} \leq S_j \rightarrow C_i + \hat{\delta}_{ij} \leq S_j, \text{ with } \hat{\delta}_{ij} = -p_i + SS_{ij}^{min}$$

$$S_j \leq S_i + SS_{ij}^{max} \rightarrow S_j \leq C_i + \hat{\delta}_{ij}, \text{ with } \hat{\delta}_{ij} = -p_i + SS_{ij}^{max}$$

$$S_i + SF_{ij}^{min} \leq C_j \rightarrow C_i + \hat{\delta}_{ij} \leq S_j, \text{ with } \hat{\delta}_{ij} = -p_i - p_j + SF_{ij}^{min}$$

$$C_j \leq S_i + SF_{ij}^{max} \rightarrow S_j \leq C_i + \hat{\delta}_{ij}, \text{ with } \hat{\delta}_{ij} = -p_i - p_j + SF_{ij}^{max}$$

$$C_i + FS_{ij}^{min} \leq S_j \rightarrow C_i + \hat{\delta}_{ij} \leq S_j, \text{ with } \hat{\delta}_{ij} = FS_{ij}^{min}$$

$$S_j \leq C_i + FS_{ij}^{max} \rightarrow S_j \leq C_i + \hat{\delta}_{ij}, \text{ with } \hat{\delta}_{ij} = FS_{ij}^{max}$$

$$C_i + FF_{ij}^{min} \leq C_j \rightarrow C_i + \hat{\delta}_{ij} \leq S_j, \text{ with } \hat{\delta}_{ij} = FF_{ij}^{min} - p_j$$

$$C_j \leq C_i + FF_{ij}^{max} \rightarrow S_j \leq C_i + \hat{\delta}_{ij}, \text{ with } \hat{\delta}_{ij} = FF_{ij}^{max} - p_j$$

Now, it is possible to define a new AoN network where between each pair of nodes i, j of the original network related to a temporal constraint, we insert a dummy activity (dummy node) i' whose duration $p_{i'}$ is

$$p_{i'} \geq \hat{\delta}_{ij}$$

in the case of minimum time lag, and

$$p_{i'} \leq \hat{\delta}_{ij}$$

in the case of maximum time lag. Note that $\hat{\delta}_{ij}$ in the former case is a lower bound on the value of $p_{i'}$, and in the latter case is an upper bound on its value.

This new network is acyclic and will have only precedence relationships of the *FS* type and zero time lags since the latter are embedded in durations $p_{i'}$ associated with the dummy activities.

Referring to the previous example, the transformed AoN network is represented in Fig. 5.3, where, based on the proposed transformation, $p_{0'} \geq 0$, $p_{1'} \geq 2$, $p_{1''} \leq 4$, $p_{1'''} \geq 0$, $p_{2'} \geq -3$, $p_{3'} \geq -1$, $p_{4'} \geq 0$.

The minimum completion time of the project is then given by the earliest starting time of node 5. In the next section, we give a mathematical programming formulation of the problem now described.

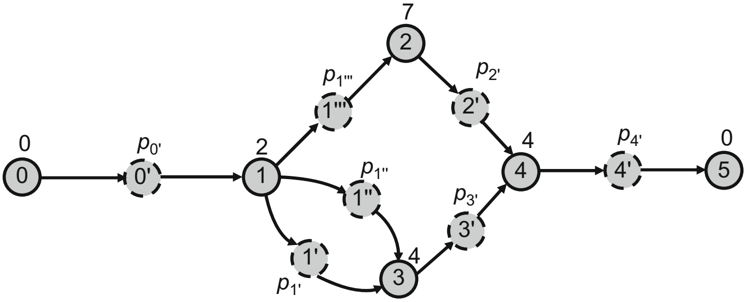


Fig. 5.3 The transformed acyclic network

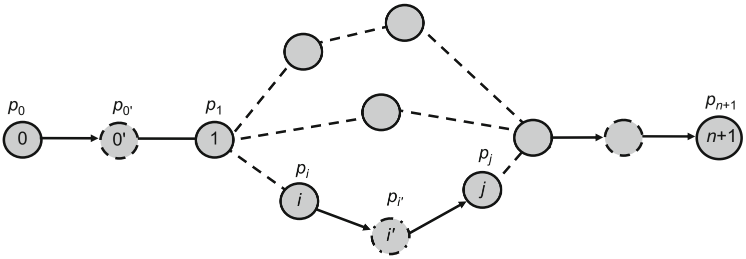


Fig. 5.4 A generic transformed AoN network

5.2.2 A Mathematical Programming Formulation of RUPSP with GPRs

Following the presentation in the previous section, given a project with GPRs, we can use the AoN network representation shown in Fig. 5.4, where temporal constraints are of the *FS* type only with zero time lags.

Let us denote by

- V the set of the nodes of the original network;
- $\bar{V}$ the set of the dummy nodes of the transformed network;
- $V \cup \bar{V}$ the set of all nodes of the transformed network;
- $\bar{U}$ the subset of nodes in $\bar{V}$ corresponding to minimum time lag constraints in the original network.

The problem of finding the minimum completion time of a project is that of computing the minimum time to reach node $n + 1$, and can be formulated in terms of linear programming as follows:

$$\begin{aligned} &\text{Min. } S_{n+1} \\ &\text{s.t.} \quad S_{i'} = S_i + p_i \quad (i' \in \bar{V}; i \in \mathit{Pred}(i')) \qquad (5.1) \end{aligned}$$

$$S_i = S_{i'} + p_{i'} \quad (i \in V; i' \in Pred(i)) \tag{5.2}$$

$$p_{i'} \geq \ell_{i'} \quad (i' \in \bar{U} \subseteq \bar{V}) \tag{5.3}$$

$$p_{i'} \leq \ell_{i'} \quad (i' \in \bar{V} \setminus \bar{U}) \tag{5.4}$$

$$S_i \in \mathbb{R}_{\geq 0} \quad (i \in V \setminus \{0\}) \tag{5.5}$$

$$S_{i'} \in \mathbb{R}_{\geq 0} \quad (i' \in \bar{V}) \tag{5.6}$$

$$S_0 = 0 \tag{5.7}$$

where

- p_i is the duration of activity $i \in V$; we assume, without loss of generality, that these durations are positive integer values;
- $\ell_{i'}$ is the bound on the duration $p_{i'}$, equal to the $\hat{\delta}_{ij}$ value obtained by means of the Finish-to-Start transformation rules;
- $Pred(i)$ and $Pred(i')$ are, respectively, the predecessor node sets of i and i';
- $S_i\,(S_{i'})$ is the starting time of activity $i \in V(i' \in \bar{V})$;
- $p_{i'}$ is the duration of activity $i' \in \bar{V}$.

Note that equality constraints (5.1) are guaranteed since each dummy node i' has a unique predecessor and, therefore, its starting time is equal to the sum of the starting time and the duration of its (unique) predecessor. Equality constraints (5.2) can be guaranteed even if a node i can admit more than one predecessor $i' \in Pred(i)$; indeed, we just need to observe that $p_{i'}$, with $i' \in Pred(i)$, are variables and, hence, can assume proper values [respecting (5.3) and (5.4)] that allow equalities to be satisfied.

The above mathematical program can be solved in $\mathcal{O}(m)$. In fact, it can be observed that constraints (5.3) and (5.4) can be merged with constraints (5.2) obtaining the following mathematical program:

$$\begin{aligned}
\text{Min.}\ & S_{n+1} \\
\text{s.t.}\ & S_{i'} = S_i + p_i \quad (i' \in \bar{V}; i \in Pred(i')) \\
& S_i \geq S_{i'} + \ell_{i'} \quad (i \in V; i' \in Pred(i) \cap \bar{U}) \\
& S_i \leq S_{i'} + \ell_{i'} \quad (i \in V; i' \in Pred(i) \cap (\bar{V} \setminus \bar{U}) \\
& S_i \in \mathbb{R}_{\geq 0} \quad (i \in V \setminus \{0\}) \\
& S_{i'} \in \mathbb{R}_{\geq 0} \quad (i' \in \bar{V}) \\
& S_0 = 0
\end{aligned}$$

The above formulation is solvable in time linear with the number of arcs for acyclic networks using a dynamic programming recursion (see, e.g., Hochbaum and Naor 1994).

Solving RUPSP with GPRs in $\mathscr{O}(m)$ time represents a novel result, compared to the currently known $\mathscr{O}(n \cdot m)$ time complexity associated with the approach of Bartusch et al. (1988) mentioned before.

In Bianco and Caramia (2010) it is also shown that:

- the minimum completion time of a project with minimum time lags only is given by the length of the longest path in the network, where the path length is given by the sum of the node weights on the path (see Proposition 3);
- when minimum and maximum time lags are present simultaneously, by exploiting the dual formulation of the mathematical program (5.1)–(5.7), the minimum completion time can be achieved with the same computational complexity $\mathscr{O}(m)$ by finding an augmenting path of longest length from node 0 to node $n + 1$ in the proposed acyclic network in which a unit capacity is installed on each arc (see Propositions 4 and 5).

The main results contained in Bianco and Caramia (2010) when minimum and maximum time lags are present simultaneously state that in the acyclic network two consecutive minimum and maximum time lag arcs have not to be oriented in the same direction when they are in the optimal augmenting path. To this end one must impose a starting flow vector whose components corresponding to minimum time lag arcs are zero and those associated with maximum time lag arcs are one. Since the flow vector must satisfy the mass balance constraints, the source is connected with the tail of each maximum time lag arc (i, i'), when on this tail node at least one minimum time lag arc is incident. Analogously, the head of each maximum time lag arc (i', i) is connected with the sink when on this head node at least one minimum time lag arc is incident. To such additional dummy arcs (denote with E^d this set of dummy arcs) a flow equal to one is assigned.

Note that the length of an augmenting path in the network can be computed as the sum of its arc weights where, according to the dual formulation and the main results proved in the paper:

- the weight of a minimum time lag arc (i, i') is given by p_i;
- the weight of a minimum time lag arc (i', i) is given by $\ell_{i'}$;
- the weight of a maximum time lag arc (i, i') is given by $-p_i$;
- the weight of a maximum time lag arc (i', i) is given by $-\ell_{i'}$
- the weight of dummy arcs in D is $-\infty$, since they must not be traversed by the augmenting path.

We finally note that the proposed approach is also able to cope with infeasibility of project without any additional computing time.

In order to analyze how the proposed model behaves experimentally a comparison with Bartusch et al.'s approach (using the $\mathscr{O}(n \cdot m)$ Bellman–Ford–Moore label correction algorithm (see Ahuja et al. 1993) has been carried out using benchmarks taken from the library of projects at the web page http://www.wiwi.tu-clausthal.de/abteilungen/produktion/forschung/schwerpunkte/project-generator.

This site contains projects of sizes ranging from 10 to 1,000 activities, where each group of projects has 90 instances each with 5 resources. The authors experimented with projects having 100 and 1,000 activities, neglecting resource constraints, and reported extensively the running times employed by the two approaches to find the minimum completion time. In the approach proposed by Bianco and Caramia running times on networks with 100 and 1,000 activities oscillate around between 10 and 100 s for 10^4 replications of the algorithm, respectively. Using instead the Bartusch et al.'s approach, running times are very close to 100 and 1,000 s, respectively, for 10^4 replications of the algorithm.

5.2.3 An Example

In this section we show an example on how the approach described in the previous section works. Consider the network in Fig. 5.5. By applying the transformation described in Sect. 5.2.1, we obtain the network depicted in Fig. 5.6. In Fig. 5.7, we display the network with weights on arcs, where the label on a generic arc represents the cost (time) to traverse such an arc.

Note that:

- paths $0-0'-1-1'-2-2'-3-3'-4-4'-5$ and $0-2-2'-3-5$ are not feasible; indeed, the unit of flow entering at node 2, in both paths, cannot traverse arc $(2,2')$ since the latter is saturated.

Therefore, the unit of flow can be routed onto three different augmenting paths, i.e.,

$$P_1:\ 0-0'-1-1'-2-2''-4-4'-5$$
$$P_2:\ 0-0'-1-1''-3-3'-4-4'-5$$
$$P_3:\ 0-0'-1-1''-3-2'-2-2''-4-4'-5$$

Augmenting paths P_1 and P_2 have length equal to 11, while path P_3 (depicted in bold in Fig. 5.7) is an augmenting path of longest length (the critical path), and gives the minimum completion time of the project equal to 12 time units.

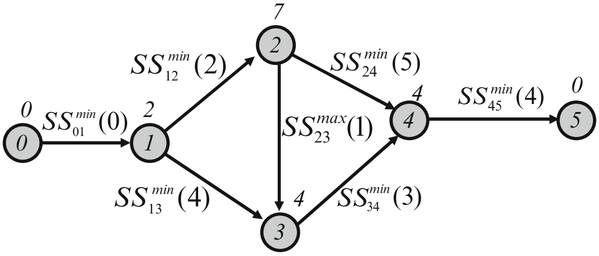


Fig. 5.5 An example of GPRs network

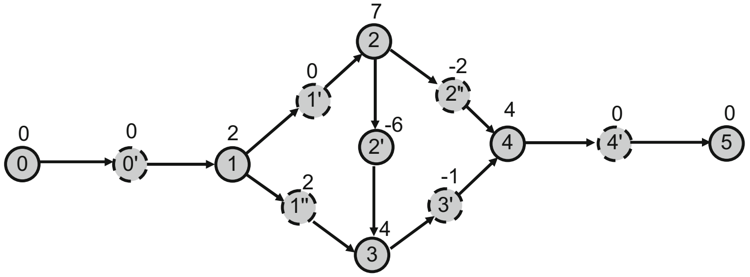


Fig. 5.6 The transformed GPRs network

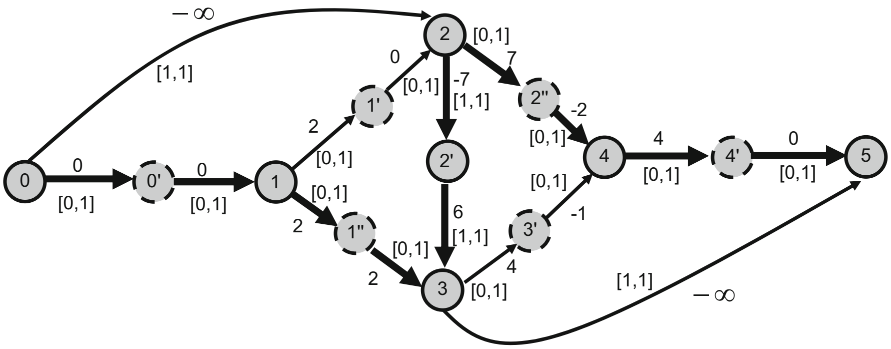


Fig. 5.7 The network with weights, flows and capacities on arcs. Note that labels [·, ·] contain arc flow and arc capacity, respectively, for each arc

5.3 The Resource Constrained Project Scheduling Problem with GPRs

In this section, the RCPSP with GPRs is considered and the most recent results obtained by Bianco and Caramia (2011a, 2012) are illustrated.

5.3.1 A New Lower Bound for RCPSP with GPRs

In Bianco and Caramia (2011a) a new lower bound is proposed. This lower bound is based on a relaxation of the resource constraints among independent activities and on a solution of the relaxed problem suitably represented by means of an AoN acyclic network. In particular, the network model proposed in Bianco and Caramia (2010) is exploited to try to get rid of the resource constraints. The analysis is restricted only on those pairs of activities for which a GPR exists to determine a lower bound on the minimum makespan. For each of these pairs it is verified whether the amount of resources requested exceeds the resource availability, for at least one resource type. In case of a positive answer, some results which allow the reduction of the problem to a new resource unconstrained project scheduling problem with different lags and additional disjunctive constraints are obtained. This last problem can be formulated as an integer linear program whose linear relaxation can be solved by means of a network flow approach (see also Bianco and Caramia 2010). Computational results confirm a better practical performance of the proposed lower bound with respect to the aforementioned ones in the introduction.

5.3.1.1 Network Representation

In order to represent the relaxed problem mentioned before on an acyclic network, it is necessary to transform all constraints in terms of Finish-to-Start relations with zero time lags. To this end let us consider a generic pair of activities i, j. The related resource incompatibility imposes that

$$S_j \geq S_i + p_i \tag{5.8}$$

or, alternatively,

$$S_i \geq S_j + p_j \tag{5.9}$$

Let us examine now how the resource constraint combines with the different GPRs. As far as constraint $SS_{ij}^{min}(\delta)$, with $\delta \geq 0$, is concerned, we note that it can be transformed in $C_i + \tilde{\delta}_{ij}$ where $\tilde{\delta}_{ij} = -p_i + \delta$. Two cases must be considered:

1. $-p_i + \delta \geq 0$, i.e., $SS_{ij}^{min}(\delta)$ dominates constraint (5.8);
2. $-p_i + \delta < 0$, i.e., constraint (5.8) dominates $SS_{ij}^{min}(\delta)$;

Constraint (5.9) does not play any role since it is not compatible with $SS_{ij}^{min}(\delta)$. Therefore, a resource incompatibility constraint between i and j joined with a $SS_{ij}^{min}(\delta)$ relation can be expressed as a $FS_{ij}^{min}(\delta')$ relation where, $\delta' = \max\{0, \tilde{\delta}_{ij}\}$. A similar analysis has been conducted in De Reyck and Herroelen (1998) (see Theorem 6). Analogously, the association of a resource constraint with a $SF_{ij}^{min}(\delta)$, $FS_{ij}^{min}(\delta)$, $FF_{ij}^{min}(\delta)$, leads to a GPR constraint of the type $FS_{ij}^{min}(\delta')$, where $\delta' = \max\{0, \tilde{\delta}_{ij}\}$ and $\tilde{\delta}_{ij} = -p_i - p_j + \delta$, $\tilde{\delta}_{ij} = \delta$, $\tilde{\delta}_{ij} = \delta - p_j$, respectively. The four cases with minimum time lags can thus be represented on an AoN transformed network as depicted in Fig. 5.8.

In the latter figure, i' is a dummy node and the relations between i, i' and i', j are of the Finish-to-Start type with zero time lags. The duration $p_{i'}$ is known only in terms of lower bound. Let us now examine the maximum time lag different scenarios.

Constraint $SS_{ij}^{max}(\delta)$ implies that $S_j \leq C_i + \tilde{\delta}_{ij}$, where $\tilde{\delta}_{ij} = -p_i + \delta$. We have two cases:

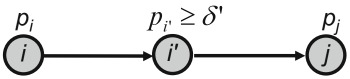


Fig. 5.8 Representation of a resource incompatibility constraint and a minimum time lag in the transformed acyclic network

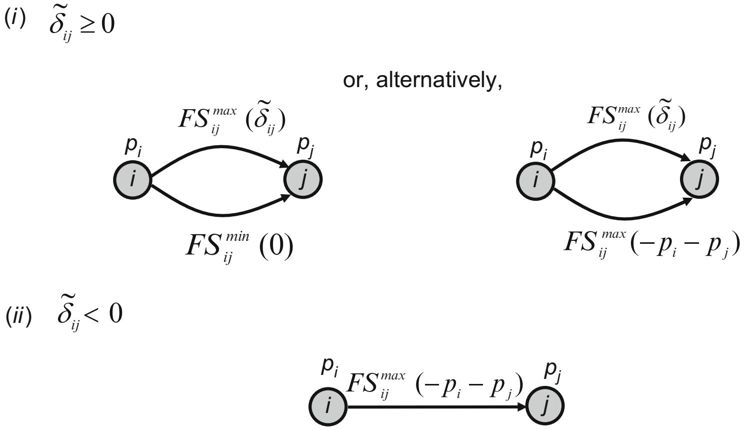

(i) $\tilde{\delta}_{ij} \geq 0$
or, alternatively,
$FS_{ij}^{max}(\tilde{\delta}_{ij})$
p_i
p_j
i
j
$FS_{ij}^{min}(0)$
$FS_{ij}^{max}(\tilde{\delta}_{ij})$
p_i
p_j
i
j
$FS_{ij}^{max}(-p_i - p_j)$
(ii) $\tilde{\delta}_{ij} < 0$
p_i
$FS_{ij}^{max}(-p_i - p_j)$
p_j
i
j

Fig. 5.9 Standardized representation of a resource incompatibility constraint combined with a maximum time lag in terms of Finish-to-Start relations

1. $-p_i + \delta \geq 0$. In this case $SS_{ij}^{max}(\delta)$ is compatible with both the relations (5.8) and (5.9) related to the resource incompatibility constraints. In fact one of the following two conditions can be verified, i.e., either

$$\begin{cases} S_j \leq C_i + \tilde{\delta}_{ij} \\ S_j \geq S_i + p_i \end{cases} \tag{5.10}$$

   or

$$\begin{cases} S_j \leq C_i + \tilde{\delta}_{ij} \\ S_i \geq S_j + p_j \end{cases} \tag{5.11}$$

   Obviously, since $\delta \geq p_i$, in alternative (5.11) resource incompatibility constraint (5.9) dominates $SS_{ij}^{max}(\delta)$.
2. $-p_i + \delta < 0$. In this case only alternative (5.11) is valid since resource incompatibility constraint (5.8) is not compatible with $SS_{ij}^{max}(\delta)$. Therefore, resource incompatibility constraint (5.9) dominates $SS_{ij}^{max}(\delta)$.

For $SF_{ij}^{max}(\delta)$, $FS_{ij}^{max}(\delta)$, $FF_{ij}^{max}(\delta)$ we have similar occurrences where $\tilde{\delta}_{ij} = -p_i - p_j + \delta$, $\tilde{\delta}_{ij} = \delta$, $\tilde{\delta}_{ij} = \delta - p_j$, respectively. All the four GPRs with maximum time lags correspond to the two Finish-to-Start scenarios depicted in Fig. 5.9.

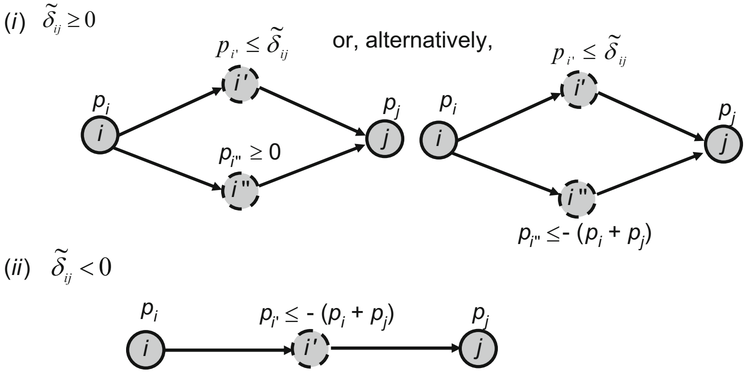

(i) $\tilde{\delta}_{ij} \geq 0$
$p_{i'} \leq \tilde{\delta}_{ij}$
or, alternatively,
$p_{i'} \leq \tilde{\delta}_{ij}$
p_i
i
i'
$p_{i''} \geq 0$
i''
p_j
j
p_i
i
i'
i''
p_j
j
$p_{i''} \leq - (p_i + p_j)$
(ii) $\tilde{\delta}_{ij} < 0$
p_i
i
$p_{i'} \leq - (p_i + p_j)$
i'
p_j
j

Fig. 5.10 Representation of a resource incompatibility constraint combined with a maximum time lag in the transformed acyclic network

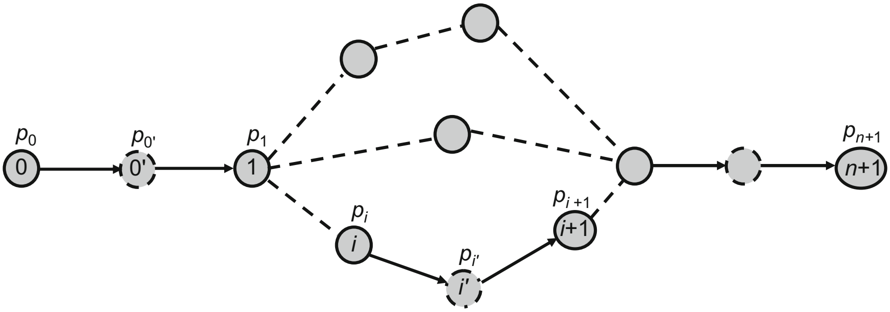


Fig. 5.11 A representation on the AoN acyclic network of the relaxed RCPSP with GPRs

This representation on an AoN network can be transformed in a network representation with zero time lags, as depicted in Fig. 5.10.

In the latter picture, the dashed nodes are dummy nodes and the relation between any pair of adjacent nodes is of the type Finish-to-Start with zero time lag. The durations associated with the dummy nodes are known only in terms of upper bounds and lower bounds.

5.3.1.2 A Mathematical Programming Formulation

Following what we presented in the previous section, given a project with GPRs and resource incompatibility constraints only between pairs of activities with GPRs, we can use the AoN network representation, shown in Fig. 5.11, where only temporal constraints of the Finish-to-Start type with zero time lags are present.

The minimum S_{n+1} is the optimal solution of the relaxed RCPSP with GPRs, and then a lower bound of the original problem. Let us now denote:

- V the set of the nodes of the original network;
- $\bar{V} = \bar{V}_g \cup \bar{V}_G \cup \bar{V}_m \cup \bar{V}_M$ the set of dummy nodes of the transformed network;
- $\bar{V}_g$ the subset of dummy nodes derived from the GPRs with minimum time lags among pairs of activities for which no resource incompatibility exist;
- $\bar{V}_G$ the subset of dummy nodes derived from the GPRs with maximum time lags among pairs of activities for which no resource incompatibility exist;
- $\bar{V}_m$ the subset of dummy nodes derived from the GPRs with minimum time lags among pairs of activities for which resource incompatibility exist;
- $\bar{V}_M = \bar{V}'_M \cup \bar{V}''_M$ the subset of dummy nodes derived from the GPRs with maximum time lags among pairs of activities for which resource incompatibility exist;
- $\bar{V}'_M$ the subset of $\bar{V}_M$ such that $\tilde{\delta}_{ij} \geq 0$;
- $\bar{V}''_M$ the subset of $\bar{V}'_M$ such that $\tilde{\delta}_{ij} < 0$.

Moreover, $\bar{V}'_M = \bar{V}'_{M_1} \cup \bar{V}'_{M_2}$ where $\bar{V}'_{M_1}$ is the subset of dummy nodes representing only the maximum time lag constraints and $\bar{V}'_{M_2}$ is the subset of dummy nodes representing only the resource incompatibility constraints compatible with the GPRs with maximum time lags (see Fig. 5.10). Of course $|\bar{V}'_{M_1}| = |\bar{V}'_{M_2}|$.

Defining a binary variable $w_{i'}$ to model the disjunctive scenarios defined in the previous section, the problem of finding the minimum S_{n+1} is that of computing the minimum time to reach node $n+1$ on that network and can be formulated in terms of mixed integer programming as follows:

$$\text{Min. } S_{n+1}$$

$$\text{s.t. } S_{i'} = S_i + p_i \quad (i' \in \bar{V}; i \in Pred(i') \subseteq V) \tag{5.12}$$

$$S_i = S_{i'} + p_{i'} \quad (i \in V; i' \in Pred(i) \subseteq \bar{V}) \tag{5.13}$$

$$p_{i'} \geq \ell_{i'} \quad \ell_{i'} = \delta', (i' \in \bar{V}_m); \ell_{i'} = \tilde{\delta}_{ij}, (i' \in \bar{V}_g) \tag{5.14}$$

$$p_{i'} \leq \ell_{i'} \quad \ell_{i'} = \tilde{\delta}_{ij}, (i' \in \bar{V}'_{M_1} \cup \bar{V}_G) \tag{5.15}$$

$$p_{i'} \geq -M \cdot w_{i'} + \ell_{i'} \quad \ell_{i'} = 0, (i' \in \bar{V}'_{M_2}) \tag{5.16}$$

$$p_{i'} \leq M \cdot (1 - w_{i'}) + \ell_{i'} \quad \ell_{i'} = -p_i - p_j, (i' \in \bar{V}'_{M_2}) \tag{5.17}$$

$$p_{i'} \leq \ell_{i'} \quad \ell_{i'} = -p_i - p_j, (i' \in \bar{V}''_M) \tag{5.18}$$

$$w_{i'} \in \{0, 1\} \quad (i' \in \bar{V}'_{M_2}) \tag{5.19}$$

$$S_i \in \mathbb{R}_{\geq 0} \quad (i \in V \setminus \{0\}) \tag{5.20}$$

$$S_{i'} \in \mathbb{R}_{\geq 0} \quad (i' \in \bar{V}) \tag{5.21}$$

$$S_0 = 0 \tag{5.22}$$

Constraints (5.12) and (5.13) model the Finish-to-Start relations in the acyclic network. Constraints (5.14), (5.15), (5.16), (5.17) and (5.18) model the bound on the values of $p_{i'}$ found from the analysis in the previous section. In particular, note that constraints (5.16) and (5.17) refer to the disjunctive situations depicted in Figs. 5.9 and 5.10. In these constraints M is a very large positive number such that if $w_{i'} = 1$ constraints (5.16) are trivially satisfied and constraints (5.17) are effective; if $w_{i'} = 0$ we have the opposite situation, i.e., constraints (5.16) are effective and constraints (5.17) are always satisfied. Constraints (5.19), (5.20), (5.21) and (5.22) define the range of variability of the decision variables.

Let us consider the linear relaxation of the above formulation, i.e., replacing constraint (5.19) with

$$w_{i'} \leq 1 \quad (i' \in \bar{V}'_{M_2}) \tag{5.23}$$

$$w_{i'} \geq 0 \quad (i' \in \bar{V}'_{M_2}) \tag{5.24}$$

and let us define the corresponding dual problem. To this end let us define the following dual variables $y_{ii'}$, $y_{i'i}$, $z_{i'}$, $z^a_{i'}$, $z^b_{i'}$, $\bar{w}_{i'}$ corresponding to constraints (5.12), (5.13), [(5.14), (5.15) and (5.18)], (5.16), (5.17), [(5.23) and (5.24)], respectively. The dual formulation of the previous relaxed problem is the following:

$$\text{Max.}\ \{\sum_{i \in V} \sum_{i' \in Succ(i)} p_i\, y_{ii'} + \sum_{i' \in \bar{V}_m \cup \bar{V}_g} \ell_{i'} z_{i'} + \sum_{i' \in \bar{V}'_{M_1} \cup \bar{V}''_M \cup \bar{V}_G} \ell_{i'} z_{i'} +$$

$$\sum_{i' \in \bar{V}'_{M_2}} (-M + p_i + p_j) \cdot z^b_{i'} + \sum_{i' \in \bar{V}'_{M_2}} \bar{w}_{i'}\}$$

$$\text{s.t.} \quad \sum_{i \in Pred(i')} y_{ii'} - \sum_{i \in Succ(i')} y_{i'i} \leq 0 \quad (i' \in \bar{V}) \tag{5.25}$$

$$\sum_{i' \in Pred(i)} y_{i'i} - \sum_{i' \in Succ(i)} y_{ii'} \leq 0 \quad (i \in V \setminus \{0, n+1\}) \tag{5.26}$$

$$\sum_{i' \in Pred(n+1)} y_{i'n+1} \leq 1 \tag{5.27}$$

$$z_{i'} - y_{i'i} = 0 \quad (i' \in \bar{V}_m \cup \bar{V}'_{M_1} \cup \bar{V}''_M \cup \bar{V}_g \cup \bar{V}_G) \tag{5.28}$$

$$-\, y_{i'i} + z^a_{i'} - z^b_{i'} = 0 \quad (i' \in \bar{V}'_{M_2}) \tag{5.29}$$

$$-\, M \cdot z^b_{i'} + M \cdot z^a_{i'} + \bar{w}_{i'} \leq 0 \quad (i' \in \bar{V}'_{M_2}) \tag{5.30}$$

$$y_{ii'} \text{ free} \quad (i \in V; i' \in Succ(i)) \tag{5.31}$$

$$y_{i'i} \text{ free} \quad (i' \in \bar{V}; i \in Succ(i')) \tag{5.32}$$

$$z_{i'} \geq 0 \quad (i' \in \bar{V}_m \cup \bar{V}_g) \tag{5.33}$$

$$z_{i'} \leq 0 \quad (i' \in \bar{V}'_{M_1} \cup \bar{V}''_M \cup \bar{V}_G) \tag{5.34}$$

$$\bar{w}_{i'} \leq 0 \quad (i' \in \bar{V}'_{M_2}) \tag{5.35}$$

$$z^a_{i'} \geq 0 \quad (i' \in \bar{V}'_{M_2}) \tag{5.36}$$

$$z^b_{i'} \geq 0 \quad (i' \in \bar{V}'_{M_2}) \tag{5.37}$$

5.3.1.3 Analysis on the Primal-Dual Formulation: Main Result

In this section, we provide the main result on the possibility to find the optimal solution to the problem in polynomial time $\mathscr{O}(m)$. Exploiting the equality constraints (5.28) and (5.29) by substituting (5.28) in the objective function and (5.29) in constraint (5.30), we have the following equivalent mathematical formulation:

$$\text{Max.} \{\sum_{i \in V} \sum_{i' \in Succ(i)} p_i y_{ii'} + \sum_{i \in Succ(i')} \sum_{i' \in \bar{V}_m \cup \bar{V}_g \cup \bar{V}'_{M_1} \cup \bar{V}''_M \cup \bar{V}_G} \ell_{i'} y_{i'i} +$$

$$\sum_{i' \in \bar{V}'_{M_2}} (-M + p_i + p_j) \cdot z^b_{i'} + \sum_{i' \in \bar{V}'_{M_2}} \bar{w}_{i'}\}$$

$$\text{s. t.} \quad \sum_{i \in Pred(i')} y_{ii'} - \sum_{i \in Succ(i')} y_{i'i} \leq 0 \quad (i' \in \bar{V}) \tag{5.38}$$

$$\sum_{i' \in Pred(i)} y_{i'i} - \sum_{i' \in Succ(i)} y_{ii'} \leq 0 \quad (i \in V \setminus \{0, n+1\}) \tag{5.39}$$

$$\sum_{i' \in Pred(n+1)} y_{i'n+1} \leq 1 \tag{5.40}$$

$$M \cdot y_{i'i} + \bar{w}_{i'} \leq 0 \quad (i' \in \bar{V}'_{M_2}; i \in Succ(i^{\prime)})) \tag{5.41}$$

$$y_{ii'} \text{ free} \quad (i \in V; i' \in Succ(i)) \tag{5.42}$$

$$y_{i'i} \geq 0 \quad (i' \in \bar{V}_m \cup \bar{V}_g; i \in Succ(i')) \tag{5.43}$$

$$y_{i'i} \leq 0 \quad (i' \in \bar{V}_{M_1} \cup \bar{V}''_M \cup \bar{V}_G; i \in Succ(i')) \tag{5.44}$$

$$y_{ii'} \text{ free} \quad (i' \in \bar{V}'_{M_2}; i \in Succ(i')) \tag{5.45}$$

$$\bar{w}_{i'} \leq 0 \quad (i' \in \bar{V}'_{M_2}) \tag{5.46}$$

$$z^b_{i'} \geq 0 \quad (i' \in \bar{V}'_{M_2}) \tag{5.47}$$

Since M is positive and arbitrarily large and $z^b_{i'} \geq 0$, the objective function is maximized imposing that $z^{b*}_{i'} = 0$, where $z^{b*}_{i'} = 0$ is the optimal value of $z^b_{i'}$. Moreover, for the complementary slackness conditions referred to constraint (5.41) we have that:

$$w^*_{i'}(-M \cdot y^*_{i'i} - \bar{w}^*_{i'}) = 0 \quad (i' \in \bar{V}'_{M_2}; i \in Succ(i')) \tag{5.48}$$

For node i' we can have either $w^*_{i'} = 0$ or $w^*_{i'} = 1$. Let us examine these two situations separately.

Case a: $w^*_{i'} = 1$

By (5.48) we have that

$$w^*_{i'} = -M \cdot y^*_{i'i} \quad (i' \in \bar{V}'_{M_2}; i \in Succ(i')) \tag{5.49}$$

By constraint (5.46) we have that:

$$y^*_{i'i} \geq 0 \quad (i' \in \bar{V}'_{M_2}; i \in Succ(i')) \tag{5.50}$$

Substituting (5.49) and (5.50) in the mathematical formulation we have:

$$\text{Max.} \{\sum_{i \in V} \sum_{i' \in Succ(i)} p_i y_{ii'} + \sum_{i' \in \bar{V}} \sum_{i \in Succ(i')} \ell_{i'} y_{i'i}\}$$

$$\text{where } \ell_{i'} = -M, \ (i' \in \bar{V}'_{M_2})$$

$$\text{s. t.} \quad \sum_{i \in Pred(i')} y_{ii'} - \sum_{i \in Succ(i')} y_{i'i} \leq 0 \quad (i' \in \bar{V}) \tag{5.51}$$

$$\sum_{i' \in Pred(i)} y_{i'i} - \sum_{i' \in Succ(i)} y_{ii'} \leq 0 \quad (i \in V \setminus \{0, n+1\}) \tag{5.52}$$

$$\sum_{i' \in Pred(n+1)} y_{i'n+1} \leq 1 \tag{5.53}$$

$$y_{ii'} \text{ free} \quad (i \in V; i' \in Succ(i)) \tag{5.54}$$

$$y_{i'i} \geq 0 \quad (i' \in \bar{V}_m \cup \bar{V}'_{M_2} \cup \bar{V}_g; i \in Succ(i')) \tag{5.55}$$

$$y_{i'i} \leq 0 \quad (i' \in \bar{V}_{M_1} \cup \bar{V}''_M \cup \bar{V}_G; i \in Succ(i')) \tag{5.56}$$

Case b: $w^*_{i'} = 0$

By the complementary slackness property on constraint (5.23) we have that

$$w^*_{i'}(1 - w^*_{i'}) = 0 \quad (i' \in \bar{V}'_{M_2})$$

by which we obtain that

$$w^*_{i'} = 0 \quad (i' \in \bar{V}'_{M_2})$$

This implies that, by constraint (5.30),

$$z_{i'}^{a^*} \leq z_{i'}^{b^*} \quad (i' \in \bar{V}'_{M_2})$$

Since the objective function is maximized by imposing $z_{i'}^{b^*} = 0$, we have that also [see constraint (5.36)]

$$z_{i'}^{a^*} = 0 \quad (i' \in \bar{V}'_{M_2})$$

The dual problem then transforms into:

$$\text{Max.} \{\sum_{i \in V} \sum_{i' \in Succ(i)} p_i y_{ii'} + \sum_{i' \in \bar{V}_m \cup \bar{V}_g} \ell_{i'} z_{i'} + \sum_{i' \in \bar{V}'_{M_1} \cup \bar{V}''_M \cup \bar{V}_G} \ell_{i'} z_{i'}\}$$

$$\text{s. t.} \quad \sum_{i \in Pred(i')} y_{ii'} - \sum_{i \in Succ(i')} y_{i'i} \leq 0 \quad (i' \in \bar{V}) \tag{5.57}$$

$$\sum_{i' \in Pred(i)} y_{i'i} - \sum_{i' \in Succ(i)} y_{ii'} \leq 0 \quad (i \in V \setminus \{0, n+1\}) \tag{5.58}$$

$$\sum_{i' \in Pred(n+1)} y_{i'n+1} \leq 1 \tag{5.59}$$

$$z_{i'} - y_{i'i} = 0 \quad (i' \in \bar{V}_m \cup \bar{V}'_{M_1} \cup \bar{V}''_M \cup \bar{V}_g \cup \bar{V}_G; i \in Succ(i')) \tag{5.60}$$

$$-y_{i'i} = 0 \quad (i' \in \bar{V}'_{M_2}; i \in Succ(i')) \tag{5.61}$$

$$y_{ii'} \text{ free} \quad (i \in V; i' \in Succ(i)) \tag{5.62}$$

$$y_{i'i} \text{ free} \quad (i' \in \bar{V}; i \in Succ(i')) \tag{5.63}$$

$$z_{i'} \geq 0 \quad (i' \in \bar{V}_m \cup \bar{V}_g) \tag{5.64}$$

$$z_{i'} \leq 0 \quad (i' \in \bar{V}'_{M_1} \cup \bar{V}''_M \cup \bar{V}_G) \tag{5.65}$$

Substituting now constraints (5.60) in the objective function we have the following formulation

$$\text{Max.} \{\sum_{i \in V} \sum_{i' \in Succ(i)} p_i y_{ii'} + \sum_{i' \in \bar{V}} \sum_{i \in Succ(i')} \ell_{i'} y_{i'i}\}$$

where we can assign $\ell_{i'} = -M$, $(i' \in \bar{V}'_{M_2})$, being $y_{i'i} = 0$, $(i' \in \bar{V}'_{M_2}; i \in Succ(i'))$, by (5.61)

$$\text{s. t.} \quad \sum_{i \in Pred(i')} y_{ii'} - \sum_{i \in Succ(i')} y_{i'i} \leq 0 \quad (i' \in \bar{V}) \tag{5.66}$$

$$\sum_{i' \in Pred(i)} y_{i'i} - \sum_{i' \in Succ(i)} y_{ii'} \leq 0 \quad (i \in V \setminus \{0, n+1\}) \tag{5.67}$$

$$\sum_{i' \in Pred(n+1)} y_{i'n+1} \leq 1 \tag{5.68}$$

$$-y_{i'i} = 0 \quad (i' \in \bar{V}'_{M_2}; i \in Succ(i')) \tag{5.69}$$

$$y_{ii'} \text{ free} \quad (i \in V; i' \in Succ(i)) \tag{5.70}$$

$$y_{i'i} \geq 0 \quad (i' \in \bar{V}_m \cup \bar{V}_g; i \in Succ(i')) \tag{5.71}$$

$$y_{i'i} \leq 0 \quad (i' \in \bar{V}_{M_1} \cup \bar{V}''_M \cup \bar{V}_G; i \in Succ(i')) \tag{5.72}$$

We note that in both of the two cases analyzed we achieved the same mathematical formulation, meaning that either $w^*_{i'} = 0$ or $w^*_{i'} = 1$ we can use the same solution strategy. Moreover, we note that this mathematical model obtained is polynomially solvable in $\mathcal{O}(m)$ by means of a dynamic programming approach since the formulation has at most two variable per constraint (see Hochbaum and Naor 1994). Furthermore, this formulation is the same as that found in absence of resource constraints by Bianco and Caramia (2010). Therefore, there holds the property demonstrated in that paper, i.e., the optimal solution to the relaxed RCPSP with GPRs, and therefore the lower bound of the original problem, is an augmenting path of maximum length on the described AoN network where unit capacities have been installed on arcs.

5.3.1.4 Computational Results

In this section, we show the results of our experimentation. Our algorithm and the other lower-bound algorithms used for comparison have been implemented in the C language. The machine used for the experiments is a Pentium IV PC with a 3 GHz processor and 2 Gb RAM. All the computing times are negligible (i.e., less than 1 s) and therefore are omitted in the presentation of the results. In order to assess the quality of our lower bound with respect to lower bound algorithms from the state of the art, we have experimented the proposed approach on networks with 100 and 500 activities. These networks have been generated at random with the following parameters:

- Maximum number of initial (without predecessors) activities: 10;
- Maximum number of terminal (without successors) activities: 10;
- Maximum indegree of activities: 5;
- Maximum time lag constraints ranging from 0 to 20 % of the total number of arcs;

We generated 90 problems for each network size, as done also in the RCPSP-max library, and reported our results in Figs. 5.12 and 5.13.

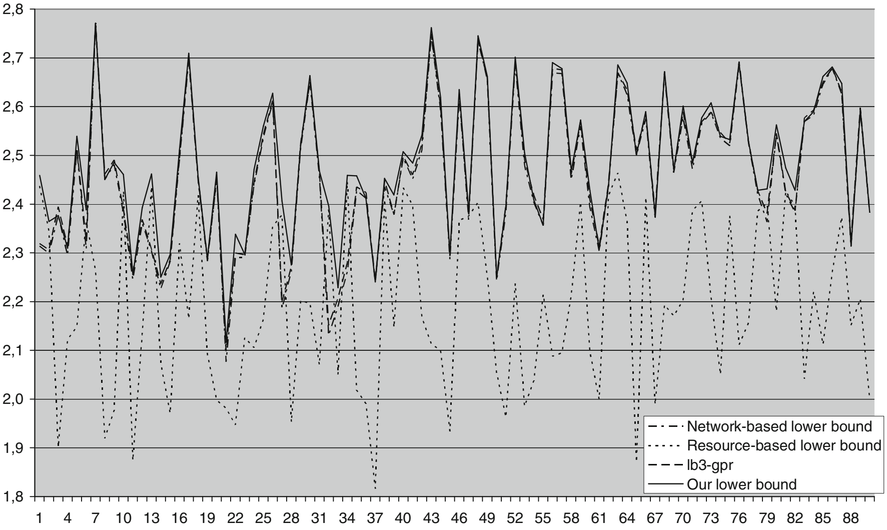


Fig. 5.12 Comparison among our lower bound (Our_LB), the network-based lower bound (LB1), the resource-based one (LB2), and lb3-gpr (LB3) on 100 activity networks

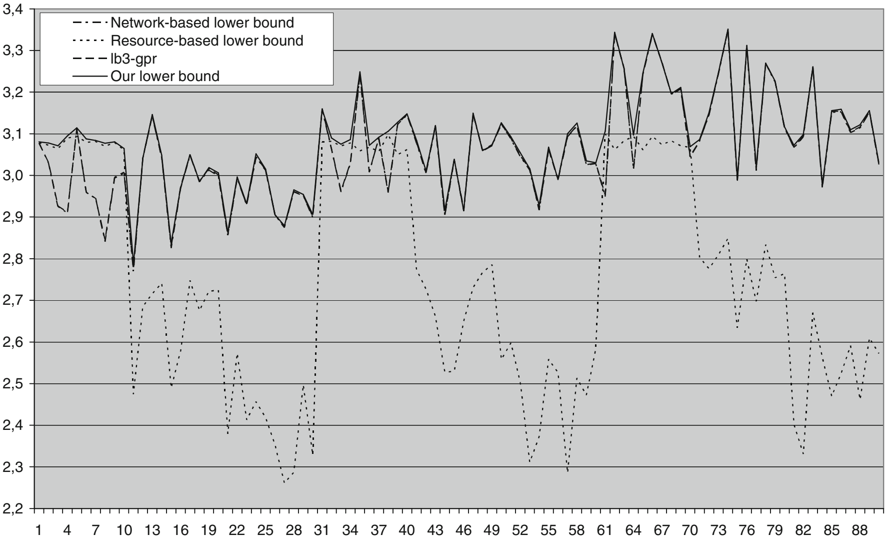


Fig. 5.13 Comparison among our lower bound (Our_LB), the network-based lower bound (LB1), the resource-based one (LB2), and lb3-gpr (LB3) on 500 activity networks

The chart in Fig. 5.12 shows the performance of our lower bound, denoted in the following with Our_LB, compared to the network based lower bound, denoted with LB1, the resource-based lower bound, denoted with LB2, and the lb3-gpr lower bound, denoted with LB3, on 100 activity networks. Figure 5.13 reports the same comparison on 500 activity networks. Values in the y-axis are reported as base ten logarithms. Analyzing in detail the behaviour of LB1, LB2, and LB3, we notice that, as one can expect, LB2 is quite sensitive to the resource-strength factor.[1] Indeed, looking for instance at the chart in Fig. 5.13 (where this appears clearer), we observe that for instances ranging from 1 to 10, from 31 to 40, and from 61 to 70, where the resource strength parameter is zero, LB2 outperforms LB1 and LB3. For the other instance classes, whose resource strength parameter ranges from 0.25 to 0.5, we have that the values of LB1 and LB3 tend to overlap (with LB3 having a slightly better behaviour) and dominate LB2. Indeed, from a general viewpoint, while it appears that LB1 and LB3 tend to outperform LB2 on projects with 100 activities, it happens that there is not a striking dominance among LB1, LB2, and LB3 when 500 activities are considered. Analysing the behaviour of Our_LB, one can note that it is robust to the different input instances, unlike the three competing lower bounds. In fact, Our_LB is able to outperform LB1, LB2, and LB3 on all the tested instances. In particular, by analysing the computational results, we note that, on 100 activity networks, Our_LB improves the best value among LB1, LB2, and LB3 by 5.2 %, while on projects with 500 activities the improvement is 4.4 %.

5.3.2 An Exact Algorithm for RCPSP with GPRs

In the following, a new mathematical formulation of the RCPSP with GPRs in terms of mixed integer programming is presented. Moreover, a branch-and-bound algorithm exploiting both the latter formulation and a Lagrangian relaxation based lower bound is described (Bianco and Caramia 2012). In particular, the proposed lower bound is based on a fast method developed in Bianco and Caramia (2011b) to compute an estimate of the optimal Lagrangian multipliers. This estimate is then used to feed a standard subgradient algorithm. An extensive experimentation, and a comparison with both known lower bounds and the exact algorithms by De Reyck and Herroelen (1998), Schwindt (1998), Fest et al. (1999), Dorndorf et al. (2000), show the performance of the proposed algorithm.

[1] Given resource k, let r_k^{min} be the maximum usage of this resource by a single activity, that is $r_k^{min} = \max_{i \in V} r_{ik}$. Let r_k^{max} denote the peak demand of resource k in the earliest start schedule with infinite resource capacity. The resource strength of resource k is thus defined as $RS_k = \frac{R_k - r_k^{min}}{r_k^{max} - r_k^{min}}$ (Kolish et al. 1995).

5.3.2.1 The Mathematical Model

In this section, a novel formulation for the project scheduling problem with GPRs, scarce resources and minimum makespan objective is described. This formulation will be used by the enumerative scheme presented in the next section.

Differently from the standard GPRs formulation present in the literature, mentioned in Sect. 5.2.1, we assume that there is a planning horizon within which all the activities have to be carried out. In particular, we denote such a planning horizon as $[0, T)$, where T is an upper bound on the minimum project makespan. Moreover, we assume, without loss of generality, that the time horizon is discretized into T unit-width time periods $[0, 1), [1, 2), \ldots, [T-1, T)$, indexed by $t = 1, \ldots, T$. Let us define the following parameters:

- K, the number of renewable (continuously divisible) resources, each one available in an amount of R_k units, with $k = 1, \ldots, K$;
- $\bar{r}_{ik}$, the overall amount of units of resource k necessary to carry out activity i; it is given by the (renewable) resource request per period times the duration of the activity;
- p_i, the duration of activity i.

Furthermore, according to the previous hypotheses on time discretization, let us consider the following decision variables:

- ζ_{it}, the percentage of activity i executed within the end of time period t.
- ξ_{it}, a binary variable that assumes value 1 if activity i has started within the beginning of a time period $\tau \leq t$, and assumes value 0 otherwise.
- ξ'_{it}, a binary variable that assumes value 1 if activity i has finished within the end a time period $\tau \leq t$, and assumes value 0 otherwise.

Since the completion time of an activity $i \in V$ can be expressed as $\left(T - \sum_{t=1}^{T} \xi'_{it} + 1\right)$, the objective function can be written as

$$\text{Min.} \left\{ \text{Max.}_{i \in V} \left\{ T - \sum_{t=1}^{T} \xi'_{it} + 1 \right\} \right\}$$

and the constraints can be modelled as follows:

$$\sum_{\tau=1}^{T} \xi_{i\tau} \geq \sum_{\tau=1}^{T} \xi_{j\tau} + \delta_{ij} \quad ((i,j) \in E) \tag{5.73}$$

$$\zeta_{it} - \zeta_{i,t-1} = \frac{1}{p_i}(\xi_{it} - \xi'_{i,t-1}) \quad (i \in V \setminus \{0, n+1\}; t = 1, \ldots, T) \tag{5.74}$$

$$\xi_{it} \leq \xi_{i,t+1} \quad (i \in V; t = 1, \ldots, T) \tag{5.75}$$

$$\xi'_{i,t-1} \leq \xi'_{it} \quad (i \in V; t = 1, \ldots, T) \tag{5.76}$$

$$\xi_{iT} = \xi'_{iT} = \zeta_{iT} = 1 \quad (i \in V \setminus \{0, n+1\}) \tag{5.77}$$

$$\xi_{i0} = \xi'_{i0} = \zeta_{i0} = 0 \quad (i \in V \setminus \{0, n+1\}) \tag{5.78}$$

$$\xi'_{it} \leq \zeta_{it} \leq \xi_{it} \quad (i \in V \setminus \{0, n+1\}; t = 1, \ldots, T) \tag{5.79}$$

$$\xi'_{00} = \xi_{01} = \xi'_{n+1,T} = \xi_{n+1,T+1} = 1 \tag{5.80}$$

$$\sum_{i=1}^{|V|} \bar{r}_{ik}(\zeta_{it} - \zeta_{i,t-1}) \leq R_k \quad (k = 1, \ldots, K; t = 1, \ldots, T) \tag{5.81}$$

$$\xi_{it} \in \{0, 1\} \quad (i \in V; t = 1, \ldots, T+1) \tag{5.82}$$

$$\xi'_{it} \in \{0, 1\} \quad (i \in V; t = 0, \ldots, T) \tag{5.83}$$

$$\zeta_{it} \geq 0 \quad (i \in V; t = 1, \ldots, T) \tag{5.84}$$

By posing $\varDelta = \max_{i \in V}\left(T - \sum_{t=1}^{T} \xi'_{it} + 1\right)$, this problem can be rewritten as:

$$\begin{aligned}
&\text{Min. } \varDelta \\
&\text{s. t.} \quad \sum_{\tau=1}^{T} \xi_{i\tau} \geq \sum_{\tau=1}^{T} \xi_{j\tau} + \delta_{ij} \quad ((i, j) \in E) \\
&\qquad \ldots \\
&\qquad \ldots \\
&\qquad \zeta_{it} \geq 0 \quad (i \in V; t = 1, \ldots, T) \\
&\qquad \varDelta \geq \left(T - \sum_{t=1}^{T} \xi'_{it} + 1\right) \quad (i \in V)
\end{aligned} \tag{5.85}$$

Constraints (5.73) model *Start-to-Start* precedence constraints with minimum time lags δ_{ij}, $(i, j) \in E$. In fact, since the starting time of activity i is $(T - \sum_{\tau=1}^{T} \xi_{i\tau})$ and, similarly, the starting time of activity j is $(T - \sum_{\tau=1}^{T} \xi_{j\tau})$, the generic *Start-to-Start* precedence constraint between $(i, j) \in E$ can be written as $T - \sum_{\tau=1}^{T} \xi_{j\tau} \geq T - \sum_{\tau=1}^{T} \xi_{i\tau} + \delta_{ij}$. By simplifying terms in the previous equation, one obtains constraint (5.73). Constraints (5.74) regulate the total amount processed of an activity $i \in V \setminus \{0, n+1\}$ over time. We note that either this value is 0 or it is equal to $\frac{1}{p_i}$. This constraint imposes also that pre-emption is not allowed. Indeed if activity i starts at time t then until its finishing time it must be processed without interruption. Constraints (5.75) imply that if an activity $i \in V$ is started at time t, then variable $\xi_{i\tau} = 1$ for every $\tau \geq t$, and, on the contrary, if activity i is not started at time t, $\xi_{i\tau} = 0$ for every $\tau \leq t$. Constraints (5.76) are the same as constraints (5.75) when finishing times are concerned. Constraints (5.77) say that every activity $i \in V$ must start and finish within the planning horizon. Constraints (5.78) represent the initialization conditions for variables $\xi_{it}, \xi'_{it}, \zeta_{it}$ when $t = 0$. Constraints (5.79) force ζ_{it} to be zero if $\xi_{it} = 0$, and ξ'_{it} to be zero if $\zeta_{it} < 1$. Constraints (5.80) are boundary conditions for dummy activities. Resource constraints are represented by relations (5.81). Constraints (5.82), (5.83), and (5.84) limit the range of variability of the variables.

In order to improve the comprehension of the model, and, in particular, the role played by the variables, we consider a simple example of a project with a time horizon of three periods and a single activity i with a unitary duration. Assume that activity i starts and finishes in time period 2. Then, the values assumed by the variables are: $\xi_{i1} = \xi'_{i1} = 0$; $\xi_{i2} = \xi_{i3} = 1$; $\xi'_{i2} = \xi'_{i3} = 1$.

Referring to constraints (5.74) we have: $\zeta_{i1} - \zeta_{i0} = \xi_{i1} - \xi'_{i0} = 0$ which implies that $\zeta_{i1} = 0$ by constraints (5.79). Similarly, we have: $\zeta_{i2} - \zeta_{i1} = \xi_{i2} - \xi'_{i1} = 1$ which implies that $\zeta_{i2} = 1$. The completion time of activity i is $\big(3 - (\xi'_{i1} + \xi'_{i2} + \xi'_{i3}) + 1\big) = 2$.

5.3.2.2 The Exact Solution Algorithm Description

In this section, the search tree structure, a Lagrangian based lower bound, the branching rule adopted and some node fathoming rules, are illustrated.

The Search Tree Structure

The enumerative algorithm proposed exploits both the mathematical formulation presented in the previous section and branch-and-bound rules.

The root node α_0 of the search tree is associated with the whole problem denoted with P_0. With P_0 we associate the time horizon T (an upper bound on the minimum makespan) and a lower bound on the minimum makespan based on the Lagrangian relaxation of the resource constraints (5.81). The time horizon T may be easily computed by summing up all the activity durations and the positive time lags δ_{ij}, with $(i, j) \in E$. This is clearly an upper bound because the temporal relationships are of the *Start-to-Start* type with minimum time lags.

The search tree is structured in such a way that each level is associated with an activity in V, which means that the tree will have at most $|V|$ levels.

Denoting with i the activity associated with the first level of the search tree (see Fig. 5.14), at most T subproblems can be generated in such a level from the root. Let us denote with P_{1t}, with $t \in \{1, \dots, T\}$, the generic subproblem associated with level 1, and with α_{1t} the corresponding node of the search tree. Each subproblem P_{it} is associated with a time slot $t = 1, \dots, T$ that represents the latest time at which activity i can start, and is obtained from P_0 (its parent) by imposing $\xi_{it} = 1$ (the branching variable). The generic subproblem P_{it} is associated with a node α_{it} in the search tree.

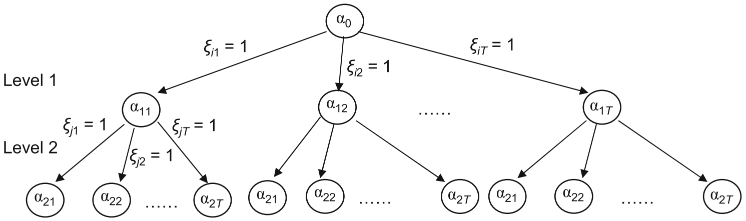


Fig. 5.14 Two levels of the branch-and-bound tree. i and j are the activities associated with level 1 and level 2, respectively

The same analysis described for the first level can be applied to every level of the search tree, i.e., from a subproblem $P_{i\tau}$ at level i we can generate T subproblems $P_{i+1,t}$ at level $i+1$, with $t = 1, \ldots, T$, each obtained by $P_{i\tau}$ by fixing $\xi_{jt} = 1$, where j is the activity associated with level $i+1$.

The Lagrangian Relaxation Based Lower Bound

The lower bound used at each node of the search tree is based on the Lagrangian relaxation of the resource constraints. Let λ be the vector of the Langrangian multipliers, with components $\lambda_{kt}, k = 1, \ldots, K, t = 1, \ldots, T$. The resulting Lagrangian problem P_{LaR} is as follows:

$$\text{Min.} \left\{ \Delta - \sum_{k=1}^{K} \sum_{t=1}^{T} \lambda_{kt} \left[R_k - \sum_{i=1}^{|V|} \bar{r}_{ik}(\zeta_{it} - \zeta_{i,t-1}) \right] \right\}$$

$$\text{s. t.} \quad \sum_{\tau=1}^{T} \xi_{i\tau} \geq \sum_{\tau=1}^{T} \xi_{j\tau} + \delta_{ij} \quad ((i,j) \in E)$$

$$\ldots$$

$$\xi'_{it} \leq \zeta_{it} \leq \xi_{it} \quad (i \in V \setminus \{0, n+1\}; t = 1, \ldots, T)$$

$$\xi'_{00} = \xi_{01} = \xi'_{n+1,T} = \xi_{n+1,T+1} = 1$$

$$\xi_{it} \in \{0,1\} \quad (i \in V; t = 1, \ldots, T+1)$$

$$\xi'_{it} \in \{0,1\} \quad (i \in V; t = 0, \ldots, T)$$

$$\zeta_{it} \geq 0 \quad (i \in V; t = 1, \ldots, T)$$

$$\Delta \geq \left(T - \sum_{t=1}^{T} \xi'_{it} + 1 \right) \quad (i \in V)$$

where for non-negative values of the λ_{kt} multipliers, with $k = 1, \ldots, K, t = 1, \ldots, T$, the value of an optimal solution of P_{LaR} provides a lower bound to the optimal solution of problem P_0. The dual Lagrangian problem is to find the vector λ^* maximizing P_{LaR}, that is

$$\begin{aligned}
\text{Max.}_{\lambda_{kt}} \quad & \left\{ \text{Min.} \left\{ \varDelta - \sum_{k=1}^{K} \sum_{t=1}^{T} \lambda_{kt} \left[R_k - \sum_{i=1}^{|V|} \bar{r}_{ik} (\zeta_{it} - \zeta_{i,t-1}) \right] \right\} \right\} \\
\text{s. t.} \quad & \sum_{\tau=1}^{T} \xi_{i\tau} \geq \sum_{\tau=1}^{T} \xi_{j\tau} + \delta_{ij} \quad ((i, j) \in E) \\
& \ldots \\
& \xi'_{it} \leq \zeta_{it} \leq \xi_{it} \quad (i \in V \setminus \{0, n+1\}; t = 1, \ldots, T) \\
& \xi'_{00} = \xi_{01} = \xi'_{n+1,T} = \xi_{n+1,T+1} = 1 \\
& \xi_{it} \in \{0, 1\} \quad (i \in V; t = 1, \ldots, T+1) \\
& \xi'_{it} \in \{0, 1\} \quad (i \in V; t = 0, \ldots, T) \\
& \zeta_{it} \geq 0 \quad (i \in V; t = 1, \ldots, T) \\
& \varDelta \geq \left(T - \sum_{t=1}^{T} \xi'_{it} + 1 \right) \quad (i \in V) \\
& \lambda_{kt} \geq 0 \quad (k = 1, \ldots, K; t = 1, \ldots, T)
\end{aligned} \tag{5.86}$$

Usually λ^* is computed by means of a subgradient algorithm which, in general, is very time consuming. This method, in fact, starts with $\lambda = 0$ and iteratively computes the Lagrangian multipliers until a stopping criterion is met. The values so obtained are either the optimal ones or an estimate of λ^*. In the following, we apply a method to estimate λ^*, proposed in Bianco and Caramia (2011b), that was experimented to be very fast and effective. In the same paper it was proved that the Mixed Integer Linear Program (MILP) presented above can be solved in pseudo-polynomial time. However, since in the latter paper the project scheduling problem studied is not exactly the same as the RCPSP with GPRs, the estimate of λ^* found is used to initialize the execution of a very limited number of iterations of a subgradient algorithm. This is done to verify if it is possible to improve the estimate of the optimal Lagrangian multipliers.

For the sake of completeness, in the following, the different steps of the aforementioned method to estimate λ^* are reported.

By previous constraints (5.79) and (5.85) we have $\zeta_{it} \geq \xi'_{it}$ and $\varDelta \geq (T - \sum_{t=1}^{T} \xi'_{it} + 1)$; therefore,

$$\varDelta \geq (T - \sum_{t=1}^{T} \zeta_{it} + 1)$$

and, hence,

$$\sum_{t=1}^{T} \zeta_{it} \geq (T - \Delta + 1) \tag{5.87}$$

If we multiply both the left- and the right-hand sides of (5.87) by $\lambda_{kt}\bar{r}_{ik}$, and sum up with respect to i, k, t, we get the following relation:

$$\sum_{i=1}^{|V|}\sum_{k=1}^{K}\sum_{t=1}^{T} \lambda_{kt}\bar{r}_{ik}\zeta_{it} \geq (T - \Delta + 1)\sum_{i=1}^{|V|}\sum_{k=1}^{K}\sum_{t=1}^{T} \lambda_{kt}\bar{r}_{ik} \tag{5.88}$$

Rewriting the objective function accordingly, we have:

$$\begin{aligned}
&\text{Max.}_{\lambda_{kt}} \left\{ -\sum_{k=1}^{K}\sum_{t=1}^{T} \lambda_{kt}R_k + \text{Min.}\left[\Delta + \sum_{i=1}^{|V|}\sum_{k=1}^{K}\sum_{t=1}^{T} \lambda_{kt}\bar{r}_{ik}\zeta_{it} \right.\right.\\
&\qquad\left.\left. - \sum_{i=1}^{|V|}\sum_{k=1}^{K}\sum_{t=1}^{T} \lambda_{kt}\bar{r}_{ik}\zeta_{i,t-1} \right]\right\} \geq \\
&\text{Max.}_{\lambda_{kt}} \left\{ -\sum_{k=1}^{K}\sum_{t=1}^{T} \lambda_{kt}R_k + \text{Min.}\left[\Delta + (T - \Delta + 1)\sum_{i=1}^{|V|}\sum_{k=1}^{K}\sum_{t=1}^{T} \lambda_{kt}\bar{r}_{ik} \right.\right.\\
&\qquad\left.\left. - \sum_{i=1}^{|V|}\sum_{k=1}^{K}\sum_{t=1}^{T} \lambda_{kt}\bar{r}_{ik} \right]\right\}
\end{aligned}$$

where the second expression is obtained by exploiting (5.88) in the triple summation with the positive sign of the Min. term, and by posing $\zeta_{i,t-1} = 1$ in the triple summation with the negative sign.

Now we are in the position to find an estimate $\tilde{\lambda}_{kt}$ of the optimal Lagrangian multipliers λ^{*}_{kt} by solving the following problem:

$$\begin{aligned}
&\text{Max.}_{\lambda_{kt}} \left\{ -\sum_{k=1}^{K}\sum_{t=1}^{T} \lambda_{kt}R_k + \text{Min.}\left[\Delta \left(1 - \sum_{i=1}^{|V|}\sum_{k=1}^{K}\sum_{t=1}^{T} \lambda_{kt}\bar{r}_{ik} \right) \right.\right.\\
&\qquad\left.\left. + \sum_{i=1}^{|V|}\sum_{k=1}^{K}\sum_{t=1}^{T} T\lambda_{kt}\bar{r}_{ik} \right]\right\}\\
&\text{s.t. } \Delta \geq 0\\
&\lambda_{kt} \geq 0 \quad (k = 1, \ldots, K; t = 1, \ldots, T)
\end{aligned}$$

In fact, noting that the above problem admits a bounded solution if

$$\left(1 - \sum_{i=1}^{|V|} \sum_{k=1}^{K} \sum_{t=1}^{T} \lambda_{kt} \bar{r}_{ik}\right) \geq 0$$

$\tilde{\lambda}_{kt}$ multipliers are obtained by solving the following linear program:

$$\begin{aligned} \text{Max.}_{\lambda_{kt}} \quad & \sum_{k=1}^{K} \sum_{t=1}^{T} \lambda_{kt} \left[\sum_{i=1}^{|V|} T \bar{r}_{ik} - R_k \right] \\ \text{s. t.} \quad & \sum_{i=1}^{|V|} \sum_{k=1}^{K} \sum_{t=1}^{T} \lambda_{kt} \bar{r}_{ik} \leq 1 \\ & \lambda_{kt} \geq 0 \quad (k = 1, \ldots, K, t = 1, \ldots, T) \end{aligned}$$

These $\tilde{\lambda}_{kt}$ values are used as starting Lagrangian multipliers for the subgradient algorithm (Held and Karp 1970) to possibly improve the quality of the lower bound. Let $z(P_{LaR}(\tilde{\lambda}))$ be the lower bound associated with the $\tilde{\lambda}$ values found by the above linear program. The subgradient algorithm used is the following:

```
Step 1.  Let $\alpha := 2$.
    Let $\lambda := \tilde{\lambda}$ the initial Lagrangian multipliers.
    Let $LB := z(P_{LaR}(\lambda))$.
    Let $z(UB)$ the value of a heuristic solution to the problem.
Step 2.  Let $(\zeta^*_{it}, \xi^*_{it}, \xi'^*_{it})$ be the optimal solution to $P_{LaR}$ of value $z(P_{LaR}(\lambda))$.
    Let $LB := \max[LB, z(P_{LaR}(\lambda))]$.
    **if** $\sum_{i=1}^{|V|} \bar{r}_{ik}(\zeta^*_{it} - \zeta^*_{i,t-1}) \leq R_k, \quad (k = 1 \ldots, K; t = 1, \ldots, T)$
    and $\lambda_{kt} \cdot (R_k - \sum_{i=1}^{|V|} \bar{r}_{ik}(\zeta^*_{it} - \zeta^*_{i,t-1})) = 0, \quad (k =, 1 \ldots, K; t = 1, \ldots, T)$ **then**
    $(\zeta^*_{it}, \xi^*_{it}, \xi'^*_{it})$ is the optimal solution to the original problem and **stop**.
Step 3.  Let the vector $sg$ of components $sg_{kt}$, $(k = 1, \ldots, K; t = 1, \ldots, T)$, be a
    subgradient associated with the relaxed constraints where
    $sg_{kt} := \sum_{i=1}^{|V|} \bar{r}_{ik}(\zeta^*_{it} - \zeta^*_{i,t-1}) - R_k$, $(k = 1 \ldots, K; t = 1, \ldots, T)$.
Step 4.  Let $\Theta := \alpha \frac{[z(UB) - z(P_{LaR}(\lambda))]}{\sum_{k=1}^{K} \sum_{t=1}^{T} sg^2_{kt}}$.
Step 5.  Update the Lagrangian penalties as follows
    $\lambda_{kt} := \max[0, \lambda_{kt} + \Theta sg_{kt}], \quad (k = 1, \ldots, K; t = 1, \ldots, T)$.
    **go to** Step 2.
```

A similar approach can be found in Möhring et al. (1999, 2003). In the latter paper the authors provide a Lagrangian relaxation of a time-indexed integer programming formulation and relaxation-based list scheduling, along with an idea borrowed from approximation algorithms for machine scheduling problems. The efficiency of the algorithm results from the fact that the relaxed problem can be solved by computing a minimum cut in a proper directed graph.

The Branching Rule, the Bounding Phase and the Node Fathoming Rules

We first describe the branching rule adopted by the branch-and-bound algorithm. The idea stems from the observation that a GPR between i, j is such that activity i constraints the starting time of activity j. By representing all the GPRs in a graph, where an arc (i, j) exists if there is a GPR between i and j, the higher the number of successors of an activity, the higher should be the degree of constrainedness that this activity may impose to its successors. Therefore, the rule we adopt is that of assigning, to each level of the search tree, the activity with the greatest number of successors (ties are broken arbitrarily).

Let us now consider the bounding phase, with the corresponding fathoming rules. Each subproblem P_{it} associated with the tree node α_{it} (see Fig. 5.14) undergoes a bounding phase in which the Lagrangian lower bound on the minimum makespan is computed as done for the root node. The calculation of the Lagrangian multipliers and the successive solution of the MILP associated with the Lagrangian problem is done by means of a commercial solver. Here we can have four alternative outcomes:

1. the time slot t of subproblem P_{it} is such that $t \geq UB^*$ where UB^* is the best upper bound found so far;
2. the mathematical program associated with the Lagrangian lower bound is infeasible, i.e., some GPRs cannot be obeyed;
3. the solution of the Lagrangian relaxation is not feasible with respect to some resource constraints;
4. the latter solution respects all the resource constraints.

Clearly, in the case 1, the tree is pruned and a backtracking phase is executed. Also in the case 2, the subtree generating from problem P_{it} is fathomed since a feasible solution cannot be found, with a consequent backtracking to the previous tree level; in the case 3, the Lagrangian relaxation solution value $f(P_{it})_{LaR}$, which is a lower bound for subproblem P_{it}, is compared to the best upper bound UB^* found so far; if $f(P_{it})_{LaR} \geq UB^*$ then the tree is pruned again (with a consequent backtracking), otherwise the search is continued in a depth first search strategy.

In the last occurrence, i.e., in the case 4, the solution value $f(P_{it})$, obtained by entering the $\zeta_{it}, \xi_{it}, \xi'_{it}$ values obtained by the Lagrangian relaxation in P_{it}, is an upper bound for the latter subproblem and therefore it is an upper bound for the whole problem P_0. Now, if this solution to P_{it} satisfies the complementary slackness conditions, it is the optimal solution for P_{it} and the tree can be pruned, possibly updating UB^* to $f(P_{it})$ if the former is greater than the latter. If the solution is not optimal for P_{it}, then the search continues in a depth first search strategy possibly updating UB^* to $f(P_{it})$ if $UB^* > f(P_{it})$.

5.3.3 Computational Results

5.3.3.1 Implementation Details

The implementation of the algorithm has been carried out in the C language; the mathematical formulations presented above have been implemented in the AMPL language and solved by means of the commercial solver CPLEX, version 8.0.0. The machine used for the experiments is a PC Core Duo with a 1.6 GHz Intel Centrino Processor and 1 GB RAM.

The proposed approach has been experimented on networks with 30, 50, 80, and 100 activities. These networks have been generated at random with the following parameters:

- maximum number of initial (without predecessors) activities: 10;
- maximum number of terminal (without successors) activities: 10;
- maximum indegree of activities: 5;
- maximum time lag constraints equal to 0, 10, and 20 % of the total number of GPRs;
- number K of renewable resources equal to 5;
- amount R_k of resource availability per period for each resource $k = 1, \dots, K$ equal to 4;
- request per period r_{ik} of resource $k = 1, \dots, K$ for every activity $i \in V$ assigned uniformly at random from 1 to 3;
- values δ_{ij} assigned uniformly at random in the range $[-10, 10]$.

Ninety problems for each network size have been generated as done in the RCPSP-max library (see the web site http://www.wiwi.tu-clausthal.de/abteilungen/produktion/forschung/schwerpunkte/project-generator).

5.3.3.2 Analysis of the Results

An extensive experimentation can be found in the appendix of the paper by Bianco and Caramia (2012). Here only a comparison in terms of average performance between the De Reyck and Herroelen, and Bianco and Caramia algorithms with respect to different lower bounds (i.e., network-based, resource-based, Bianco and Caramia 2011a denoted as BC11, lb3-gpr, Lagrangian-based denoted as Lagr) is reported. In Table 5.1, the average percentage gap among the makespan of the De Reyck and Herroelen and our algorithm with respect to the different lower bounds, computed as $\frac{\text{makespan}-\text{lowerbound}}{\text{makespan}}$ is presented. Results show that the percentage gap in Bianco and Caramia is always lower than in De Reyck and Herroelen.

It was studied also the effect of varying resource strength RS based on the deviations of the makespans of the two algorithms from the relative lower bounds. Table 5.2 reports the average gaps for different RS and $|V|$ values, computed as $\frac{\text{our makespan}-\text{Lagr}}{\text{our makespan}}$ and $\frac{\text{De Reyck Herroelen makespan}-\text{lb3}-\text{gpr}}{\text{De Reyck Herroelen makespan}}$, respectively. By Table 5.2,

Table 5.1 Average percentage gap between exact approach solutions and lower bound values

	De Reyck and Herroelen					Bianco and Caramia				
$\|V\|$	Network-based	Resource-based	BC11	lb3-gpr	Lagr	Network-based	Resource-based	BC11	lb3-gpr	Lagr
30	22.0%	20.3%	8.6%	12.4%	8.1%	15.3%	13.5%	0.8%	4.9%	0.3%
50	47.9%	61.7%	28.9%	30.5%	21.3%	44.6%	59.2%	24.4%	26.0%	16.3%
80	43.7%	41.0%	24.9%	30.7%	23.4%	37.7%	34.6%	16.8%	23.3%	15.2%
100	35.2%	63.6%	30.3%	33.8%	25.3%	29.1%	60.1%	23.7%	27.6%	18.2%

Table 5.2 Average percentage gap between the makespans and the lower bounds for different RS and $|V|$ values

	De Reyck and Herroelen			Bianco and Caramia		
$\|V\|$	RS = 0	RS = 0.25	RS = 0.5	RS = 0	RS = 0.25	RS = 0.5
30	12.1%	12.5%	12.6%	0.25%	0.26%	0.28%
50	30.2%	30.6%	30.7%	15.7%	16.5%	16.8%
80	29.7%	29.7%	32.8%	14.2%	15.4%	16.0%
100	33.8%	33.8%	33.9%	17.9%	18.2%	18.7%

it may be noticed that these gaps tend to increase when RS increases. Since the makespan decreases for increasing values of RS (see detailed results in Bianco and Caramia 2012), this implies a corresponding greater decrease of the lower bound values, that is meaningful since, when the instance is less constrained in terms of resources, the lower bounds are less effective.

5.3.3.3 Further Comparison with State of the Art Algorithms on Benchmarks

To further assess the effectiveness of our approach, we compared the performance of our algorithm with those of De Reyck and Herroelen (1998), Schwindt (1998), Fest et al. (1999), Dorndorf et al. (2000) approaches. Tests have been conducted on known benchmarks, i.e., the 1,080 problems with 100 activities generated by Schwindt (1996) using the generator ProGen/Max. Among these 1,080 problems, 31 do not admit a feasible solution.

In this comparison, results obtained by the competing algorithms have been taken verbatim from the paper of Dorndorf et al. (2000). We note that the machines used to experiment such algorithms were the following: the results of De Reyck and Herroelen (1998) were obtained by means of a Pentium 60/PC; those of Schwindt (1998), Fest et al. (1999), Dorndorf et al. (2000) by machines with 200 MHz clocks, i.e., Pentium/200 PC, Sun Ultra 200 MHz, and Pentium Pro/200 PC, respectively. Hence, the algorithm of De Reyck and Herroelen has computing times corresponding to 60/200 of the computation times of the other three approaches. To achieve a fair comparison we used a (old) Pentium Pro/200 PC to run our code.

Table 5.3 Comparison, on known benchmarks with 100 activities, of our algorithm with four competing approaches

Algorithm	t_{cpu}^{lim} (s)	Feasible	Optimal	Infeasibility proven
De Reyck and Herroelen	3	97.3	54.8	1.4
	30	97.5	56.4	1.4
	100	–	–	–
Schwindt	3	98.1	58.0	1.9
	30	98.1	62.5	1.9
	100	98.1	63.4	1.9
Fest et al.	3	92.2	58.1	1.9
	30	98.1	69.4	1.9
	100	98.1	71.1	1.9
Dorndorf et al.	3	97.8	66.2	1.9
	30	98.1	70.4	1.9
	100	98.1	71.1	1.9
Bianco and Caramia	3	98.1	67.6	1.9
	30	98.1	71.8	1.9
	100	98.1	72.2	1.9

This test consists in running the algorithm with a time limit t_{cpu}^{lim} of 3, 30, and 100 s, respectively. For each experiment, three percentages have been collected, i.e., the number of projects over the 1,080 instances for which:

- a feasible solution has been found (see column “feasible”);
- an optimal solution has been found and certified (see column “optimal”);
- infeasibility has been proven (see column “infeasibility proven”).

By the results reported in Table 5.3, it appears that the Bianco and Caramia approach is the most effective among the competing ones. Indeed, the Bianco and Caramia algorithm was always able to find either a feasible solution, when it exists, or infeasibility. Moreover, it was able to find optimal solutions on projects ranging from 67.6 to 72.2 % over the 1,080 instances, that represents the best performance among the competing approaches.

5.4 Conclusions

In this chapter, project scheduling with generalized precedence relations is considered. In the first part a new formulation of the resource unconstrained project scheduling problem is given. By means of this formulation we show that the minimum completion time can be obtained in $\mathcal{O}(m)$ time complexity against $\mathcal{O}(n \cdot m)$ time which is the traditional result present in the literature. In the second part, project scheduling with resource constraints is analyzed. For this problem a new lower bound, based on the previous resource unconstrained formulation, is proposed and experimented. Moreover, a new formulation in terms of mixed integer programming and a new exact algorithm are presented. The algorithm, based on branch-and-bound rules, exploits the previous formulation and a Lagrangian relaxation based lower bound. The extensive computational results presented support the effectiveness of the proposed approaches.

References

Ahuja RK, Magnanti T, Orlin J (1993) Network flows. Prentice Hall, New York

Bartusch M, Möhring RH, Radermacher FJ (1988) Scheduling project networks with resource constraints and time windows. Ann Oper Res 16(1):201–240

Bianco L, Caramia M (2010) A new formulation of the resource-unconstrained project scheduling problem with generalized precedence relations to minimize the completion time. Networks 56(4):263–271

Bianco L, Caramia M (2011a) A new lower bound for the resource-constrained project scheduling problem with generalized precedence relations. Comput Oper Res 38(1):14–20

Bianco L, Caramia M (2011b) Minimizing the completion time of a project under resource constraints and feeding precedence relations: a Lagrangian relaxation based lower bound. 4OR-Q J Oper Res 9(4):371–389

Bianco L, Caramia M (2012) An exact algorithm to minimize the makespan in project scheduling with scarce resources and generalized precedence relations. Eur J Oper Res 219(1):73–85

Demeulemeester EL, Herroelen WS (1997a) New benchmark results for the resource-constrained project scheduling problem. Manage Sci 43(11):1485–1492

Demeulemeester EL, Herroelen WS (1997b) A branch-and-bound procedure for the generalized resource-constrained project scheduling problem. Oper Res 45(2):201–212

Demeulemeester EL, Herroelen WS (2002) Project scheduling: a research handbook. Kluwer, Boston

De Reyck B (1998) Scheduling projects with generalized precedence relations: exact and heuristic approaches. Ph.D. dissertation, Department of Applied Economics, Katholieke Universiteit Leuven, Leuven

De Reyck B, Herroelen W (1998) A branch-and-bound procedure for the resource-constrained project scheduling problem with generalized precedence relations. Eur J Oper Res 111(1):152–174

Dorndorf U (2002) Project scheduling with time windows. Physica, Heidelberg

Dorndorf U, Pesch E, Phan-Huy T (2000) A time-oriented branch-and-bound algorithm for resource-constrained project scheduling with generalised precedence constraints. Manage Sci 46(10):1365–1384

Elmaghraby SEE, Kamburowski J (1992) The analysis of activity networks under generalized precedence relations (GPRs). Manage Sci 38(9):1245–1263

Fest A, Möhring RH, Stork F, Uetz M (1999) Resource-constrained project scheduling with time windows: a branching scheme based on dynamic release dates. Technical Report 596, Technical University of Berlin, Berlin

Held M, Karp RM (1970) The traveling-salesman problem and minimum spanning trees. Oper Res 18(6):1138–1162

Hochbaum D, Naor J (1994) Simple and fast algorithms for linear and integer programs with two variables per inequality. SIAM J Comput 23(6):1179–1192

Kelley JE (1963) The critical path method: resource planning and scheduling. In: Muth JF, Thompson GL (eds) Industrial scheduling. Prentice-Hall Inc., Englewood Cliffs, pp 347–365

Klein R, Scholl A (1999) Computing lower bounds by destructive improvement: an application to resource-constrained project scheduling. Eur J Oper Res 112(2):322–346

Kolisch R, Sprecher A, Drexl A (1995) Characterization and generation of a general class of resource-constrained project scheduling problems. Manage Sci 41(10):1693–1703

Moder JJ, Philips CR, Davis EW (1983) Project management with CPM, PERT and precedence diagramming, 3rd edn. Van Nostrand Reinhold Company, New York

Möhring RH, Schulz AS, Stork F, Uetz M (1999) Resource-constrained project scheduling: computing lower bounds by solving minimum cut problems. Lecture notes in computer science, vol 1643. Springer, Berlin, pp 139–150

Möhring RH, Schulz AS, Stork F, Uetz M (2003) Solving project scheduling problems by minimum cut computations. Manage Sci 49(3):330–350

Neumann K, Schwindt C, Zimmerman J (2003) Project scheduling with time windows and scarce resources, 2nd edn. LNEMS, vol 508. Springer, Berlin

Radermacher FJ (1985) Scheduling of project networks. Ann Oper Res 4(1):227–252

Schwindt C (1996) ProGen/Max: generation of resource-constrained scheduling problems with minimal and maximal time lags. Technical Report WIOR-489, University of Karlsruhe, Karlsruhe

Schwindt C (1998) Verfahren zur Lösung des ressourcenbeschränkten Projektdauerminimierungsproblems mit planungsabhängigen Zeitfenstern. Shaker, Aachen

Chapter 6
A Precedence Constraint Posting Approach

Amedeo Cesta, Angelo Oddi, Nicola Policella, and Stephen F. Smith

Abstract This chapter summarizes some previous work on a constraint-based scheduling approach effectively applied to Resource-Constrained Project Scheduling problems. The approach is based on a formulation of the problem as a Constraint Satisfaction Problem (CSP). In particular the problem is reduced to the one of establishing sufficient precedence constraints between activities that require the same resource so as to eliminate all possible resource contention, defining what is called the *Precedence Constraint Posting* (PCP) approach. The PCP scheduling approach has two attractive properties: first it operates in a search space that avoids overcommitment to specific activity start times, and can be more efficiently searched; second, the solution generated is a so-called "flexible schedule", designating a set of acceptable futures, which provides a basis for efficiently responding to unexpected disruptions during execution. This chapter summarizes a body of work developed over the years on PCP-based scheduling to take advantage of such properties. In particular, the chapter presents an overview on a number of original algorithms for efficiently finding a solution to a scheduling problem, for generating robust schedules, and for searching near-optimal makespan solutions.

Keywords Constraint-based reasoning • Generalized precedence relations • Makespan minimization • Renewable resources • Robust scheduling • Temporal flexibility

A. Cesta (✉) • A. Oddi
Institute of Cognitive Sciences and Technologies, CNR - Italian National Research Council, Rome, Italy
e-mail: amedeo.cesta@istc.cnr.it; angelo.oddi@istc.cnr.it

N. Policella
European Space Operations Centre, European Space Agency, Darmstadt, Germany
e-mail: nicola.policella@esa.int

S.F. Smith
Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA
e-mail: sfs@cmu.edu

C. Schwindt, J. Zimmermann (eds.), *Handbook on Project Management and Scheduling Vol. 1*, International Handbooks on Information Systems,
DOI 10.1007/978-3-319-05443-8_6

6.1 Introduction

Constraint-based scheduling models combine rich representational flexibility with compositional search procedures and heuristics, and this combination can provide significant leverage in addressing complex scheduling problems. These approaches start with a formulation of the problem of interest as a *Constraint Satisfaction Problem* (CSP), which involves specification of a set of decision variables together with a set of constraints on their mutual values. Much like the MILP formulations discussed in Chap. 2 of this handbook, there are different possibilities. One basic approach is to formulate the problem as one of finding a consistent assignment of start times for all constituent activities to be scheduled (analogous to the time-indexed formulations of Chap. 2 of this handbook). This fixed times assignment formulation, however, has a couple of drawbacks. From a constraint reasoning perspective, it promotes over-commitment (i.e., the problem constraints likely do not require commitment to specific start times to ensure feasibility), which results in a correspondingly larger solution space to search. From an operational perspective, a fixed-times solution offers no resilience against executional uncertainty, and is likely to quickly become invalid.

An alternative CSP formulation, which we adopt in this chapter, is to reduce the problem to one of establishing sufficient precedence constraints between activities competing for the same resources to eliminate all possible resource contention, and ensure both time and resource feasibility. This approach, referred to as Precedence Constraint Posting (PCP), address the over-commitment issue raised above by instead producing a so-called "flexible schedule" where activity start times are constrained to occur within an interval that is consistent with the problem constraints. The original formulation (Smith and Cheng 1993) was shown to achieve order-of-magnitude speedup in solving time over a corresponding fixed times assignment procedure in a basic job shop scheduling setting. More recently, this approach has been generalized to address cumulative resources, has been effectively applied to the resource constrained project scheduling problem (Cesta et al. 2002), and has been extended to generate Partial Order Schedules (POSs), activity networks with the property that any temporal solution to the graph is also resource-feasible (Policella et al. 2007). The ability to generate POSs provides a basis for efficiently responding to unexpected disruptions at execution time. It also provides a conceptual framework for optimizing solution robustness in the absence of knowledge about the uncertainties in the execution environment (similar in some ways to the robust scheduling techniques discussed in Chap. 40 in the second volume of this handbook, and in contrast to the stochastic models of Chap. 37 in the second volume of this handbook).

In this chapter we summarize this PCP approach to Project Scheduling. We present a basic set of core solving algorithms for generating a solution in the form of an activity network N, a directed graph $N = (V, E)$, where the edges in E are simple precedence constraints imposed on the set of problem activities V. Via a polynomial time calculation, such a network N can be always turned into a Partial

Order Schedule (*POS*). We also discuss incorporation of these core algorithms into extended optimizing search procedures targeted on two different objectives: improve the robustness of a solution (Policella et al. 2009) and minimize the solution makespan (e.g., Oddi et al. 2010a).

The chapter is organized as it follows: Sect. 6.2 reviews the state-of-the-art in Constraint-based Scheduling. After the introduction of the reference scheduling problem (RCPSP/max, Sect. 6.3) and its CSP representation (Sect. 6.4), the chapter than continues describing the basic concepts behind PCP (Sect. 6.5). A core PCP framework is discussed in Sect. 6.6. The last part of the chapter then summarizes the results along two different directions of work: a first for generating robust schedules (Sect. 6.7); a second one for the generation of optimal solutions (Sect. 6.8). The final conclusions are discussed in Sect. 6.9.

6.2 Constraint-Based Scheduling

Constraint Programming (see Rossi et al. 2006) is an approach to solving combinatorial search problems based on the Constraint Satisfaction Problem (CSP) paradigm (Kumar 1992; Montanari 1974; Tsang 1993). Constraints are just relations and a CSP states which relations should hold among the given problem *decision variables*. This framework is based on the combination of *search* techniques and *constraint propagation*. Constraint propagation consists of using constraints actively to prune the search space. Different propagation algorithms have been defined for different kinds of constraints. Their aim is to reduce the domains of variables involved in the constraints by removing the values that cannot be part of any feasible solution. The filtering algorithm is invoked any time a domain of some variable is changed (either as a result of search decisions or during constraint propagation) to propagate the consequences of the change over all decision variables. As described by Dechter and Rossi (2002), "in general, constraint satisfaction tasks, like finding one or all solutions or the best solution, are computationally intractable, $\mathscr{NP}$-hard". For this reason the constraint propagation process cannot be complete, that is, some infeasible values may still remain in the domains of the variables and thus decisions are necessary to find a complete feasible valuation of the variables. In general, constraint propagation is aimed at achieving local consistency among subsets of variables.

Constraint satisfaction and propagation rules have been successfully used to model, reason and solve about many classes of problems in such diverse areas as scheduling, temporal reasoning, resource allocation, network optimization and graphical interfaces. In particular, CSP approaches have proven to be an effective way to model and solve complex scheduling problems (see, for instance, Baptiste et al. 2001; Beck et al. 1998; Cesta et al. 2002; Fox 1990; Sadeh 1991; Smith 1994). The use of variables and constraints provides representational flexibility and reasoning power. For example, variables can represent the start and the end times of an activity, and these variables can be constrained in arbitrary ways.

In the remainder of this section we first provide a formal definition of the Constraint Satisfaction Problem, next a brief description of a generic solver, and finally an overview of the different ways in which this paradigm has been used to solve scheduling problems.

6.2.1 Constraint Satisfaction Problem

A *Constraint Satisfaction Problem*, CSP, consists of a finite set of *decision variables*, each associated with a domain of values, and a set of constraints that define the relation between the values that the variables can assume. Therefore, a CSP is defined by the tuple $\langle V, D, \mathfrak{C} \rangle$ where:

- $V = \{v_1, v_2, \ldots, v_n\}$ is a set of n variables;
- $D = \{D_1, D_2, \ldots, D_n\}$ is the set of corresponding domains of values for any variable, such that $v_i \in D_i, i = 1, \ldots, n$;
- $\mathfrak{C} = \{c_1, c_2, \ldots, c_\nu\}$, is a set of ν constraints, $c_\mu(v_1, v_2, \ldots, v_n)$, that are predicates defined on the Cartesian product of the variable domains, $D_1 \times D_2 \times \ldots \times D_n$.

A *solution s* is a value assignment to each variable v_i, from its domain,

$$(\lambda_1, \lambda_2, \ldots, \lambda_n) \in D_1 \times D_2 \times \ldots \times D_n$$

such that the set of constraints is satisfied.

Constraint processing tasks include not only the satisfaction task, but also *Constraint Optimization Problems* (COP). In this case an *objective function* $f(S)$ (or cost function) evaluates any single feasible solution S. The goal is to find an optimal solution S^* which minimizes (or maximizes) the objective function $f()$.

Finally, we observe an instance of a CSP $\langle V, D, \mathfrak{C} \rangle$ can be represented as a constraint graph, $G = (V, E)$. For every variable $v_i \in V$, there is a corresponding node in the graph. For every set of variables connected by a constraint $c_j \in \mathfrak{C}$, there is a corresponding hyper-edge. In the particular case of binary constraints (each constraint involves at most two variables) the hyper-edges become simply edges. A well known example of binary CSP (extensively used in this chapter) is the Simple Temporal Problem (STP) introduced by Dechter et al. (1991).

6.2.2 A Generic CSP Solver

A complete CSP solving procedure consists of the following three steps (Algorithm 6.1): (a) current problem P is checked for consistency [CheckConsistency(P)] by the application of a *propagation* procedure, if at least one constraint is violated the algorithm exits with failure. If the problem P is also a solution [IsSolution(P)],

Algorithm 6.1: CSP-Solver(P)

```
if CheckConsistency($P$) then
   if IsSolution($P$) then
      $S \leftarrow P$
      return $S$
   else
      $v_i \leftarrow$ SelectVariable($P$)
      $\lambda_i \leftarrow$ Choose-Value($P, v_i$)
      CSP-Solver($P \cup \{\lambda_i\}$)
   end if
else
   return $\emptyset$
end if
```

then the algorithm exits and returns the generated solution $S = P$. Otherwise the problem is still not solved and the following two steps are executed; (b) a variable v_i is selected by a *variable ordering* heuristic; (c) a value λ_i is chosen by a value ordering heuristic and added to P. The solver is recursively called on the updated problem $P \cup \{\lambda_i\}$.

6.2.3 CSP Approaches to Scheduling Problems

Scheduling problems are difficult combinatorial optimization problems and represent an important application area for constraint directed search. Different constraint programming approaches have been developed in this direction, for instance, the reader can refer to Baptiste et al. (2001) for a thorough analysis of different constraint based techniques for scheduling problems. The work of Constraint directed Scheduling of the 1980s (see for example Fox 1990; Sadeh 1991; Smith 1994) has developed into Constraint-based Scheduling approaches in the late 1990s (see Baptiste and Le Pape 1995; Beck et al. 1998; Hentenryck and Michel 2009; Nuijten and Aarts 1996; Smith and Pyle 2004). These approaches all focus on the use of constraints as a basis for representing and managing the search for a solution to the scheduling problem at hand. As mentioned above, the search for a solution to a CSP can be viewed as modifying the constraint graph $G = (V, E)$ through the addition and removal of constraints, where the constraint graph is an evolving representation of the search state, and a solution is a state in which a single value remains in the domain of each variable and all constraints are satisfied.

As mentioned at the outset, research in constraint-based scheduling has pursued two basic formulations of the scheduling problem. One set of approaches (e.g., Nuijten and Le Pape 1998; Sadeh 1991; Smith and Pyle 2004) has formulated the problem as that of finding a consistent assignment of start times for each activity to be performed. Under this model, decision variables are time points that designate the start times of various activities and CSP search focuses on determining a consistent assignment of start time values. The "serial and parallel" methods discussed in Chap. 1 of this handbook similarly aim to find consistent sets of start times. A second set of approaches have focused on a problem formulation more akin to least-commitment frameworks. In this model, which is based on a disjunctive graph representation (Adams et al. 1988) and is referred to as *Precedence Constraint Posting* (Smith and Cheng 1993), solving consists of posting additional precedence constraints between pairs of activities contending for the same resources to ensure feasibility with respect to time and capacity constraints. Solutions generated in this way generally represent a set of feasible schedules (i.e., the sets of activity start times that remain consistent with posted sequencing constraints), as opposed to a single assignment of start times.

6.3 The Reference Scheduling Problem: RCPSP/max

We adopt the Resource-Constrained Project Scheduling Problem with minimum and maximum time lags, RCPSP/max,[1] as a reference problem (see Bartusch et al. 1988). The basic entities of interest in this problem are *activities*. The set of activities is denoted by $V = \{1, 2, \dots n\}$ where each activity has a fixed *processing time*, or *duration*, p_i and must be scheduled without *preemption*.

A *schedule* is an assignment of start times to each activity in V, i.e., a vector $S = (S_1, S_2, \dots, S_n)$ where S_i denotes the start time of activity i. The time at which activity i has been completely processed is called its *completion time* and is denoted by C_i. Since we assume that processing times are deterministic and preemption is not permitted, completion times are determined by:

$$C_i = S_i + p_i \quad (i \in V) \tag{6.1}$$

Schedules are subject to two types of constraints, *temporal constraints* and *resource constraints*. In their most general form temporal constraints designate arbitrary minimum and maximum time lags between the start times of any two activities,

$$d_{ij}^{min} \leq S_j - S_i \leq d_{ij}^{max} \quad ((i, j) \in V) \tag{6.2}$$

where d_{ij}^{min} and d_{ij}^{max} are the minimum and maximum time lag of activity j relative to i. A schedule $S = (S_1, S_2, \dots, S_n)$ is *time feasible*, if all inequalities given by the activity precedences/time lags (6.2) and durations (6.1) hold for start times S_i.

During their processing, activities require specific resource units from a set $\mathscr{R} = \{1, 2, \dots, K\}$ of resources. Resources are *reusable* (renewable), i.e., they are released when no longer required by an activity and are then available for use by another activity. Each activity $i \in V$ requires r_{ik} units of the resource $k \in \mathscr{R}$ during its processing time p_i. Each resource $k \in \mathscr{R}$ has a limited capacity of R_k units.

[1] The three field classification of RCPSP/max is $PS|temp|C_{max}$.

A schedule S is *resource feasible* if at each time t the demand for each resource $k \in \mathscr{R}$ does not exceed its capacity R_k, i.e.,

$$r_k(S,t) = \sum_{i \in V : S_i \leq t < C_i} r_{ik} \leq R_k \quad (k \in \mathscr{R};\ t \in [0,T]) \tag{6.3}$$

A schedule S is called *feasible* if it is both time and resource feasible.

The RCPSP/max problem is a complex scheduling problem: in fact not only the optimization version but also the feasibility problem is $\mathscr{NP}$-hard (see Bartusch et al. 1988). The reason for this $\mathscr{NP}$-hardness result lies in the presence of maximum time lags. In fact these imply the presence of deadline constraints, transforming feasibility problems for precedence-constrained scheduling to scheduling problems with time windows.

6.4 The RCPSP/max as a CSP

As introduced above, we formulate the project scheduling problem as a Constraint Satisfaction Problem (CSP), in particular we refer to the definition of the problem proposed in the work Bartusch et al. (1988), such that *decision variables* are the so-called *Forbidden Sets* also known as *Minimal Critical Sets* (see Laborie and Ghallab 1995 or Chap. 2 of this handbook, we use MCS in the rest of the chapter).

Given a generic resource k, a *conflict* is a set of activities requiring the resource k, which can mutually overlap and whose combined resource requirement is in excess of the resource capacity R_k. A Minimal Critical Set, $MCS \subseteq V$, represents a resource conflict of minimal size (each subsets is not a resource conflict), which can be *resolved* by posting a single precedence constraint between two of the competing activities in the conflict set. Hence, in CSP terms, a decision variable is defined for each MCS and the domain of possible vales is the set of all possible feasible precedence constraints $i \prec j$ which can be imposed between any pair of activities in the MCS.

A *solution* of the scheduling problem is a set of precedence constraints (added to the original problem described in the previous Sect. 6.3) such that removes all the MCSs.

A solution takes the form of an activity network N_S, a directed graph $N_S = (V_S, E)$, where $V_S = V \cup \{source, sink\}$, the set of problem activities V plus two fictitious activities *source* and *sink*, and E is the set of directed edges (i, j), representing the set of precedence constraints $i \prec j$ defined among the activities in V_S. In particular, the set E is partitioned in two subsets, $E = E_{prob} \cup E_{post}$, where E_{prob} is the set of precedence constraints originating from the problem definition and E_{post} is the set of precedence constraints posted to resolve resource conflicts. In general, the directed graph $N_S(V_S, E)$ represents a set of temporal solutions $(S_1, S_2, \ldots, S_n)$, that is a set of assignments to the activities' start-times which are consistent with the set of constraints E and the set of imposed resource constraints.

We observe as in the new formulation of the problem it becomes a *pure* disjunctive temporal problem (Oddi et al. 2010b), such that the original resource constraints are *compiled* into a set of MCSs, each MCS can be seen as a disjunctive temporal clause. A drawback of the previous MCS reduction is the large number of MCSs obtained for each resource k; if n_k is the number activities requiring resource k, the number of MCSs is $\binom{|n_k|}{R_k+1}$, hence $\mathcal{O}(n_k^{R_k+1})$.

The next section proposes a summary of the works (Cesta et al. 1999, 2002) which describes how to overcame the limitation imposed by the large size set of MCSs. The proposed approach, targeted to identification of decision variables, attempts to reconcile two, typically conflicting desiderata: (1) on one hand to always take the decision centers on the most critical precedence constraint to post and (2) on the other to minimize the amount of time spent in the analysis that leads to this decision. For details on the empirical evaluation of the algorithms the author can refer to the original works (Cesta et al. 1999, 2002).

6.5 Precedence Constraint Posting

The proposed Precedence Constraint Posting (PCP) approach was first introduced by Smith and Cheng (1993) for problems with binary resources and then extended to more general problems in subsequent research, aims at synthesizing additional precedence constraints between pairs of activities for the purpose of pruning all inconsistent allocations of resources to activities. The general schema of this approach is provided in Fig. 6.1 and consists of representing, analyzing, and solving different aspects of the problem in *two separate layers*. In the former the temporal aspects of the scheduling problem, e.g., activity durations, constraints between pairs of activities, due dates, release time, etc., are considered. The second layer, instead, represents and analyzes the resource aspects of the problem. Let us now explain the details of the two layers.

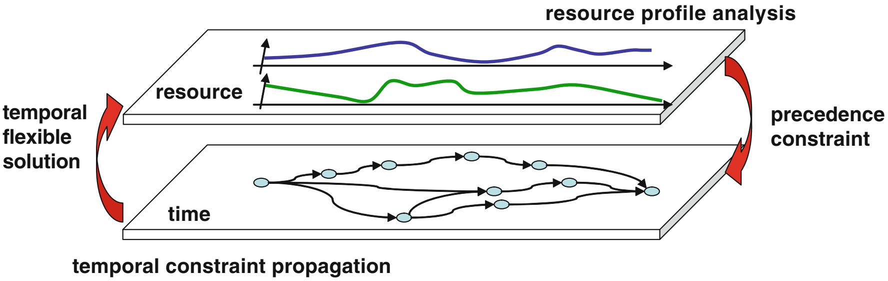

resource profile analysis
resource
temporal flexible solution
precedence constraint
time
temporal constraint propagation

Fig. 6.1 Precedence Constraint Posting schema

6.5.1 Time Layer

The temporal aspects of the scheduling problems are represented through an STP (Simple Temporal Problem) network (see Dechter et al. 1991). This is a temporal graph in which the set of nodes represents a set of temporal variables named *time-points*, tp_μ, while linear temporal constraints, of the form $tp_\mu - tp_\nu \leq d_{\mu\nu}$, define the distances among them. Each time point has initially a domain of possible values equal to $[0, T]$ where T is the temporal horizon of the problem. The problem is represented by associating with each activity a pair of time points which represent, respectively, the start and the end-time of the activity. Therefore a temporal constraint may be imposed between a pair of time points that can "belong" to the same activity or not. In the latter case (when they do not belong to the same activity) the temporal constraints represent constraints between two activities of the problem. If alternatively, the two time-points belong to the same activity, the temporal constraints represent the duration, or processing time, of the activity. By propagating the temporal constraints it is possible to bound the domains of each time-point, $tp_\mu \in [lb_\mu, ub_\mu]$. In the case of empty domains for one or more time-points the temporal graph does not admit any solution. In Dechter et al. (1991) it was proved that it is possible to completely propagate the whole set of temporal constraints in polynomial time, $\mathcal{O}(n^3)$, and, moreover, that a solution can be obtained by selecting for each time-point its lower bound value, $tp_\mu = lb_\mu$ (this solution is referred to as the *Earliest Start-Time Solution*). The temporal layer then, given the temporal aspects of a scheduling problem, provides, in polynomial time (using constraint propagation) a set of solutions defined by a temporal graph. This result is taken as input in the second layer. In fact, at this stage we have a set of temporal solutions (time feasible) that need to also be proven to be resource feasible.

6.5.2 Resource Layer

This layer takes into account the other aspect of the scheduling problem, namely the constraints on resources (i.e., capacity). In general, resources can be binary, multi-capacitive, or consumable (non-renewable). As described above, the input to this layer in the PCP approach is a temporally flexible solution—a set of temporal solutions (see also Fig. 6.1). Like in the previous layer it is possible to use constraint propagation to reduce the search space. Even though there are different methodologies described in the literature, these propagation procedures are not sufficient in general (see Laborie 2003; Nuijten and Aarts 1996). In fact they are not complete, which implies that they are not able to prune all inconsistent temporal solutions. For this reason a PCP procedure uses a *Resource Profile* (see Cesta et al. 2002) to analyse resource usage over time and detect MCS decision variables. The procedure then proceeds to post further constraints to level (or solve) some of the detected conflicts. These new constraints are propagated in the underlying layer to check the temporal consistency. Then the time layer provides a new temporally flexible solution that is analyzed again using the resource profiles. The search stops when either the temporal graph becomes inconsistent or the resource profiles are consistent with the resource capacities.

6.6 The Core Constraint-Based Scheduling Framework

The core of the implemented framework is based on the greedy procedure described in Algorithm 6.2, which is an instance of the procedure described in Sect. 6.2.2. Within this framework, a solution is generated by progressively detecting time periods where resource demand is higher than resource capacity (conflicts) and posting sequencing constraints between competing activities to reduce demand and eliminate capacity conflicts. As explained above, after the current situation is initialized with the input problem, $S^0 \leftarrow P$, the procedure builds an estimate of the required resource profile according to the current temporal precedences in the network and detects resource conflicts—SELECTCONFLICTSET(S^0). If the set of conflicts F is not empty then new constraints are synthesized, SELECTLEVELING-CONSTRAINT(F), and posted on the current situation. The search proceeds until either the STP temporal graph becomes inconsistent or a solution is found.

Using the reference scheduling problem (RCPSP/max) introduced above, we proceed now to summarize the core components needed to fully specify the approach. In fact, the introduction of a specific scheduling problem allows us to set the general framework introduced above. The first issue in the framework implementation concerns how to identify activities that are in a conflicting situation. This allows identification of the points in the current solution state that need to be resolved. The second issue concerns the heuristics (variable and value ordering) used to respectively select and solve conflicts that have been identified.

6.6.1 Consider Resource Utilization

A first, important issue that needs to be explored is how to compute *resource utilization profiles*. In fact, the input temporal graph represents a set of solutions, possibly infinite, and to consider all the possible combinations is impossible in practice.

A possible affordable alternative consists of computing bounds on resource utilization. Examples of bounding procedures can be found in Drabble and Tate (1994), Cesta and Stella (1997), Laborie (2003), Muscettola (2002). It is worth noting that consideration of resource bounds as resource profiles implies that all temporal solutions represented by the temporal graph are also resource feasible.

A different approach to dealing with resources consists of focusing attention on a specific temporal solution and its resource utilization. In contrast to resource

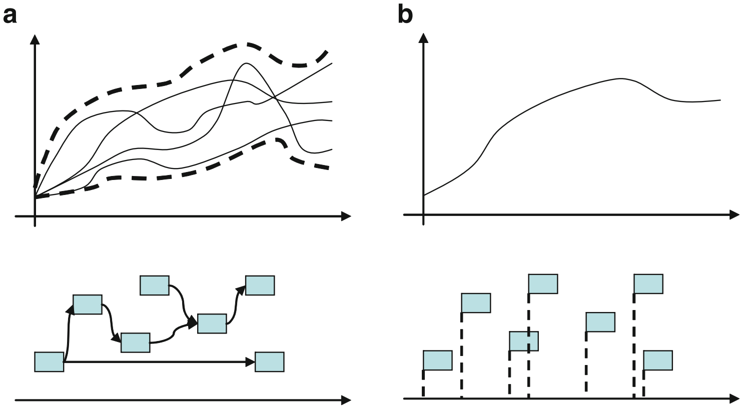


Fig. 6.2 Two different ways to consider the resource utilization. (**a**) Bounds of the resource utilization for the set of solutions defined by a temporal graph. (**b**) Resource utilization of a single temporal solution

bounding approaches, this process only assures that the final temporal graph contains at least one resource feasible solution (the one for which the resource utilization is in fact considered); some of the temporal solutions may not be resource feasible. Since only a single feasible solution is computed instead of encapsulating all possible feasible solutions, this approach gains substantial computational efficiency.

Figure 6.2 summarizes the two alternative resource profiles. In the first case resource bounds are used to consider all the temporal solutions and their associated resource utilization (Fig. 6.2a). Alternatively, only one temporal solution of the set is considered in the second case (Fig. 6.2b).

6.6.2 How to Identify Decision Variables

The starting point in identifying the possible conflicts is the computation of the possible contention *peaks*. A *contention peak* is a set of activities whose simultaneous execution exceeds the resource capacity. A contention peak designates a conflict of a certain size (corresponding to the number of activities in the peak). In Algorithm 6.2 the function SELECTCONFLICTSET(S^0) collects all *maximal peaks*[2] in the current schedule. Then selects a decision variable (MCS) from the set of peaks. An alternative selection procedure first ranks the selected peaks, next picks the more critical one (e.g., one with maximal size), and last selects a decision

[2] Specifically, we follow a strategy of collecting sets of activities such that none of the sets is a subset of the others.

Algorithm 6.2: GREEDYPCP(P)

```
Require: a problem $P$
Ensure: a solution $S$ (or the empty set otherwise)
  $S^0 \leftarrow P$
  if Exists an unresolvable conflict in $S^0$ then
    $S \leftarrow \emptyset$
  else
    $F \leftarrow$ SELECTCONFLICTSET($S^0$)
    if $F = \emptyset$ then
      $S \leftarrow S^0$
    else
      $\{i \prec j\} \leftarrow$ SELECTLEVELINGCONSTRAINT($F$)
      $S^0 \leftarrow S^0 \cup \{i \prec j\}$
      $S \leftarrow$ GREEDYPCP($S^0$)
    end if
  end if
  return $S$
```

variable from the selected peaks. The selected MCS is solved by imposing on the conflicting activities a single precedence constraint $i \prec j$. Next Sect. 6.6.3 gives more details about the used selection procedures.

Cesta et al. (1999, 2002) showed that much of the advantage of this type of global conflict analysis can be retained by using an approximate polynomial procedure for computing MCSs. In particular, Cesta et al. (1999, 2002) proposed two polynomial strategies for sampling MCSs from a peak of size $|F|$. These strategies are based on the idea of sorting the activities of each peak according to their resource usage (greatest first), then MCSs are collected by visiting such a list and extracting subsequences of activities corresponding to MCSs. The two methods are named *linear* and *quadratic* according to their complexity (they respectively collect $\mathcal{O}(|F|)$ and $\mathcal{O}(|F|^2)$ elements). An additional and effective strategy is based on the idea of imposing a *lexicographical* order on the set of searched MCSs, the reader can refer to the paper Cesta et al. (2002) to see the detail of the procedure.

6.6.3 Selecting and Solving Conflicts

According to the proposed CSP framework, in all cases where no mandatory decisions can be deduced from the propagation phase, heuristics and methods used to respectively select and solve one of the conflicts are introduced by defining *variable* and *value* ordering heuristics for the decision variables. The basic idea is to repeatedly evaluate the decision variables and select the one with the best heuristic evaluation. The selection of which variable to assign next is based on the *most constrained first* (MCF) principle, and the selection of values follows the *least constraining value* (LCV) heuristic. More specifically, the following heuristics are assumed:

Ranking conflicts: for evaluating MCSs we have used the heuristic estimator $\kappa()$ described by Laborie and Ghallab (1995), where the MCS with highest value of $\kappa(MCS)$ is then chosen. A conflict is unsolvable if no pair of activities in the conflict can be ordered. Basically, $\kappa()$ will measure how close a given conflict is to being unsolvable.

Slack-based value selection: to choose an ordering decision among the possible, the choice which retains the most temporal slack is taken.

It is worth underscoring that the above PCP framework establishes resource feasibility strictly by sequencing conflicting activities. It remains non-committal on activity start times. As such, PCP preserves temporal flexibility that follows from problem constraints. Further, the two heuristic choices adopt a minimal commitment strategy with respect to preserving temporal slack, and this again favors temporal flexibility.

6.7 Flexible Solutions, Robustness, and Partial Order Schedules

As shown in the previous section, the outcome of a Precedence Constraint Posting solver is a Simple Temporal Problem (STP) network or STN, that not only contains the temporal constraints belonging to the initial problem, but also the additional precedences that have been added during the resolution process. In a STN, each time point is associated with a bounded interval of values which represents the set of admissible values for that time point; hence, the adjective "temporally flexible" is often used to refer to these kind of solutions. Therefore the PCP approach tries to retain the temporal flexibility of the underlying STN to the extent possible (somehow maximizing the domain size of the time points). In particular, the use of PCP approaches (and the schedules produced by them) can be justified in two ways:

- As a means of retaining the flexibility implied by problem constraints (time and capacity) and avoiding over commitment;
- As a means of establishing conditions for guaranteed *executability*.

In fact, in most practical scheduling environments, off-line schedules can have a very limited lifetime and scheduling is really an ongoing process of responding to unexpected and evolving circumstances. In such environments, insurance of robust response is generally the first concern. Unfortunately, the lack of guidance that might be provided by a schedule often leads to myopic, sub-optimal decision-making.

One way to address this problem is *reactively*, through schedule repair. To keep pace with execution, the repair process must be both fast and complete. The response to a disruption must be fast because of the need to re-start execution of the schedule as soon as possible. A repair must also be complete in the sense of accounting for all changes that have occurred, while attempting to avoid the introduction of new changes. As these two goals can be conflicting, a compromise solution is often required. Different approaches exist and they tend to favour either timeliness (Smith 1994) or completeness (El Sakkout and Wallace 2000) of the reactive response.

An alternative, *proactive* approach to managing execution in dynamic environments is to focus on building schedules that retain flexibility and are able to absorb some amount of unexpected events without rescheduling. One technique consists of factoring time and/or resource redundancy into the schedule, taking into account the types and nature of uncertainty present in the target domain (Davenport et al. 2001; Hiatt et al. 2009). An alternative technique is to construct an explicit set of contingencies (i.e., a set of complementary solutions) and use the most suitable with respect to the actual evolution of the environment (Drummond et al. 1994). Both of these proactive techniques presume an awareness of the possible events that can occur in the operating environment, and in some cases, these knowledge requirements can present a barrier to their use.

Research approaches are based on different interpretations of the concept of a robust solution, e.g., the ability to preserve solution qualities or the ability to maintain a stable solution. The concept of robustness, on which this work is based, can be viewed as execution-oriented; a solution to a scheduling problem will be considered *robust* if it provides two general features: (1) the ability to absorb exogenous and/or unforeseen events without loss of consistency, and (2) the ability to keep the pace with the execution guaranteeing a prompt answer to the various events.

Figure 6.3a describes the execution of a schedule. This is given to an executor (it can be either a machine or a human being) that manages the different activities. If something happens (i.e., an unforeseen event occurs) the executor will give feedback to a scheduler module asking for a new solution. Then, once a new

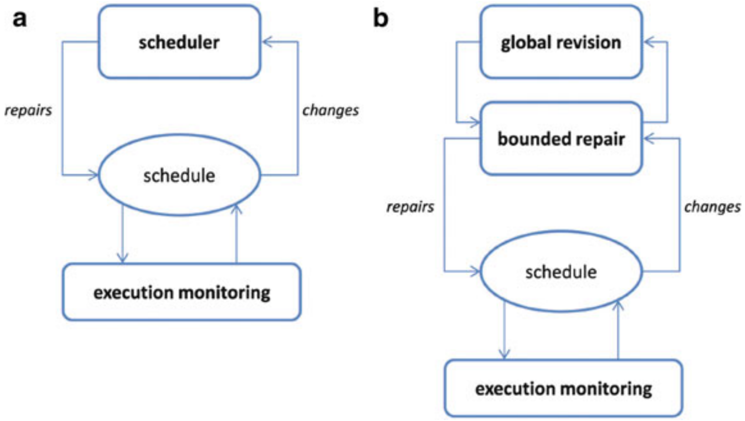


Fig. 6.3 Rescheduling actions during the execution. (**a**) General rescheduling phase. (**b**) Rescheduling phase using a flexible solution

solution is computed, it is given back to the executor. In Fig. 6.3b, instead, the execution of a flexible schedule is highlighted. The substantial difference in this case is that the use of flexible solutions allows the introduction of two separate rescheduling phases: the first enabling rapid response by immediate means like temporal constraint propagation over the set of activities, and the second entailing to more extensive re-computation of the schedule when the first phase cannot offer a response. In practice, the first phase exploits the flexibility characteristics of the solution (and for this reason we named this module *bounded repair scheduler*). Of course it is possible that an unforeseen event will force the system outside of the bounds provided by the flexible solution. In this case, it will be necessary to invoke the second, more complete scheduling phase. This second phase involves re-computation of the overall flexible schedule, and performs a much more extensive constraint-based search procedure (*global revision* module). Note that the use of flexible schedules makes it possible to bypass this extended computation in many circumstances in favor of a prompt answer.[3]

6.7.1 Partial Order Schedules

To take maximum advantage of the opportunity to bypass extended computation in the event of unexpected events, a stronger form of flexible schedule is required. Policella et al. (2004) and Policella (2005) further elaborated the idea of exploiting temporal flexibility by adopting a graph formulation of the scheduling problem and focusing on generation of the *Partial Order Schedules* (*POS*s).

Definition 6.1 (Partial Order Schedule). A Partial Order Schedule *POS* for a problem *P* is an activity network, such that any possible temporal solution is also a resource-consistent assignment.

Within a *POS*, each activity retains a set of feasible start times, and these options provide a basis for responding to unexpected disruptions.

An attractive property of a *POS* is that reactive response to many external changes can be accomplished via simple propagation in an underlying temporal network (a polynomial time calculation); only when an external change exhausts all options for an activity is it necessary to recompute a new schedule from scratch. In fact the augmented duration of an activity, as well as a greater release time, can be modeled as a new temporal constraint to post on the graph. To propagate all the effects of the new edge over the entire graph it is necessary to achieve the *arc-consistency* of the graph (that is, ensure that any activity has a legal allocation with respect to the temporal constraints of the problem).

Note that, even though the propagation process does not explicitly consider consistency with respect to the resource constraints, it is guaranteed to obtain a

[3] Although the reader should also note that these solutions are in general sub-optimal.

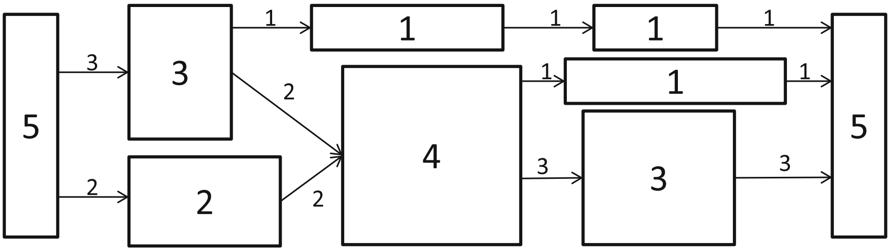


Fig. 6.4 An example of Partial Order Schedule (*POS*)

feasible solution by definition. Therefore a partial order schedule provides a means to find a new solution and ensures to compute it in a fast way.

The common thread underlying a *POS* is the characteristic that activities which require the same resource units are linked via precedence constraints into precedence *chains*. Given this structure, each constraint becomes more than just a simple precedence constraint, but represents a *producer-consumer* relation, allowing each activity to be connected with the set of predecessors which supply the units of resource it requires for execution. In this way, the resulting network of chains can be interpreted as a *flow* of resource units through the schedule; each time an activity completes its execution, it passes its resource unit(s) on to its successors (a similar formulation is used in Chap. 2 of this handbook). Figure 6.4 shows an example of Partial Order Schedule for a single resource with capacity five. In particular, activities are represented as rectangles and edges represent the precedence constraints. The numbers inside the rectangles represent the resource requirements and the labeling numbers on the directed edges represents the flow of resource units supplied to a generic activity i from its predecessors in order to satisfy the imposed resource constraint, independently from the start time values of the activities. For example, the activity which requires four units of resource receives two units of resource from each of its two predecessors and supplies one and three units of resource respectively to its two successors. In general, in a *POS* solution each activity has a set of inputs predecessors which supply the units of resource needed for its execution.

Hence, an activity network $N_S(V_S, E_{POS})$ is in *POS-form* if for each resource k there exists a labeling function $f_k : E_{POS} \longrightarrow [0..R_k]$ representing the flow of resource units among the activities such that for each activity i the following constraint holds:

$$\sum_{j \in Pred(i)} f_k(j,i) = \sum_{j \in Succ(i)} f_k(i,j) = r_{ik} \tag{6.4}$$

where $Pred(i) = \{j \in V | \exists (j,i) \in E_{POS})\}$ and $Succ(i) = \{j \in V | \exists (i,j) \in E_{POS})\}$.[4] Given an input solution S (represented either as a graph or as a set of

[4] $Pred(source) = Succ(sink) = \emptyset$.

start-time values) a polynomial transformation method, named CHAINING, can be defined that creates sets of activity *chains* (Policella et al. 2007). This operation can be accomplished in three steps:

1. all the previously posted leveling constraints are removed from the input partial order;
2. the activities are sorted by increasing activity earliest start times;
3. for each resource and for each activity i (according to the increasing order of start times), one or more predecessors p are chosen, which supplies the units of resource required by i – a precedence constraint $p \prec i$ is posted for each predecessor p. The last step is iterated until all the activities are linked by precedence chains and the constraints in (6.4) are satisfied.

Before concluding we make a further remark about partial order schedules. Roy and Sussman (1964) introduced the disjunctive graph representation of the classical job shop scheduling problem and describe how a solution can be achieved by solving all the disjunctive constraints and transforming each into a conjunctive one. Also in our case of RCPSP/max, solution of all disjunctive constraints is required to achieve a *POS*. In essence, the disjunctive graph representation has been extended to the more general case where multi-capacity resources are defined. In this case "disjunctive" hyper-constraints among activities that use the same resource are introduced, the so-called MCSs. Based on this representation we can note that a partial order schedule is obtained once any disjunctive MCS is solved. In this case, a set of precedence constraints is posted to solve each MCS.

6.8 Extended Optimizing Search

In the previous sections we have presented a basic set of core solving algorithms for generating a solution in the form of an activity network N. In addition, we have also shown as such a network can be always turned into a Partial Order Schedule (*POS*) via a polynomial time calculation. Now in this section we summarize how to use these core algorithms into a set of extended optimizing search procedures targeted on two different objectives: minimize the solution makespan and improve the *robustness* of a solution.

A first procedure is described in (Cesta et al. 2002), where an iterated version of Algorithm 6.2, called the ISES procedure, is proposed. This algorithm is an iterative method, which at each step utilizes a *randomized* version of Algorithm 6.2 to produce different solutions. The key idea underlying the described approach is to heuristically bias random choices in a dynamic fashion, according to how well (or how poor) the available search heuristics (variable and value ordering) discriminate among several alternatives.

A generalization of the previous procedure is the so-called *Iterative Flattening Search* (IFS, Cesta et al. 2000; Oddi et al. 2010a). The concept of iterative flattening search is quite general and provides a framework for designing effective procedures

Algorithm 6.3: IFS(S, *MaxFail*)

```
$S_{best} \leftarrow S$
$counter \leftarrow 0$
while $counter \leq MaxFail$ do
    Relax($S$)
    $S \leftarrow$ PCP($S$)
    if $C_{max}(S) < C_{max}(S_{best})$ then
        $S_{best} \leftarrow S$
        counter $\leftarrow 0$
    else
        $counter \leftarrow counter + 1$
    end if
end while
return $S_{best}$
```

for scheduling optimization, this concept is also known in literature as Large Neighborhood Search (LNS) and was independently developed for solving vehicle routing problems in Shaw (1998). It iteratively applies two steps: (1) Random relaxation of the current solution; (2) An incremental solving step to regain solution feasibility. Algorithm 6.3 introduces the generic IFS procedure. The algorithm alternates relaxation and flattening steps until a better solution is found or a maximal number of non-improving iterations is reached. The procedure takes two parameters as input: (1) an initial solution S; (2) a positive integer *MaxFail*, which specifies the maximum number of consecutive non makespan-improving moves that the algorithm will tolerate before terminating. After initialization, a solution is repeatedly modified within the while loop by applying the RELAX procedure, and a PCP procedure is used as solving step. At each iteration, the RELAX step reintroduces the possibility of resource contention, and the PCP step is called again to restore resource feasibility. If a better makespan solution is found, the new solution is saved in S_{best}. If no improvement is found within *MaxFail* moves, the algorithm terminates and returns the best solution found. It is worth noting that Partial Order Schedules in the context of IFS (or LNS) are an effective way for designing neighborhood structures, examples of neighborhoods for solving scheduling problems with cumulative renewable resources are given in the papers Oddi et al. (2010a), in addition, as shown in Laborie and Godard (2007), the concept can be extended to various types of resources.

The problem of increasing (optimizing) the robustness of generated *POS*s is addressed via an iterative (randomized) chaining procedure in Policella et al. (2009). In particular, the problem is addressed by separating the phase of problem solution, which may pursue a standard optimization criterion (e.g., minimal makespan), from a subsequent phase of solution *robustification* in which a more flexible set of solutions is obtained and compactly represented through a Partial Order Schedule. In particular, the paper focuses on specific heuristic algorithms for synthesis of *POS*s, starting from a pre-existing schedule and different extensions of the technique CHAINING algorithm described in Sect. 6.7.1, which progressively introduces temporal flexibility into the representation of the solution. In fact, we can observe that given the form of the output solution (an activity network), "classical" objective like the minimization of project makespan, C_{max}, co-exists very naturally with a *POS* solution and the objective of increasing the solution's robustness. In fact, in the best case, the early start time solution which minimizes makespan is executed; but in the event that unexpected events prevent this possibility, the accompanying *POS* provides a feasible, bounded relaxation strategy.

6.9 Conclusions

In this chapter we have summarized a constraint satisfaction problem solving (CSP) framework for solving project scheduling problems. We have advocated a somewhat unconventional Precedence Constraint Posting (PCP) approach, which exploits the expressiveness and computational efficiency of a simple temporal network (STN) to enforce complex temporal constraints and interdependencies between activities, and establishes resource feasibility by iteratively sequencing activities that are found to be competing for the same resources until all potential resource conflicts have been resolved. One important advantage of a PCP approach is that a generated solution consists of a set of feasible schedules, as opposed to a single assignment of activity start times that is the target of many scheduling approaches. In fact, such a flexible solution can be efficiently transformed into a Partial Order Schedule (*POS*), which guarantees resource feasibility over ranges of activity start times that are consistent with posted constraints. *POS*s can be enable quick reaction to unforeseen events during execution via efficient temporal constraint propagation procedures.

We have discussed core technology components for managing complex temporal constraints and for detecting and responding to potential resource conflicts, which are necessary ingredients for configuring a PCP-based scheduling procedure. We have also illustrated their applicability to the resource constrained project scheduling problem with minimum and maximum lag times (RCPSP/max). Due to space considerations, we have not focused heavily on the issue of optimization of project schedules. However, it is important to note that the PCP approach we have described can be directly embedded as a sub-procedure in various forms of extended optimizing search (see Cesta et al. 2002; Oddi et al. 2010a; Policella et al. 2009).

Minimization of project makespan, for example, is an objective that co-exists very naturally with a *POS* solution—In the best case, the early start time solution which minimizes makespan is executed; but in the event that unexpected events prevent this possibility, the accompanying *POS* provides a feasible, bounded relaxation strategy. Overall, PCP provides a flexible and efficient framework for solving complex combinatorial problems like project scheduling.

Acknowledgements Authors would like to thank the anonymous reviewers for detailed comments to previous drafts of this chapter. Stephen Smith was supported in part by the US Air Force

Research Laboratory under contracts FA8650-12-C-6269 and FA8750-12-C-0068, and the CMU Robotics Institute. CNR authors were supported by internal funds for basic research.

References

Adams J, Balas E, Zawack D (1988) The shifting bottleneck procedure for job shop scheduling. Manage Sci 34(3):391–401

Baptiste P, Le Pape C (1995) A theoretical and experimental comparison of constraint propagation techniques for disjunctive scheduling. In: IJCAI-95. Morgan Kaufmann, San Francisco, pp 600–606

Baptiste P, Le Pape C, Nuijten W (2001) Constraint-based scheduling. Kluwer, Boston

Bartusch M, Mohring RH, Radermacher FJ (1988) Scheduling project networks with resource constraints and time windows. Ann Oper Res 16(1):201–240

Beck JC, Davenport AJ, Davis ED, Fox MS (1998) The ODO project: towards a unified basis for constraint-directed scheduling. J Sched 1:89–125

Cesta A, Stella C (1997) A time and resource problem for planning architectures. In: ECP-97. Lecture notes in computer science, vol 1348. Springer, New York, pp 117–129

Cesta A, Oddi A, Smith SF (1999) An iterative sampling procedure for resource constrained project scheduling with time windows. In: IJCAI-99. Morgan Kaufmann, San Francisco, pp 1022–1029

Cesta A, Oddi A, Smith SF (2000) Iterative flattening: a scalable method for solving multi-capacity scheduling problems. In: AAAI-00. AAAI Press, Menlo Park, pp 742–747

Cesta A, Oddi A, Smith SF (2002) A constraint-based method for project scheduling with time windows. J Heuristics 8(1):109–136

Davenport AJ, Gefflot C, Beck JC (2001) Slack-based techniques for robust schedules. In: ECP-01. Lecture notes in computer science. Springer, Heidelberg, pp 7–18

Dechter R, Rossi F (2002) Constraint satisfaction. In: Nadel L (ed) Encyclopedia of cognitive science, Nature Publishing Group, London

Dechter R, Meiri I, Pearl J (1991) Temporal constraint networks. Artif Intell 49(1–3):61–95

Drabble B, Tate A (1994) The use of optimistic and pessimistic resource profiles to inform search in an activity based planner. In: AIPS-94. AAAI Press, Menlo Park, pp 243–248

Drummond M, Bresina J, Swanson K (1994) Just-in-case scheduling. In: AAAI-94. AAAI Press, Menlo Park, pp 1098–1104

El Sakkout HH, Wallace MG (2000) Probe backtrack search for minimal perturbation in dynamic scheduling. Constraints 5(4):359–388

Fox MS (1990) Constraint guided scheduling: a short history of scheduling research at CMU. Comp Ind 14(1–3):79–88

Hentenryck PV, Michel L (2009) Constraint-based local search. MIT Press, Cambridge

Hiatt LM, Zimmerman TL, Smith SF, Simmons R (2009) Strengthening schedules through uncertainty analysis agents. In: IJCAI-09. AAAI Press, Menlo Park

Kumar V (1992) Algorithms for constraint-satisfaction problems: a survey. Artif Intell Mag 13(1):32–44

Laborie P (2003) Algorithms for propagating resource constraints in A.I. planning and scheduling: existing approaches and new results. Artif Intell 143(2):151–188

Laborie P, Ghallab M (1995) Planning with sharable resource constraints. In: IJCAI-95. Morgan Kaufmann, San Francisco, pp 1643–1651

Laborie P, Godard D (2007) Self-adapting large neighborhood search: application to single-mode scheduling problems. In: Proceedings MISTA-07, Paris, pp 276–284

Montanari U (1974) Networks of constraints: fundamental properties and applications to picture processing. Inform Sci 7:95–132

Muscettola N (2002) Computing the envelope for stepwise-constant resource allocations. In: CP-2002. Lecture notes in computer science, vol 2470. Springer, Heidelberg, pp 139–154

Nuijten WPM, Aarts EHL (1996) A computational study of constraint satisfaction for multiple capacitated job shop scheduling. Eur J Oper Res 90(2):269–284

Nuijten W, Le Pape C (1998) Constraint-based job shop scheduling with ILOG-scheduler. J Heuristics 3(4):271–286

Oddi A, Cesta A, Policella N, Smith S (2010a) Iterative flattening search for resource constrained scheduling. J Intell Manuf 21(1):17–30

Oddi A, Rasconi R, Cesta A (2010b) Project scheduling as a disjunctive temporal problem. In: ECAI 2010. IOS Press, Amsterdam, pp 967–968

Policella N (2005) Scheduling with uncertainty: a proactive approach using partial order schedules. Ph.D. dissertation, Department of Computer and Systems Science, University of Rome "La Sapienza", Rome

Policella N, Smith SF, Cesta A, Oddi A (2004) Generating robust schedules through temporal flexibility. In: ICAPS'04. AAAI Press, Menlo Park, pp 209–218

Policella N, Cesta A, Oddi A, Smith S (2007) From precedence constraint posting to partial order schedules: a CSP approach to robust scheduling. AI Commun 20(3):163–180

Policella N, Cesta A, Oddi A, Smith S (2009) Solve-and-robustify. J Sched 12(3):299–314

Rossi F, van Beek P, Walsh T (2006) Handbook of constraint programming. Foundations of artificial intelligence, Elsevier Science, Amsterdam

Roy B, Sussman B (1964) Les problemes d'ordonnancement avec contraintes disjonctives, note DS n. 9 bis. SEMA, Paris

Sadeh NM (1991) Look-ahead techniques for micro-opportunistic job shop scheduling. Ph.D. dissertation, School of Computer Science, Carnegie Mellon University, Pittsburgh

Shaw P (1998) Using constraint programming and local search methods to solve vehicle routing problems. In: CP98. Lecture notes in computer science, vol 1520. Springer, Berlin, pp 417–431

Smith SF (1994) OPIS: a methodology and architecture for reactive scheduling. In: Fox M, Zweben M (eds) Intelligent scheduling, Morgan Kaufmann, San Francisco, pp 29–66

Smith SF, Cheng C (1993) Slack-based heuristics for constraint satisfactions scheduling. In: AAAI-93. AAAI Press, Menlo Park, pp 139–144

Smith TB, Pyle JM (2004) An effective algorithm for project scheduling with arbitrary temporal constraints. In: AAAI'04. AAAI Press, Menlo Park, pp 544–549

Tsang EPK (1993) Foundations of constraint satisfaction. Academic Press, London/San Diego

Chapter 7
A Satisfiability Solving Approach

Andreas Schutt, Thibaut Feydy, Peter J. Stuckey, and Mark G. Wallace

Abstract Boolean satisfiability solving is a powerful approach for testing the feasibility of propositional logic formulae in conjunctive normal form. Nowadays, Boolean satisfiability solvers efficiently handle problems with millions of clauses and hundreds of thousands of Boolean variables. But still many combinatorial problems such as resource-constrained project scheduling are beyond their capabilities. However, hybrid solution approaches have recently been proposed for resource-constrained project scheduling with generalized precedence constraints ($PS|temp|C_{max}$) that incorporate advanced Boolean satisfiability technology such as nogood learning and conflict-driven search. In this chapter, we present a generic exact method for $PS|temp|C_{max}$ using one of the most successful hybrid approaches called lazy clause generation. This approach combines constraint programming solving with Boolean satisfiability solving.

Keywords Generalized precedence relations • Lazy clause generation • Makespan minimization • Project scheduling • Resource constraints

A. Schutt (✉) • T. Feydy • P.J. Stuckey
National ICT Australia & Computing and Information Systems, University of Melbourne, Melbourne, VIC, Australia
e-mail: andreas.schutt@nicta.com.au; thibaut.feydy@nicta.com.au; peter.stuckey@nicta.com.au

Mark G. Wallace
Opturion & Faculty of Information Technology, Monash University, Caulfield, VIC, Australia
e-mail: mark.wallace@monash.edu

C. Schwindt, J. Zimmermann (eds.), *Handbook on Project Management and Scheduling Vol. 1*, International Handbooks on Information Systems,
DOI 10.1007/978-3-319-05443-8_7

7.1 Introduction

The resource-constrained project scheduling problem with generalized precedence relations[1] consists of scarce resources, activities, and precedence relations between pairs of activities, where activities require some resources during their execution. The goal is to create a schedule of activities within the planning period such that the resource usage does not exceed the resource capacity and all precedence relations are satisfied. In this chapter, we restrict ourselves to project scheduling problems on *renewable* resources (i.e., the resource capacity is constant during the planning period), *non-preemptive* activities (i.e., no interruption is allowed during execution), and finding a schedule with a minimal project duration. This problem is denoted as $PS|temp|C_{max}$ in Brucker et al. (1999). Bartusch et al. (1988, Theorem 3.10) show that the feasibility problem, whether an instance is feasible given an unlimited project duration, is $\mathscr{NP}$-hard.

$PS|temp|C_{max}$ is a very general problem. Practical scheduling problems can include substantially varied restrictions on the resources and activities. The following restrictions can be modeled with generalized precedence relations: minimal and maximal overlaps of activities, synchronization of start or end times for activities, change of the resource requirement during the activity's execution, fixed start times of activities, setup times, or non-delay execution of activities (see, e.g., Bartusch et al. 1988; Dorndorf et al. 2000; Neumann and Schwindt 1997). Moreover, a variation of the resource availability over time can be modeled by adding fictitious activities.

$PS|temp|C_{max}$ is widely studied and some of its applications can be found in Bartusch et al. (1988), e.g., civil engineering, building projects, and processor scheduling. A problem instance consists of a set of resources, a set of activities, and a set of generalized precedence constraints between activities. Each resource is characterized by its integral capacity, and each activity by its integral duration and its resource requirements. Generalized precedence relations express relations of start-to-start, start-to-end, end-to-start, and end-to-end times between pairs of activities. All these relations can be formulated as start-to-start time precedences. They have the form $S_i + \delta_{ij} \leq S_j$ where S_i and S_j are the start times of the activities i and j, respectively, and δ_{ij} is an integral distance between them. If $\delta_{ij} \geq 0$ this imposes a *minimal time lag*, while if $\delta_{ij} < 0$ this imposes a *maximal time lag* between start times.

Example 7.1. A simple example of an $PS|temp|C_{max}$ problem consists of the five activities with start times S_1, S_2, S_3, S_4, and S_5, durations $2, 5, 3, 1$, and 2 and resource requirements on a single resource $3, 2, 1, 2$, and 2 with a resource capacity of 4. Suppose we also have the generalized precedence relations $S_1 + 2 \leq S_2$ (activity 1 ends before activity 2 starts), $S_2 + 1 \leq S_3$ (activity 2 starts at least 1

[1] In the literature, generalized precedence relations are also known as temporal precedences, arbitrary precedences, minimal and maximal time lags, or time windows.

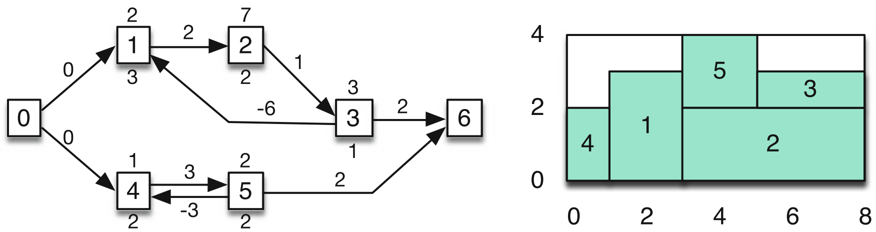


Fig. 7.1 The Activity-on-Node project network and the Gantt chart of a solution to a small $PS|temp|C_{max}$

time unit before activity 3 starts), $S_3 - 6 \leq S_1$ (activity 3 cannot start later than 6 time units after activity 1 starts), $S_4 + 3 \leq S_5$ (activity 4 starts at least 3 time units before activity 5 starts), and $S_5 - 3 \leq S_4$ (activity 5 cannot start later than 3 time units after activity 4 starts). Note that the last two precedence relations express the relation $S_4 + 3 = S_5$ (activity 4 starts exactly 3 time units before activity 5).

Let the maximal project duration, in which all activities must be completed, be 8. Figure 7.1 illustrates the Activity-on-Node project network between the five tasks and source 0 at the left (time 0) and sink 6 at the right (time 8), as well as a potential solution to this problem in the Gantt chart, where a rectangle for activity i has width equal to its duration and height equal to its resource requirements.

Note that additional edges between the source (sink) are drawn in the Activity-on-Node project network. These edges reflect the constraints that the activities must be executed in the planning period created by start time of the source and end time of the sink, which is between time 0 and time 8. □

Solution approaches that incorporate advanced Boolean satisfiability (SAT) technology based on the Davis-Putnam-Logemann-Loveland (DPLL) procedure for solving resource-constrained project scheduling are relatively new. In 2009, Schutt et al. present a generic exact solution approach using lazy clause generation (LCG) (Ohrimenko et al. 2009) for the basic resource-constrained project scheduling problem ($PS|prec|C_{max}$) that only involves "standard" precedence relations. LCG is a constraint programming (CP) approach with underlying advanced SAT technology from which it inherits the advantage of nogood learning for pruning the search space and conflict driven search for search guidance. This first approach (Schutt et al. 2009) modeled the resource constraints by decomposition. Later in Schutt et al. (2011, 2013a), LCG was extended with the global constraint `cumulative` which natively models resource constraints and performs better as the size of the problem grows. These approaches were then adopted for $PS|temp|C_{max}$ in Schutt et al. (2013b) and $PS|prec|\Sigma c_i^F \beta^{C_i}$ in Schutt et al. (2012) (or see Chap. 14 of this handbook).

In 2010, Horbach proposes a hand-tailored exact solution approach for solving $PS|prec|C_{max}$. He modifies a SAT solver so that the resource constraints are handled outside by decomposition. This approach is very similar to a satisfiability modulo theory solver. In the same year, Berthold et al. (2010) showed a generic exact solution approach for $PS|prec|C_{max}$ using the constraint integer solver SCIP (Achterberg 2009) which combines CP, SAT, and integer programming and handles resource constraint by the global constraint `cumulative`. This approach was then adopted for $PS|temp|C_{max}$ (Heinz et al. 2013). In 2011, Ansótegui et al. present a generic exact solution approach for $PS|prec|C_{max}$ that uses the satisfiability modulo theory solver Yices (Dutertre and de Moura 2006) and handles resource constraints by decomposition. Although, the approaches Horbach (2010) and Ansótegui et al. (2011) only have been applied on $PS|prec|C_{max}$ they can be extended for $PS|temp|C_{max}$.

All these approaches benefit from the underlying SAT technology and perform similarly on $PS|prec|C_{max}$ whereas the LCG approach (Schutt et al. 2011) seems to have the edge. The main differences between the approaches are the Boolean representation of the possible start times of activities, the model of resource constraints, and the optimization strategy, i.e., branch and bound or dichotomic search. Despite that, all these approaches outperform exact solution approaches that do not take advantage of SAT technologies.

In this chapter, we present a generic exact solution approach that incorporates advanced SAT technology, on the example of the LCG approach described in Schutt et al. (2013b). We show that the approach to solving $PS|temp|C_{max}$ performs better than published methods so far, especially for improving an initial solution once a solution is found, and proving optimality. We state the limitations of our current model and how to overcome them. We compare our approach to the best known approaches to $PS|temp|C_{max}$ on standard benchmark suites.

For that we give an overview in finite domain propagation (which is a part of CP only dealing with variables having finite domains), SAT solving, and LCG solving in Sect. 7.2. Then, in Sect. 7.3 we present a basic model for $PS|temp|C_{max}$ and discuss some improvements to it. In Sect. 7.4, we discuss the various branch-and-bound procedures that we use to search for optimal solutions. In Sect. 7.6, we compare our algorithm to the best approaches we are aware of on three challenging benchmark suites. Finally, we conclude in Sect. 7.7.

7.2 Preliminaries

In this section, we explain LCG by first introducing finite domain propagation and DPLL-based SAT solving. Then we describe the hybrid approach. We discuss how the hybrid explains conflicts and briefly discuss how a cumulative propagator is extended to explain its propagations.

Before starting with the introduction, we restrict the definitions of intervals to the set of integers in this chapter because $PS|temp|C_{max}$ is a combinatorial problem where real numbers do not play any role.

$$[a, b[:= \{x \in \mathbb{Z} \mid a \le x < b\}$$

$$]a,b] := \{x \in \mathbb{Z} \mid a < x \leq b\}$$
$$[a,b] := \{x \in \mathbb{Z} \mid a \leq x \leq b\}$$

In the remainder, we also referred these intervals of integers as *ranges* of integers.

7.2.1 Finite Domain Propagation

Finite domain (FD) propagation (see, e.g., Marriott and Stuckey 1998; Tsang 1993) is a powerful approach to tackling combinatorial problems. An FD problem $(\mathfrak{C}, D)$ consists of a set of constraints $\mathfrak{C}$ over a set of variables X, and a domain D, which determines the finite set of possible values of each variable in X. A *domain* D is a complete mapping from X to finite sets of integers. Hence, given domain D, then $D(x)$ is the set of possible values that variable $x \in X$ can take. Let $\min_D(x) := \min(D(x))$ and $\max_D(x) := \max(D(x))$. In this chapter, we will focus on domains where $D(x)$ is a range for all $x \in X$. The *initial domain* is referred as D^0. Let D_1 and D_2 be two domains, then D_1 is *stronger* than D_2, written $D_1 \sqsubseteq D_2$, if $D_1(x) \subseteq D_2(x)$ for all $x \in X$. Similarly, if $D_1 \sqsubseteq D_2$ then D_2 is *weaker* than D_1. For instance, all domains D that occur will be stronger than the initial domain, i.e., $D \sqsubseteq D^0$. A *false domain* is a domain D that maps at least one variable x to the empty set, i.e., $\exists x \in X$ with $D(x) = \emptyset$.

A *valuation* θ is a mapping of variables to values, written $\{x_1 \mapsto d_1, \ldots, x_n \mapsto d_n\}$. We extend the valuation θ to map expressions or constraints involving the variables in the natural way. Let *vars* be the function that returns the set of variables appearing in an expression, constraint or valuation. In an abuse of notation, we define a valuation θ to be an element of a domain D, written $\theta \in D$, if $\theta(x) \in D(x)$ for all $x \in vars(\theta)$. Note that a false domain D has no element valuations. A *valuation domain* D is a domain where $|D(x)| = 1, \forall x \in X$. We can define the corresponding valuation θ_D for a valuation domain D as $\{x \mapsto d \mid D(x) = \{d\}, x \in X\}$.

A constraint $c \in \mathfrak{C}$ is a set of valuations over $vars(c)$ which give the permissible values for a set of variables. In FD solvers, constraints are implemented by propagators. A *propagator* g implementing c is an inclusion-decreasing function on domains such that for all domains $D \sqsubseteq D^0$: $g(D) \sqsubseteq D$ and no solutions are lost, i.e., $\{\theta \in D \mid \theta \in c\} = \{\theta \in g(D) \mid \theta \in c\}$. We assume each propagator g is *checking*, that is if D is a valuation domain then $g(D) = D$ if and only if θ_D restricted to $vars(c)$ is a solutions of c. Conversely, $g(D)$ is a false domain if and only if θ_D restricted to $vars(c)$ is not a solution of c. Given a set of constraints $\mathfrak{C}$ we assume a corresponding set of propagators $F := \{g \mid c \in \mathfrak{C}, g \text{ implements } c\}$.

A *propagation solver* $solv(F, D)$ for a set of propagators F and current domain D repeatedly applies all the propagators in F starting from domain D until there is no further change in resulting domain. $solv(F, D)$ is some weakest domain $D' \sqsubseteq D$ which is a fixed point (i.e., $g(D') = D'$) for all $g \in F$.

FD solving interleaves propagation with search decisions. Given an initial problem ($\mathfrak{C}$,D) where F are the propagators for the constraints $\mathfrak{C}$, we first run the propagation solver $solv(F, D) = D'$. If this determines failure (D' is a false domain) then the problem has no solution and we backtrack to visit the next unexplored choice. If D' is a valuation domain then we have determined a solution. Otherwise we pick a variable $x \in X$ where $|D'(x)| \geq 2$ and split its domain $D'(x)$ into two disjoint parts $U_1 \cup U_2 = D'(x)$ creating two subproblems $(\mathfrak{C}, D_1)$ and $(\mathfrak{C}, D_2)$, where $D_i(x) = U_i$ and $D_i(v) = D'(v), v \neq x$, whose solutions are also solutions of the original problem. We then recursively explore the first problem, and when we have shown it has no solutions we explore the second problem.

As defined above FD propagation is only applicable to *satisfaction problems*. FD solvers solve optimization problems by mapping them to repeated satisfaction problems. Given an objective function f to minimize under constraints $\mathfrak{C}$ with domain D, the finite domain solving approach first finds a solution θ to $(\mathfrak{C}, D)$, and then finds a solution to $(\mathfrak{C} \cup \{f < \theta(f)\}, D)$, that is, the satisfaction problem of finding a better solution than previously found. It repeats this process until a problem is reached with no solution, in which case the last found solution is optimal. If the process is halted before proving optimality, then the solving process just returns the last solution found as the best known. We note that a dichotomic search on the values of objective function values is also possible, and requires solving fewer satisfaction problems to optimally solve the problem, but it does not allow the reuse of all nogoods between the different satisfaction problems.

FD propagation is a powerful generic approach to solving combinatorial optimization problems. Its chief strengths are the ability to model problems at a very high level, and the use of global propagators, that is specialized propagation algorithms, for important constraints.

7.2.1.1 Generalized Precedence Constraints

A *binary inequality propagator* g for a generalized precedence relation $x + d \leq y$ updates the domains of x and y in constant time as follows

$$g(D)(u) := \begin{cases} D(x) \cap\,]-\infty, \max_D(y) - d] & \text{if } u = x \\ D(y) \cap [\min_D(x) + d, \infty[& \text{if } u = y \\ D(u) & \text{otherwise} \end{cases}$$

where $x, y, u \in X$. Hence, the propagator infers a new upper bound on x if $\max_D(y) - d < \max_D(x)$ and a new lower bound on y if $\min_D(x) + d > \min_D(y)$.

7.2.1.2 Cumulative Resource Constraints

Beside generalized precedence relations $PS|temp|C_{max}$ involves resources with a limited resource capacities which cannot be exceeded in any time during the planning horizon. In CP, such resource restrictions are called *cumulative (resource) constraints*. These constraints can be modeled by decompositions or the global propagator/constraint `cumulative` which offers stronger and more efficient pruning algorithms. Since the approach of Schutt et al. (2013b) uses `cumulative` we concentrate on this global propagator.

The constraint `cumulative` introduced by Aggoun and Beldiceanu (1993) imposes the resource restrictions for one scarce resource $k \in \mathscr{R}$ by

$$\texttt{cumulative}([S_i \mid i \in V], [p_i \mid i \in V], [r_{ik} \mid i \in V], R_k) \equiv$$
$$\sum_{i \in V} r_{ik}(t) \leq R_k \quad (t \in [0, T[)$$

where T is the initial upper bound on S_{n+1}, i.e., $T := \max_{D^0}(S_{n+1})$, and $r_{ik}(t)$ is the resource usage of activity i on resource k at time t, i.e., it equals to r_{ik} if i runs at time t and 0 otherwise. Recall that S_i, p_i, and r_{ik} respectively are the start time variable, the fixed duration, and the fixed resource usage of activity i and all values are integral whereas p_i and r_{ik} are non-negative.

Example 7.2. Consider the five activities from Example 7.1 with durations 2, 5, 3, 1, and 2 and resource usages 3, 2, 1, 2, and 2 and a resource capacity of 4. This is represented by the cumulative constraint.

$$\texttt{cumulative}([S_1, S_2, S_3, S_4, S_5], [2, 5, 3, 1, 2], [3, 2, 1, 2, 2], 4)$$

Imagine each task must start at time 0 or after and finish before time 8. The cumulative problem corresponds to scheduling the activities shown in Fig. 7.2a into the Gantt chart shown to the left. □

There are many propagation algorithms for the cumulative constraint (see, e.g., Baptiste et al. 2001; Mercier and Van Hentenryck 2008; Schutt and Wolf 2010; Vilím 2011), but the most widely used for project scheduling problems is based on timetable propagation (see, e.g., Le Pape 1994), because generalized precedence relations usually impose an order between many pairs of activities so that not many activities can run concurrently. In the following, the timetable propagation is described.

An activity i has a *compulsory part* given domain D from $[\max_D S_i, \min_D S_i + p_i[$, that requires that activity i makes use of r_{ik} resources at each of the times in $[\max_D S_i, \min_D S_i + p_i[$ if the range is non-empty. The timetable propagator for `cumulative` first determines the *resource (usage) profile*, i.e., it determines $r_k(S, t)$ which is the minimal amount of resource k used at time t with respect to current domain bounds of the start time variables S_i. If at some time t the profile exceeds the resource capacity, i.e., $r_k(S, t) > R_k$, the constraint is violated and failure detected. If at some time t the resources used in the profile are such that there is not enough left for an activity i, i.e., $r_k(S, t) + r_{ik} > R_k$, then we can determine that activity i cannot be scheduled to run during time t. If the earliest start

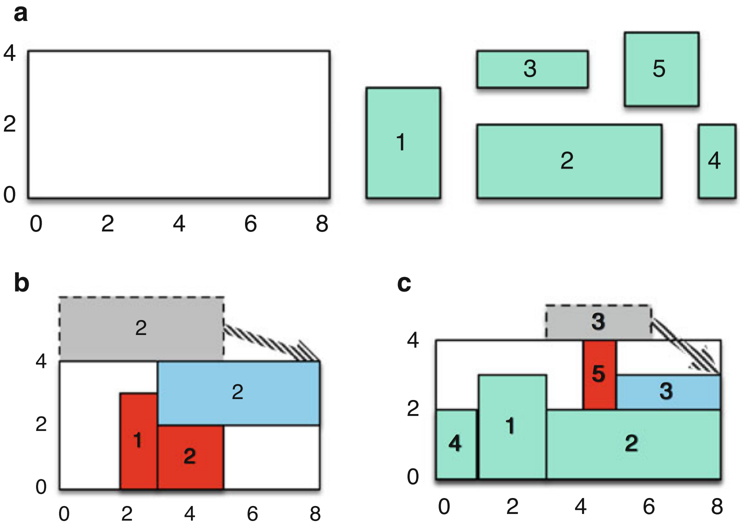


Fig. 7.2 Figure illustrates the timetable propagation of the cumulative constraint for activities 2 and 3 where *dark boxes* describe compulsory parts of unfixed activities

time $\min_D S_i$ of activity i, is such that the activity cannot be scheduled completely before time t, i.e., $\min_D S_i + p_i > t$, we can update the earliest start time to be $t + 1$, similarly if the latest start time of the activity is such that the activity cannot be scheduled completely after t, i.e., $\max_D S_i \leq t$, then we can update the latest start time to be $t - p_i$. For a full description of timetable propagation for `cumulative` (see, e.g., Schutt et al. 2011).

Example 7.3. Consider the cumulative constraint of Example 7.2. We assume that the domains of the start times are $D(S_1) := [1,2]$, $D(S_2) := [0,3]$, $D(S_3) := [3,5]$, $D(S_4) := [0,2]$, $D(S_5) := [0,4]$. Then there are compulsory parts of activities 1 and 2 in the ranges $[2,3[$ and $[3,5[$ respectively shown in Fig. 7.2b in red (dark). No other activities have a compulsory part. Hence the red contour illustrates the resource usage profile. Since activity 2 cannot be scheduled in parallel with activity 1, and the earliest start time of activity 2, which is 0, means that the activity cannot be scheduled before activity 1 we can reduce the domain of the start time for activity 2 to the singleton $[3,3]$. This is illustrated in Fig. 7.2b. The opposite holds for activity 1 that cannot be run after activity 2, hence the domain of its start time shrinks to the singleton range $[1,1]$. Once we make these changes the compulsory parts of the activities 1 and 2 increase to the ranges $[1,3[$ and $[3,8[$ respectively. This in turn causes the start times of activities 4 and 5 to become $[0,0]$ and $[3,4]$ respectively, creating compulsory parts in the ranges $[0,1[$ and $[4,5[$ respectively. The latter causes the start time of activity 3 to become fixed at 5 generating the compulsory part in $[5,8[$ which causes that the start time of activity 5 becomes fixed at 3. This is illustrated in Fig. 7.2c. In this case the timetable propagation results in a final schedule in the right of Fig. 7.1. □

7.2.2 Boolean Satisfiability Solving

Let $\mathscr{B}$ be a set of Boolean variables. A *literal* l is either a Boolean variable $b \in \mathscr{B}$, i.e., $l \equiv b$, or its negation, i.e., $l \equiv \neg b$. The negation of a literal $\neg l$ is defined as $\neg b$ if $l \equiv b$ and b if $l \equiv \neg b$. A *clause* c is a set of literals understood as a disjunction. Hence clause $\{l_1, \ldots, l_n\}$ is satisfied if at least one literal l_i is true. An *assignment* $\mathfrak{A}$ is a set of Boolean literals that does not include a variable and its negation, i.e., $\nexists b \in \mathscr{B} : \{b, \neg b\} \subseteq \mathfrak{A}$. An assignment can be seen as a partial valuation on Boolean variables, $\{b \mapsto \mathit{true} \,|b \in \mathfrak{A}\} \cup \{b \mapsto \mathit{false} \mid \neg b \in \mathfrak{A}\}$. A theory T is a set of clauses. A Sat *problem* $(T, \mathfrak{A})$ consists of a set of clauses T and an assignment $\mathfrak{A}$ over (some of) the variables occurring in T. Thus, the assignment $\mathfrak{A}$ can be a partial, possible empty, assignment. A solution for the theory T is an assignment $\mathfrak{A}$ containing each Boolean variable in either a positive or negative context, and satisfying all clauses in T. Consequently, a solution for a Sat problem $(T, \mathfrak{A})$ is a solution $\mathfrak{A}'$ of the theory T that is a superset of $\mathfrak{A}$, i.e., $\mathfrak{A}' \supseteq \mathfrak{A}$.

A Sat solver based on the DPLL procedure (Davis and Putnam 1960; Davis et al. 1962) is a form of FD propagation solver specialized for Boolean clauses. Each clause is propagated by so-called *unit propagation*. Given an assignment $\mathfrak{A}$, unit propagation detects failure using clause c if $\{\neg l \mid l \in c\} \subseteq \mathfrak{A}$, and unit propagation detects a new unit consequence l if $c \equiv \{l\} \cup c'$ and $\{\neg l' \mid l' \in c'\} \subseteq \mathfrak{A}$, in which case it adds l to the current assignment $\mathfrak{A}$. Unit propagation continues until failure is detected, or no new unit consequences can be determined.

Sat solvers exhaustively apply unit propagation to the current assignment $\mathfrak{A}$ to generate all the consequences resulting in a new assignment $\mathfrak{A}'$. They then choose an unfixed variable b and create two equivalent problems $(T, \mathfrak{A}' \cup \{b\})$, $(T, \mathfrak{A}' \cup \{\neg b\})$ and recursively search these subproblems. The literals added to the assignment by choice are termed *decision literals*.

Modern DPLL-based Sat solving is a powerful approach to solving combinatorial optimization problems because it records nogoods that prevent the search from revisiting similar parts of the search space. The Sat solver records an explanation for each unit consequence discovered (the clause that caused unit propagation), and on failure uses these explanations to determine a set of mutually incompatible decisions, a *nogood*, which is added as a new clause to the theory of the problem. These nogoods drastically reduce the size of the search space needed to be examined. Another advantage of Sat solvers is that they track which variables are involved in the most failures (called *active* variables), and use a powerful autonomous search procedure which concentrates on the variables that are most active. The disadvantages of Sat solvers are the restriction to Boolean variables and the sometime huge models that are required to represent a problem using only clauses.

7.2.3 *Lazy Clause Generation*

LCG is a hybrid of FD propagation and SAT solving. The key idea in LCG is to run a FD propagation solver, but to instrument its execution in order to build an explanation of the propagations made by the solver. These are recorded as clauses on a Boolean variable representation of the problem. Hence, as the FD search progresses, we lazily create a clausal representation of the problem. The hybrid has the advantages of FD solving, but inherits the SAT solvers ability to create nogoods to drastically reduce search and use activity based search.

7.2.3.1 Variable Representation

An LCG problem is stated as an FD problem, but each integer variable has a clausal representation in the SAT solver. In the remainder of this work, we use $[\![.]\!]$ as the *names* of Boolean variables. An integer variable $x \in \mathit{vars}$ with the initial domain $D^0(x) := [l, u]$ is represented by $2(u - l) + 1$ Boolean variables $[\![x = l]\!]$, $[\![x = l + 1]\!], \ldots, [\![x = u]\!]$ and $[\![x \leq l]\!], [\![x \leq l + 1]\!], \ldots, [\![x \leq u - 1]\!]$. The variable $[\![x = d]\!]$ is *true* if x takes the value d, and *false* if x takes a value different from d. Similarly, the variable $[\![x \leq d]\!]$ is true if x takes a value less than or equal to d and false for a value greater than d.

We use the notation $[\![d \leq x]\!]$ to refer to the literal $\neg[\![x \leq d - 1]\!]$.

Not every assignment of Boolean variables is consistent with the integer variable x, for example $\{[\![x = 3]\!], [\![x \leq 2]\!]\}$ (i.e., both Boolean variables are true) requires that x is both 3 and ≤ 2. In order to ensure that assignments represent a consistent set of possibilities for the integer variable x we add to the SAT solver the clauses $\mathit{DOM}(x)$ that encode

$$[\![x \leq d]\!] \rightarrow [\![x \leq d + 1]\!] \qquad (l \leq d < u - 1) \tag{7.1}$$

$$[\![x = l]\!] \leftrightarrow [\![x \leq l]\!] \tag{7.2}$$

$$[\![x = d]\!] \leftrightarrow ([\![x \leq d]\!] \wedge \neg[\![x \leq d - 1]\!]) \qquad (l < d < u) \tag{7.3}$$

$$[\![x = u]\!] \leftrightarrow \neg[\![x \leq u - 1]\!] \tag{7.4}$$

where $D^0(x) = [l, u]$. This equates to $u–l–1$ clauses for Eq. (7.1) and $3(u–l–1)+4$ clauses for Eqs. (7.2)–(7.4). Note that clauses in Eqs. (7.2)–(7.4) are generated lazily on demand when propagation needs to express something using the literal $[\![x = d]\!]$ (see Feydy 2010 for details).

Any assignment $\mathfrak{A}$ on these Boolean variables can be converted to a domain:

$$\mathit{domain}(\mathfrak{A})(x) := \{d \in D^0(x) \mid \forall [\![c]\!] \in \mathfrak{A}, \mathit{vars}([\![c]\!]) = \{x\} : x = d \models c\}$$

i.e., the domain includes all values for x that are consistent with all the Boolean variables related to x. It should be noted that the domain may assign no values to some variable.

Example 7.4. Consider Example 7.1 and assume the initial domains $D^0(S_i) := [0, 15]$ for $i \in \{1, 2, 3, 4, 5\}$. The assignment $\mathfrak{A} = \{\neg[\![S_1 \leq 1]\!], \neg[\![S_1 = 3]\!], \neg[\![S_1 = 4]\!], [\![S_1 \leq 6]\!], \neg[\![S_2 \leq 2]\!], [\![S_2 \leq 5]\!], \neg[\![S_3 \leq 4]\!], [\![S_3 \leq 7]\!], \neg[\![S_5 \leq 3]\!]\}$ is consistent with $S_1 = 2$, $S_1 = 5$, and $S_1 = 6$. Therefore $\mathit{domain}(\mathfrak{A})(S_1) = \{2, 5, 6\}$. For the remaining variables $\mathit{domain}(\mathfrak{A})(S_2) = [3, 5]$, $\mathit{domain}(\mathfrak{A})(S_3) = [5, 7]$, $\mathit{domain}(\mathfrak{A})(S_4) = [0, 15]$, and $\mathit{domain}(\mathfrak{A})(S_5) = [4, 15]$. Note that for brevity $\mathfrak{A}$ is not a fixed point of unit propagation for $DOM(S_1)$ since we are missing many implied literals such as $\neg[\![S_1 = 0]\!]$, $\neg[\![S_1 = 8]\!]$ etc. □

7.2.3.2 Explaining Propagators

In LCG, a propagator is no longer simply a mapping from domains to domains, it is also a generator of clauses describing propagation. When $g(D) \neq D$ we assume the propagator g can determine a clause c to explain each domain change. Similarly, when $g(D)$ is a false domain the propagator must create a clause c that explains the failure.

Example 7.5. Consider the binary inequality propagator g for the precedence constraint $S_1+2 \leq S_2$ from Example 7.1. When applied to the domains $D(S_i) = [0, 15]$ for $i \in \{1, 2\}$ it obtains $g(D)(S_1) = [0, 13]$, and $g(D)(S_2) = [2, 15]$. The clausal explanation of the change in domain of S_1 is $[\![S_2 \leq 15]\!] \rightarrow [\![S_1 \leq 13]\!]$, similarly the change in domain of S_2 is $\neg[\![S_1 \leq -1]\!] \rightarrow \neg[\![S_2 \leq 1]\!]$ ($[\![0 \leq S_1]\!] \rightarrow [\![2 \leq S_2]\!]$). These become the clauses $\neg[\![S_2 \leq 15]\!] \vee [\![S_1 \leq 13]\!]$ and $[\![S_1 \leq -1]\!] \vee \neg[\![S_2 \leq 1]\!]$. □

The explanation clauses of the propagation are sent to the SAT solver on which unit propagation is performed. The clauses will always have the form $c \rightarrow l$ where c is a conjunction of literals true in the current assignment, and l is a literal not true in the current assignment, the newly added clause will always cause unit propagation, adding l to the current assignment.

Example 7.6. Consider the propagation from Example 7.5. The clauses $\neg[\![S_2 \leq 15]\!] \vee [\![S_1 \leq 13]\!]$ and $[\![S_1 \leq -1]\!] \vee \neg[\![S_2 \leq 1]\!]$ are added to the SAT theory. Unit propagation infers that $[\![S_1 \leq 13]\!]$ is *true* and $\neg[\![S_2 \leq 1]\!]$ is *true* since $\neg[\![S_2 \leq 15]\!]$ and $[\![S_1 \leq -1]\!]$ are *false*, and adds these literals to the assignment. Note that the unit propagation is not finished, since for example the implied literal $[\![S_1 \leq 14]\!]$, can be detected *true* as well. □

The unit propagation on the added clauses c is guaranteed to be as strong as the propagator f on the original domains. This means if $\mathit{domain}(\mathfrak{A}) \sqsubseteq D$ then $\mathit{domain}(\mathfrak{A}') \sqsubseteq g(D)$ where $\mathfrak{A}'$ is the resulting assignment after addition of c and unit propagation (see Ohrimenko et al. 2009).

Note that a single new propagation could be explained using different sets of clauses. In order to get maximum benefit from the explanation we desire a "strongest" explanation as possible. A set of clauses c_1 is *stronger* than a set of clauses c_2 if c_1 implies c_2. In other words, c_1 restricts the search space at least as much as c_2.

Example 7.7. Consider explaining the propagation of the start time of the activity 3 described in Example 7.3 and Fig. 7.2c. The domain change $\llbracket 5 \leq S_3 \rrbracket$ arises from the compulsory parts of activity 2 and 5 as well as the fact that activity 3 cannot start before time 3. An explanation of the propagation is hence $\llbracket 3 \leq S_3 \rrbracket \wedge \llbracket 3 \leq S_2 \rrbracket \wedge \llbracket S_2 \leq 3 \rrbracket \wedge \llbracket 3 \leq S_5 \rrbracket \wedge \llbracket S_5 \leq 4 \rrbracket \rightarrow \llbracket 5 \leq S_3 \rrbracket$. We can observe that if $2 \leq S_3$ then the same domain change $\llbracket 5 \leq S_3 \rrbracket$ follows due to the compulsory parts of activity 2 and 5. Therefore, a stronger explanation is obtained by replacing the literal $\llbracket 3 \leq S_3 \rrbracket$ by $\llbracket 2 \leq S_3 \rrbracket$.

Moreover, the compulsory parts of the activity 2 in the ranges $[3, 3]$ and $[5, 8]$ are not necessary for the domain change. We only require that there is not enough resources at time 4 to schedule task c. Thus the refined explanation can be further strengthened by replacing $\llbracket 3 \leq S_2 \rrbracket \wedge \llbracket S_2 \leq 3 \rrbracket$ by $\llbracket S_2 \leq 4 \rrbracket$ which is enough to force a compulsory part of S_2 at time 4. This leads to the stronger explanation $\llbracket 2 \leq S_3 \rrbracket \wedge \llbracket S_2 \leq 4 \rrbracket \wedge \llbracket 3 \leq S_5 \rrbracket \wedge \llbracket S_5 \leq 4 \rrbracket \rightarrow \llbracket 5 \leq S_3 \rrbracket$. □

In this example the final explanation corresponds to a pointwise explanation defined in Schutt et al. (2011). In this work, we use the timetable propagation, as earlier described in this section, that generates pointwise explanations. The maximum size of these explanations (*maxLenCumu*) is bounded by $2 \times \max\{T \subseteq \{1, 2, \ldots, n\} \mid \sum_{i \in V} r_{ik} > R_k$ and $\forall j \in V : \sum_{i \in V \setminus \{j\}} r_{ik} \leq R_k\}$ literals where n is the number of activities requiring some resource units of the renewable resource and R_k is the resource capacity of resource k. For a full discussion about the best way to explain propagation of `cumulative` see Schutt et al. (2011).

7.2.3.3 Nogood Generation

Since all propagation steps in LCG have been mapped to unit propagation on clauses, we can perform nogood generation just as in a SAT solver. Here, the nogood generation is based on an *implication graph* and the *first unique implication point* (1UIP). The graph is a directed acyclic graph where nodes represent fixed literals and directed edges reasons why a literal became *true*, and is extended as the search progresses. Unit propagation marks the literal it makes true with the clause that caused the unit propagation. The true literals are kept in a stack showing the order that they were determined as true by unit consequence or decisions.

For brevity, we do not differentiate between literals and nodes. A literal is fixed either by a search decision or unit propagation. In the first case, the graph is extended only by the literal and, in the second case, by the literal and incoming edges to that literal from all other literals in the clause on that the unit propagation assigned the *true* value to the literal (Fig. 7.3).

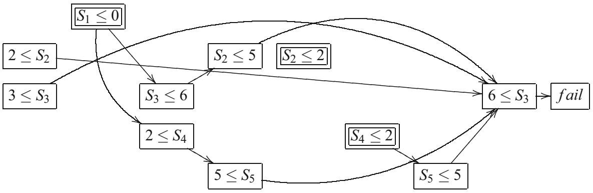


Fig. 7.3 (Part of) The implication graph for the propagation of Example 7.9 where decision literals are shown *double boxed*, while literals set by unit propagation are shown boxed

Example 7.8. Consider the strongest explanation $[\![2 \leq S_3]\!] \wedge [\![S_2 \leq 4]\!] \wedge [\![3 \leq S_5]\!] \wedge [\![S_5 \leq 4]\!] \rightarrow [\![5 \leq S_3]\!]$ from Example 7.7. It is added to the SAT database as clause $\neg[\![2 \leq S_3]\!] \vee \neg[\![S_2 \leq 4]\!] \vee \neg[\![3 \leq S_5]\!] \vee \neg[\![S_5 \leq 4]\!] \vee [\![5 \leq S_3]\!]$ and unit propagation sets $[\![5 \leq S_3]\!]$ *true*. Therefore the implication graph is extended by the edges $[\![2 \leq S_3]\!] \rightarrow [\![5 \leq S_3]\!]$, $[\![S_2 \leq 4]\!] \rightarrow [\![5 \leq S_3]\!]$, $[\![3 \leq S_5]\!] \rightarrow [\![5 \leq S_3]\!]$, and $[\![S_5 \leq 4]\!] \rightarrow [\![5 \leq S_3]\!]$. □

Every node and edge is associated with the search level at which they are added to the graph. Once a conflict occurs, a nogood which is the 1UIP in LCG is calculated based on the implication graph. A conflict is recognized when the unit propagation reaches a clause where all literals are false. This clause is the starting point of the analysis and builds a first tentative nogood. Literals in the tentative nogood are replaced one by one by the literals from their incoming edges, in the reverse order of their addition to the implication graph. This process continues until the tentative nogood contains exactly one literal associated with the current conflict search level. Thus, the time complexity of the nogood computation is bounded by the size of the extension of the implication graph at the conflict level. Given that in our case the timetable propagation of `cumulative` creates the largest explanations, the time complexity is bounded by *nprop* × *maxLenCumu* where *nprop* is the number of domain reductions performed in the conflict level and *maxLenCumu* is maximal length of an explanation for `cumulative`, which is described earlier. The resulting nogood is called the 1UIP nogood (Moskewicz et al. 2001).

Example 7.9. Consider the $PS|temp|C_{max}$ instance from Example 7.1 on page 136. Assume an initial domain of $D^0 := [0, 15]$ then after the initial propagation of the precedence constraints the domains are $D(S_1) = [0, 8]$, $D(S_2) = [2, 10]$, $D(S_3) = [3, 12]$, $D(S_4) = [0, 10]$, and $D(S_2) = [3, 13]$. Note that no tighter bounds can be inferred by the cumulative propagator.

Assume search now sets $S_1 \leq 0$. This sets the literal $[\![S_1 \leq 0]\!]$ as true, and unit propagation on the domain clauses sets $[\![S_1 = 0]\!]$, $[\![S_1 \leq 1]\!]$, $[\![S_1 \leq 2]\!]$ etc. In the remainder of the example, we will ignore propagation of the domain clauses and concentrate on the "interesting propagation."

The precedence constraint $S_3 - 6 \leq S_1$ forces $S_3 \leq 6$ with explanation $[\![S_1 \leq 0]\!] \rightarrow [\![S_3 \leq 6]\!]$. The precedence constraint $S_2 + 1 \leq S_3$ forces $S_2 \leq 5$ with explanation $[\![S_3 \leq 6]\!] \rightarrow [\![S_2 \leq 5]\!]$.

The timetable propagator for `cumulative` uses the compulsory part of activity 1 in $[0,2[$ to force $S_4 \geq 2$. The explanation for this is $[\![S_1 \leq 0]\!] \rightarrow [\![2 \leq S_4]\!]$. The precedence $S_4 + 3 \leq S_5$ forces $S_5 \geq 5$ with explanation $[\![S_4 \geq 2]\!] \rightarrow [\![5 \leq S_5]\!]$.

Suppose next that search sets $S_2 \leq 2$. It creates a compulsory part of 2 from $[2,7[$ but there is no propagation from precedence constraints or `cumulative`.

Suppose now that the search sets $S_4 \leq 2$. Then the precedence constraint $S_5 - 3 \leq S_4$ forces $S_5 \leq 5$ with explanation $[\![S_4 \leq 2]\!] \rightarrow [\![S_5 \leq 5]\!]$. This creates a compulsory part of 4 in $[2,3[$ and a compulsory part of 5 in $[5,7[$. In fact all the activities 1, 2, 4, and 5 are fixed now. Timetable propagation infers, since all resources are used at time 5, that activity 3 cannot start before time 6. A reason for this is $[\![2 \leq S_2]\!] \wedge [\![S_2 \leq 5]\!]$ (which forces 2 to use 2 resources in $[5,7[$), plus $[\![5 \leq S_5]\!] \wedge [\![S_5 \leq 5]\!]$ (which forces 5 to use 2 resources in $[5,7[$), plus $[\![3 \leq S_3]\!]$ (which forces 3 to overlap this time). Hence an explanation is $[\![2 \leq S_2]\!] \wedge [\![S_2 \leq 5]\!] \wedge [\![5 \leq S_5]\!] \wedge [\![S_5 \leq 5]\!] \wedge [\![3 \leq S_3]\!] \rightarrow [\![6 \leq S_3]\!]$.

This forces a compulsory part of 3 at time 6 which causes a resource overload at that time. An explanation of the failure is $[\![2 \leq S_2]\!] \wedge [\![S_2 \leq 5]\!] \wedge [\![5 \leq S_5]\!] \wedge [\![S_5 \leq 5]\!] \wedge [\![6 \leq S_3]\!] \wedge [\![S_3 \leq 6]\!] \rightarrow \mathit{false}$.

The nogood generation process starts from this original explanation of failure. It removes the last literal in the nogood by replacing it by its explanation. Replacing $[\![6 \leq S_3]\!]$ by its explanation creates the new nogood $[\![2 \leq S_2]\!] \wedge [\![S_2 \leq 5]\!] \wedge [\![5 \leq S_5]\!] \wedge [\![S_5 \leq 5]\!] \wedge [\![3 \leq S_3]\!] \wedge [\![S_3 \leq 6]\!] \rightarrow \mathit{false}$. Since this nogood has only one literal that was made true after the last decision $[\![S_5 \leq 5]\!]$ this is the 1UIP nogood. Rewritten as a clause it is $[\![S_2 \leq 1]\!] \vee \neg[\![S_2 \leq 5]\!] \vee [\![S_5 \leq 4]\!] \vee \neg[\![S_5 \leq 5]\!] \vee [\![S_3 \leq 2]\!] \vee \neg[\![S_3 \leq 6]\!]$.

Now the solver backtracks to the previous decision level, undoing the decision $S_4 \leq 2$ and its consequences. The newly added nogood unit propagates to force $S_5 \geq 6$ with explanation $[\![2 \leq S_2]\!] \wedge [\![S_2 \leq 5]\!] \wedge [\![5 \leq S_5]\!] \wedge [\![3 \leq S_3]\!] \wedge [\![S_3 \leq 6]\!] \rightarrow [\![6 \leq S_5]\!]$, and the precedence constraint $S_5 - 3 \leq S_4$ forces $S_4 \geq 3$ with explanation $[\![6 \leq S_5]\!] \rightarrow [\![3 \leq S_4]\!]$. Search proceeds looking for a solution. □

Nogoods generated by this process can have a size as large as the number of possible Boolean variables in the SAT representation, except that there can be at most two non-redundant inequality literals for each integer variable involved. Note that all generated nogoods encode redundant information, and we could delete any of them at any time, but also lose the search reduction that their propagation creates. In this chapter, all generated nogoods are kept permanently which requires space bounded by the maximal size of a nogood times the number of conflicts encountered during the search.

7.2.3.4 Lazy Clause Generation for Large Problems

For large problems, LCG can become inefficient if the domain sizes are large and the clause database contains a huge number of clauses. In the following, we briefly describe two improvements which tackle these issues.

If we have an initial domain $D^0(x) := [l, u]$ then the Boolean representation in the SAT solver is created upfront, which can consume a significant amount of memory if the size of the initial domain is large. Instead of this, we can always create the representation lazily by adding Boolean variables $[\![x \leq d]\!]$ or $[\![x = d]\!]$ and the corresponding clauses in the SAT solver when they appear in an explanation or nogood. Initially, the domain $D^0(x) = [l, u]$ is only represented by the global literals *false* and *true* which represent $[\![x \leq l - 1]\!]$ and $[\![x \leq u]\!]$ respectively. We add a Boolean variable $[\![x \leq d]\!]$, $d \in [l, u[$ whenever a clause, i.e., an explanation or a nogood, is added to the clause database that requires this Boolean variable. We also add clauses of the form $[\![x = d]\!]$. If at some stage during the search the integer variable is represented by more than one Boolean variable then a consistent assignment of these variables is maintained by the domain of the variable causing propagation, i.e., no clauses are added to the SAT forcing a consistent assignment. In the remainder, we refer to it as *lazy* variable representation.

In LCG an explanation clause is added to the clause database for each propagation and a nogood clause is added for each conflict. Thus, with each added clause, the performance of the unit propagation decays somewhat. The main use of an explanation is during conflict analysis for creating a strong nogood. A by-product of adding the explanation to the clause database is that the SAT solver can perform unit propagation on it. However, this propagation can also be done by the constraint propagator that built the explanation. Thus, we can exclude explanations for unit propagation in order to keep unit propagation more efficient. This is done by keeping the explanation in separate database and using the atomic constraints $x \leq d$, $x = d$, and their negations in the explanation.

In the remainder, we assume that these two improvements are not used unless stated.

7.3 Models for RCPSP/max

In this section, a basic model for $PS|temp|C_{max}$ is first presented and then a number of model improvements.

Given $PS|temp|C_{max}$ instance as a project network N with the weighted digraph (V, E, δ) and a set of resources $\mathscr{R}$. Recall that the nodes in V represent the activities and the arcs in E represent the generalized precedence relations in connection with the vector δ of arc weights. In order to obtained finite domains for the start time variables, an upper bound on the project duration must be imposed. We denote such an upper bound as T. A trivial upper bound can be computed by scheduling all activities in sequence without considering maximal time lags between activities, i.e., $T := \sum_{i \in V} \max(p_i, \max\{\delta_{ij} \mid (i, j) \in E\})$. Then the problem can be stated as follows:

$$\text{Min. } S_{n+1} \tag{7.5}$$

$$\text{s.t.}\quad S_i + \delta_{ij} \leq S_j \quad ((i,j) \in E) \tag{7.6}$$

$$\texttt{cumulative}([S_i \mid i \in V], [p_i \mid i \in V], [r_{ik} \mid i \in V], R_k) \quad (k \in \mathscr{R}) \tag{7.7}$$

$$S_i + p_i \leq S_{n+1} \quad (i \in V) \tag{7.8}$$

$$0 \leq S_i \leq T - p_i \quad (i \in V) \tag{7.9}$$

The objective is to minimize the project duration S_{n+1} Eq. (7.5) which is subjected to the generalized precedence constraints Eq. (7.6), the resource constraints Eq. (7.7), and the objective constraints Eq. (7.8). All start times must be non-negative and all activities must be scheduled in the planning period from 0 until T Eq. (7.9), which enforces an initial domain of $D^0(S_i) := [0, T - p_i]$.

A basic constraint model uses a binary inequality propagator for each precedence relation Eq. (7.6), one `cumulative` propagator for each resource constraint, and a binary inequality propagator for each objective constraint Eq. (7.8). Since none of the propagators either generate or use equality literals of the form $[\![x = d]\!]$ they and their defining clauses in $DOM(x)$ are never generated during execution. Hence, only $(n+1) \times T - \sum_{i \in V} p_i$ Boolean variables and $(n+1) \times (T-1) - \sum_{i \in V} p_i$ clauses are needed for the domain representation of the start time variables and the objective variable. Thus, the number of literals in a nogood is bounded to $2 \times n + 2$.

This basic model has a number of weaknesses: first the initial domains of the start times are large, second each precedence relation is modeled as one individual propagator, and finally the SAT solver in LCG has no structural information about activities in disjunction.

A smaller initial domain reduces the size of the problem because fewer Boolean variables are necessary to represent the integer domain in the SAT solver. It can be computed in a preprocessing step by taking into account the precedences in E as described in the next subsection. Individual propagators for precedence constraints may not be too costly for a small number of precedence constraints, but for a larger number of propagators, their queuing behavior may result in long and costly propagation sequences. A global propagator can efficiently adjust the time-bounds in $\mathscr{O}(n \log n + |E|)$ time (see Feydy et al. 2008) if the set of precedence constraints is feasible. Since the solver used herein does not offer this global propagator, an individual propagator is used for each precedence constraint.

7.3.1 Initial Domains

A smaller initial domain can be used for the start time variables if the lengths of the longest path from the dummy source to the activity $d_{0\,i}$ and from the activity to the dummy sink $d_{i,n+1}$ are known where the former path determines the earliest possible start time for i and the latter path the latest possible start time in any schedule. Then the initial domain of the start time variable S_i is $D^0(S_i) := [d_{0\,i}, T - d_{i,n+1}]$.

These longest paths can be obtained by applying the Bellman–Ford single source shortest path algorithm (see Bellman 1958; Ford and Fulkerson 1962) on the weighted digraph $(V, E, -\delta)$. Note that this digraph is the project network but with negated arc weights. Thus, the negated length of the shortest path between two nodes is the length of the longest path between them. If the digraph contains a negative-weight cycle then the $PS|temp|C_{max}$ instance is infeasible. The Bellman–Ford algorithm has a runtime complexity of $\mathcal{O}(|V| \times |E|)$.

These earliest and latest start times can not only be used for smaller initial domains, but also to improve the objective constraints by replacing them with

$$S_i + d_{i,n+1} \leq S_{n+1} \quad (i \in V)$$

since the start time will push back the minimum project duration by at least this much. These smaller domains result in $\sum_{i \in V}(d_{0i} + d_{i,n+1})$ less Boolean variables and clauses for the variable representation in the SAT solver. The maximal size of a nogood is not affected.

Preliminary experiments confirmed that starting the solution process with a smaller initial domain offers major improvements in the runtime for solving an instance and generating a first solution, especially on larger instances.

7.3.2 Activities in Disjunction

Two activities i and $j \in V$ are in *disjunction*, if they cannot be executed at the same time, i.e., the sum of their resource usages for at least one resource $k \in \mathcal{R}$ is greater than the available capacity: $r_{ik} + r_{jk} > R_k$. Activities in disjunction can be exploited in order to reduce the search space.

The simplest way to model two activities i and j in disjunction is by two half-reified constraints (Feydy et al. 2011) sharing the same Boolean variable B_{ij}.

$$B_{ij} \rightarrow S_i + p_i \leq S_j \qquad (i, j \in V : i \text{ and } j \text{ in disjunction and } i < j)$$

$$\neg B_{ij} \rightarrow S_j + p_j \leq S_i \qquad (i, j \in V : i \text{ and } j \text{ in disjunction and } i < j)$$

If B_{ij} is *true* then i must end before j starts (denoted by $i \ll j$), and if B_{ij} is *false* then $j \ll i$. The literals B_{ij} and $\neg B_{ij}$ can be directly represented in the SAT solver, consequently B_{ij} represents the relation (structure) between these activities. The propagator of such a half-reified constraint can only infer new bounds on left hand side of the implication if the right hand side is *false*, and on the start times variables if the left hand side is *true*. For example, the right hand side in the second constraint is *false* if and only if $\max_D S_i - \min_D S_j < p_j$. In this case the literal $\neg B_{ij}$ must be *false* and therefore $i \ll j$.

We add these redundant constraints to the model which allows the propagation solver to determine information about start time variables more quickly. For each constraint, the Boolean variable is directly modeled in the SAT solver and the maximal size of a nogood increases by one.

The detection of which activity runs before another can be further improved by considering the domains of the start times, and the minimal distances in the project network (see Dorndorf et al. 2000). This requires keeping track of the minimal distance between each pair of activities in the network. Since such a propagator is not offered by the LCG solver, we do not use their improvement for the benchmarks.

7.4 The Branch-and-Bound Algorithm

Our branch-and-bound algorithm is based on start-time and conflict-driven branching strategies. We use them alone or in combination. After each branch all constraints from the model, including constraints for activities in disjunctive, are propagated until a fixpoint is reached or the inconsistency of the partial schedule or the instance is proven. In the first case a new node is explored and in the second case an unexplored branch is chosen if one exists or backtracking is performed.

The propagation solver uses a priority queue in which the unit propagation in the SAT solver has the highest priority followed by propagators for precedence constraints, objective constraints, and the constraints for activities in disjunction. The propagators for cumulative constraints have the lowest priority. Thus, they are executed after no further domain reduction can be made by any other propagators.

7.4.1 Start-Time Branching

The start-time branching strategy selects an unfixed start time variable S_i with the smallest possible start time $\min_D S_i$. If there is a tie between several variables then the variable with the biggest size, i.e., $\max_D S_i - \min_D S_i$, is chosen. If there is still a tie then the variable with the lowest index i is selected. The binary branching is as follows: left branch $S_i \leq \min_D S_i$, and right branch $S_i > \min_D S_i$. In the remainder this branching is denoted by MSLF.

This branching creates a time-oriented branch-and-bound algorithm similar to Dorndorf et al. (2000), but it is simpler and does not involve any dominance rules. Hence, it is weaker than their algorithm.

7.4.2 Conflict-Driven Branching

The conflict-driven branching is a binary branching over literals in the SAT solver. In the left branch, the literal is set to *true* and, in the right branch, to *false*. As described in Sect. 7.2.3.1 on page 144 the Boolean variables in the SAT solver represent values in the integer domain of a variable x (e.g., $\neg[\![x \leq 3]\!]$ and $[\![x \leq 10]\!]$) or a disjunction between activities. Hence, it creates a branch-and-bound algorithm that can be considered as a mixture of time oriented and conflict-set oriented.

As a branching heuristic, an activity-based heuristic is used which is a variant of the Variable-State-Independent-Decaying-Sum (VSIDS) (Moskewicz et al. 2001). This heuristic is embedded in the SAT solver. In each branch, it selects the literal with the highest activity counter where an *activity counter* is assigned to each literal, and is increased during conflict analysis if the literal is related to the conflict. The analysis results in a nogood which is added to the clause database. Here, we use the 1UIP as a nogood. Once in a while all counters are decreased by the same factor not only to avoid a (possible) overflow, but also to give literals in recent conflicts more weight.

In order to accelerate the finding of solutions and increase the robustness of the search on hard instances, VSIDS can be combined with restarts, which has been shown beneficial in SAT solving. On restart the set of nogoods and the activity counters have changed, so that the search will explore a very different part of the search tree. In the remainder VSIDS with restart is denoted by RESTART. Different restart policies can be applied. Here a geometric restart on failed nodes with an initial limit of 250 and a restart factor of 2.0 is used (Huang 2007; Walsh 1999).

7.4.3 Hybrid Branching

At the beginning of a search, the activity counters of the variables have to be initialized somehow. By default they are all initialized to the same value. With no useful information in this initial setting, these activities can mislead VSIDS resulting in poor performance. To avoid this, we consider a hybrid search that uses MSLF to search initially, which has the effect of modifying the activity counts to reflect some structure of the problem, and then switch to VSIDS after the first restart. Here, we switch the searches after exploration of the first 500 nodes unless otherwise stated. The strategy is denoted by HOT START, and HOT RESTART when VSIDS is combined with restart.

7.5 Other Approaches

Beside the LCG approach, there exist only a few other exact solution approaches using Boolean satisfiability solving, mostly for $PS|prec|C_{max}$. However, these approaches to solving $PS|prec|C_{max}$ can easily be extended for $PS|temp|C_{max}$. Different Boolean satisfiability approaches proposed include Horbach (2010), Berthold et al. (2010), Ansótegui et al. (2011), Heinz et al. (2013). They are differentiated from the LCG approach by: the Boolean variable representation, the handling of the resource constraints, the optimization heuristic, and the branching strategy.

The approach in Horbach (2010) internally modifies a SAT solver, to create a hand-tailored solution for $PS|prec|C_{max}$, but could be extended for $PS|temp|C_{max}$, In contrast to LCG, it uses the time-indexed decomposition for resource constraints which are handled outside the SAT solver. This decomposition requires one auxiliary Boolean variable μ_{it} for each activity i and time t in the planning horizon. The variable μ_{it} expresses whether an activity i runs at a specific time t or not. Start time variables are only represented by Boolean variables of form $[\![S_i = d]\!]$ in the SAT solver. Precedence relations are encoded as clauses using the auxiliary Boolean variables μ_{it} and $[\![S_i = d]\!]$. In order to minimize the project duration, a sequence of feasibility problems are set up with a decreasing fixed project duration. Horbach uses the built-in branching strategy of the SAT solver which is a variant of VSIDS with a restart policy. This approach leads to comparable results to the LCG approach on $PS|prec|C_{max}$ problems.

Berthold et al. (2010) present a generic exact solution approach for $PS|prec|C_{max}$ which is then also applied to $PS|temp|C_{max}$ in Heinz et al. (2013). As in our model, they model the resource constraints by the global cumulative propagators and precedence relations by binary inequality propagators. Since their approach combines not only CP and SAT, but also integer programming, linear programming relaxations of the resource constraints can also be used for separation. They use the lazy Boolean variable representation for integer variables and they do not add explanations to the clause database. Their cumulative propagator (Heinz and Schulz 2011) generates weaker explanations than the corresponding cumulative propagator in an LCG solver. A branch and bound algorithm is used for minimizing the project duration and it is combined with a dedicated branching strategy for $PS|prec|C_{max}$. Their $PS|prec|C_{max}$ approach (Berthold et al. 2010) is competitive, but their $PS|temp|C_{max}$ approach (Heinz et al. 2013) is not competitive; using a 10 min time limit it can only solve about 40 % of the instances whereas our approach solves the majority of the same instances.

Ansótegui et al. (2011) tackles $PS|prec|C_{max}$ with a satisfiability modulo theories (SMT) solver. Such solvers work similarly to an LCG solver in which theory propagators act like constraint propagators in LCG, but consider all of one class of constraints simultaneously. Individual theory propagators only communicate through Boolean variables. They studied different decompositions of the resource constraints into linear inequality constraints, and then solved these using the theory of linear integer arithmetic propagation. Precedence relations are modelled with the same theory. In contrast to other approaches, they use a dichotomic optimization strategy. They use the default (VSIDS like) branching strategy of the SMT solver Yices (Dutertre and de Moura 2006). This approach is roughly equivalent in performance to the LCG approach in Schutt et al. (2011) for $PS|prec|C_{max}$. However, on some problems where many activities can be executed concurrently, it performs better than LCG approach.

7.6 Computational Results

We carried out experiments on $PS|temp|C_{max}$ instances from the benchmark suites CD, UBO, and SM. Here, we provide a summary of our exact solution method with different branching strategies presented in the previous section. Detailed results of our method are available at http://ww2.cs.mu.oz.au/~pjs/rcpsp. A comparison of our method with other exact and non-exact solution methods is reported in Schutt et al. (2013b).

The considered benchmark suites were systematically created by the instance generator ProGen/Max (Schwindt 1995) and have following characteristics:

CD: c, and d: each consisting of 540 instances with 100 activities and 5 resources.

UBO: ubo10, ubo20, ubo50, ubo100, ubo200, ubo500, and ubo1000: each containing 90 instances with 5 resources and 10, 20, 50, 100, 200, 500, and 1,000 activities respectively (Franck et al. 2001).

SM: j10, j20, and j30: each containing 270 instances with 5 resources and 10, 20, and 30 activities respectively (Kolisch et al. 1998).

Note that although the test set SM consists of small instances they are considerably harder than, e.g., ubo10 and ubo20.

The experiments were run on an Intel(R) Xeon(R) CPU E54052 processor with 2 GHz clock running GNU Linux. The code was written in Mercury using the G12 Constraint Programming Platform and compiled with the Mercury Compiler using grade hlc.gc.trseg. All clauses created during propagation and all nogoods inferred during conflict analysis were permanently added to the original problem, i.e., there was no garbage collection of clauses. Each run was given a 10 min runtime limit.

7.6.1 Setup and Table Notations

In order to solve each instance, a two-phase process was used. Both phases used the basic model with the two described extensions (cf. Sects. 7.3.1 and 7.3.2).

In the first phase, a Hot Start search was run to determine a first solution or to prove the infeasibility of the instance. The feasibility runs were set up with the trivial upper bound on the project duration $T := \sum_{i \in V} \max(p_i, \max\{\delta_{ij} \mid (i,j) \in E\})$. The feasibility test was run until a solution was found or infeasibility proved. If a solution was found, we use UB to denote the project duration of the resulting solution. In the first phase, the search strategy should be good at both finding a solution or proving infeasibility, but not necessarily at finding and proving the optimal solution. Hence, it could be exchanged with methods that might be more suitable than Hot Start. To improve Hot Start for finding a solution, when we used it in the first phase we explored $5 \times n$ nodes using the start-time branching strategy before switching to Vsids.

In the second optimization phase, each feasible instance was set up again this time with $T := UB$. The tighter bound is highly beneficial to lazy clause generation since it reduces the number of Boolean variables required to represent the problem. The search for optimality was performed using one of the various search strategies defined in the previous section.

The execution of the two-phased process leads to the following measurements.

$t^{\o}_{cpu}$: The average runtime in seconds (for both phases).

p_{fails}: The average number of infeasible nodes encountered in both phases of the search.

p_{feas}: The percentage of instances for which a solution was found.

p_{inf}: The percentage of instances for which the infeasibility was proven.

p_{opt}: The percentage of instances for which an optimal solution was found and proven.

Δ_{LB}: The average relative deviation (as a percentage) from the best known lower bounds of feasible instances given in Project Duration Problem RCPSP/max (2010). The relative deviation is $(best_k - LB_k)/LB_k$ for an instance k where $best_k$ and LB_k are the best found and best known lower bound of the instance k respectively.

$cmpr(i)$: Columns with this header give measurements only related to those instances that were solved by each procedure where i is the number of these instances.

$all(i)$: Columns with this header compare measurements for all instances examined in the experiment where i is the number of these instances.

7.6.2 Experiments on Instances up to 200 Activities

In the first experiment, we compare all of our search strategies against each other on all instances up to 200 activities. The strategies are compared in terms of average runtime in seconds ($t^{\o}_{cpu}$) and number of fails for each test set. For this experiment, we use the re-engineered version of LCG called LAZYFD (Feydy and Stuckey 2009).

The results are summarized in the Table 7.1. Similar to the results for the problem $PS|prec|C_{max}$ in Schutt et al. (2011) all strategies using VSIDS are superior to the start-time methods (MSLF), and similarly competitive. HOT RESTART is the most robust strategy, solving the most instances to optimality and having the lowest Δ_{LB}. Restart makes the search more robust for the conflict-driven strategies, whereas the impact of restart on MSLF is minimal.

In contrast to the results in Schutt et al. (2011) for $PS|prec|C_{max}$ the conflict-driven searches were not uniformly superior to MSLF. The three instances 67, 68, and 154 from j30 were solved to optimality by MSLF and MSLF with restart, but neither RESTART and HOT RESTART could prove the optimality in the given time limit, whereas VSIDS and HOT START were not even able to find an optimal solution

Table 7.1 Comparison on the test sets CD, UBO, ubo10-200 and SM. The best entries in each column is shown in bold

					cmpr(2230)		*all*(2340)	
Procedure	p_{feas}	p_{opt}	p_{inf}	Δ_{LB}	$t^{\varnothing}_{cpu}$	p_{fails}	$t^{\varnothing}_{cpu}$	p_{fails}
MSLF	85.0	80.60	15.0	3.96785	7.73	6,804	35.96	23,781
MSLF with restart	85.0	80.60	15.0	3.96352	7.80	6,793	36.04	23,787
VSIDS	85.0	82.26	15.0	3.76, 928	2.16	1,567	22.91	13,211
RESTART	85.0	82.26	15.0	3.73334	**2.02**	**1,363**	22.38	**12,212**
HOT START	85.0	82.31	15.0	3.84003	2.22	1,684	22.71	12,933
HOT RESTART	85.0	**82.35**	15.0	**3.73049**	2.04	1,475	**22.36**	12,341

within the time limit. Furthermore, none of our methods could find a first solution for the ubo200 instances 2, 4, and 70 nor prove the infeasibility for the ubo200 instance 40 within 10 min. Only for these instances we let our methods run until a first solution was found or infeasibility was proven. The corresponding numbers are included in Table 7.1.

7.6.3 *Experiments on Larger Instances*

The second experiment uses the largest instances of the test sets, namely the sub-test sets ubo500 and ubo1000. Due to the large initial domain size in the first phase of our solution approach the LCG solver LAZYFD was not suitable for those instances. Instead of LAZYFD, we use the LCG solver NICTA CPX which implements the lazy variable representation (Sect. 7.2.3.4).

Currently, NICTA CPX does not support restarts and the branching strategy MSLF. Furthermore, the conflict driven search of NICTA CPX works differently to VSIDS, because of the lazy variable representation and thus are incomparable. Therefore, we use following branching strategies:

FIRSTFAIL: Selects an unfixed start time variable S_i with the smallest domain size. If there is a tie then the variable with the lowest index i is chosen. Then FIRSTFAIL creates a branch for each value in the domain of S_i and explores them in increasing order.

FREE: Selects an unfixed variable and a split value in its domain based on a sparse integer conflict-driven heuristic inspired from VSIDS. Then FREE creates a binary branching that splits the domain along that value.

Moreover, we execute NICTA CPX in two different modes: lcg where explanations are added to the clause database (just as in LAZYFD) and fwd no explanations are not added to the clause database.

Table 7.2 shows on the one hand that almost for one third of the instances neither a solution was found nor the infeasibility proven, but on the other hand that the large majority of the other instances were either solved optimally or proven to be

Table 7.2 Comparison on the sub-test sets ubo500 and ubo1000. The best entries in each column is shown in bold

Procedure	p_{feas}	p_{opt}	p_{inf}	*cmpr*(108) $t^{\varnothing}_{cpu}$	*cmpr*(108) p_{fails}	*all*(180) $t^{\varnothing}_{cpu}$	*all*(180) p_{fails}
CPX with FIRSTFAIL+fwd	**66.7**	**64.4**	1.7	26.91	876	225.32	1,424
CPX with FREE+fwd	65.0	63.3	**2.2**	44.02	5,069	250.02	4,847
CPX with FIRSTFAIL+lcg	**66.7**	**64.4**	**1.7**	**26.01**	**840**	**223.91**	**1,316**
CPX with FREE+lcg	65.0	60.0	**2.2**	43.33	5,787	257.85	5,061

infeasible. If unit propagation is performed on explanations or not is insignificant for the instances considered, but the search strategy makes a difference. In this case, the search strategy FIRSTFAIL is preferable over FREE, except for infeasible instances.

We also ran NICTA CPX on the smaller instances, but the results were inferior to LAZYFD.

7.7 Conclusions

In this chapter, we introduce an exact solution method for $PS|temp|C_{max}$ based on lazy clause generation (LCG) which is a state-of-the-art method for this class of scheduling problem. LCG combines constraint programming with Boolean satisfiability solving in order to benefit from the advanced nogood learning technology and the generic conflict-driven search which branches over variables appearing often in recent conflicts.

Although the method presented provides the best results for instances up to 200 activities, bigger instances are currently beyond its capability. This is strongly related to the eager Boolean representation of integer variables which can consume a significant amount of memory when those variables have a large domain size. However, a significant part of the Boolean variables in the Boolean representation are not needed. Thus, this bottleneck can be solved by creating a Boolean variable only on demand.

Moreover, the method can further be improved by replacing the individual constraint propagators for each precedence relation by one constraint propagator that considers all precedence relations at once.

To sum up, the presented method is a state-of-the-art method for optimally solving $PS|temp|C_{max}$. The major strengths rely on two facts: the Boolean representation of integer variables by inequalities "$\leq$" and the nogood learning during conflict analysis which is nothing else than a resolution over constraints and variables related to the conflict and leads to a global valid constraint called nogood. If the nogood is added to the problem then it can exponentially prune the search space.

Acknowledgements NICTA is funded by the Australian Government as represented by the Department of Communications and the Australian Research Council through the ICT Centre of Excellence program. This work was partially supported by Asian Office of Aerospace Research and Development grant 10-4123.

References

Achterberg T (2009) SCIP: solving constraint integer programs. Math Program Comput 1(1):1–41

Aggoun A, Beldiceanu N (1993) Extending CHIP in order to solve complex scheduling and placement problems. Math Comput Model 17(7):57–73

Ansótegui C, Bofill M, Palahí M, Suy J, Villaret M (2011) Satisfiability modulo theories: an efficient approach for the resource-constrained project scheduling problem. In: Genesereth MR, Revesz PZ (eds) Proceedings of the ninth symposium on abstraction, reformulation, and approximation, SARA 2011. AAAI Press, Menlo Park

Baptiste P, Le Pape C, Nuijten W (2001) Constraint-based scheduling. Kluwer, Norwell

Bartusch M, Möhring RH, Radermacher FJ (1988) Scheduling project networks with resource constraints and time windows. Ann Oper Res 16(1):199–240

Bellman R (1958) On a routing problem. Q Appl Math 16(1):87–90

Berthold T, Heinz S, Lübbecke ME, Möhring RH, Schulz J (2010) A constraint integer programming approach for resource-constrained project scheduling. In: Lodi A, Milano M, Toth P (eds) Integration of AI and OR techniques in constraint programming for combinatorial optimization problems. Lecture notes in computer science, vol 6140. Springer, Heidelberg, pp 313–317

Brucker P, Drexl A, Möhring R, Neumann K, Pesch E (1999) Resource-constrained project scheduling: notation, classification, models, and methods. Eur J Oper Res 112(1):3–41

Davis M, Putnam H (1960) A computing procedure for quantification theory. J ACM 7:201–215

Davis M, Logemann G, Loveland D (1962) A machine program for theorem proving. Commun ACM 5(7):394–397

Dorndorf U, Pesch E, Phan-Huy T (2000) A time-oriented branch-and-bound algorithm for resource-constrained project scheduling with generalised precedence constraints. Manage Sci 46(10):1365–1384

Dutertre B, de Moura L (2006) The Yices SMT solver. http://yices.csl.sri.com/tool-paper.pdf

Feydy T (2010) Constraint programming: improving propagation. Ph.D. dissertation, University of Melbourne, Melbourne

Feydy T, Stuckey PJ (2009) Lazy clause generation reengineered. In: Gent I (ed) Proceedings of the 15th international conference on principles and practice of constraint programming. Lecture notes in computer science, vol 5732. Springer, Heidelberg, pp 352–366

Feydy T, Schutt A, Stuckey PJ (2008) Global difference constraint propagation for finite domain solvers. In: Antoy S, Albert E (eds) Proceedings of principles and practice of declarative programming – PPDP 2008. ACM, New York, pp 226–235

Feydy T, Somogyi Z, Stuckey PJ (2011) Half reification and flattening. In: Lee JHM (ed) Proceedings of principles and practice of constraint programming – CP 2011. Lecture notes in computer science, vol 6876. Springer, Heidelberg, pp 286–301

Ford LR Jr, Fulkerson DR (1962) Flows in networks. Princeton University Press, Princeton

Franck B, Neumann K, Schwindt C (2001) Truncated branch-and-bound, schedule-construction, and schedule-improvement procedures for resource-constrained project scheduling. OR Spectr 23(3):297–324

Heinz S, Schulz J (2011) Explanations for the cumulative constraint: an experimental study. In: Pardalos PM, Rebennack S (eds) Experimental algorithms. Lecture notes in computer science, vol 6630. Springer, Heidelberg, pp 400–409

Heinz S, Schulz J, Beck JC (2013) Using dual presolving reductions to reformulate cumulative constraints. Constraints 18(2):166–201

Horbach A (2010) A boolean satisfiability approach to the resource-constrained project scheduling problem. Ann Oper Res 181:89–107

Huang J (2007) The effect of restarts on the efficiency of clause learning. In: Veloso MM (ed) Proceedings of artificial intelligence – IJCAI 2007, pp 2318–2323

Kolisch R, Schwindt C, Sprecher A (1998) Benchmark instances for project scheduling problems. In: Węglarz J (ed) Project scheduling: recent models, algorithms and applications. Kluwer, Boston, pp 197–212

Le Pape C (1994) Implementation of resource constraints in ILOG SCHEDULE: a library for the development of constraint-based scheduling systems. Intell Syst Eng 3(2):55–66

Marriott K, Stuckey PJ (1998) Programming with constraints: an introduction. The MIT Press, Cambridge

Mercier L, Van Hentenryck P (2008) Edge finding for cumulative scheduling. INFORMS J Comput 20(1):143–153

Moskewicz MW, Madigan CF, Zhao Y, Zhang L, Malik S (2001) Chaff: engineering an efficient SAT solver. In: Design automation conference. ACM, New York, pp 530–535

Neumann K, Schwindt C (1997) Activity-on-node networks with minimal and maximal time lags and their application to make-to-order production. OR Spectr 19:205–217

Ohrimenko O, Stuckey PJ, Codish M (2009) Propagation via lazy clause generation. Constraints 14(3):357–391

Project Duration Problem RCPSP/max (2010). http://www.wior.uni-karlsruhe.de/LS_Neumann/Forschung/ProGenMax/rcpspmax.html

Schutt A, Wolf A (2010) A new $\mathscr{O}(n^2 \log n)$ not-first/not-last pruning algorithm for cumulative resource constraints. In: Cohen D (ed) Principles and practice of constraint programming – CP 2010. Lecture notes in computer science, vol 6308. Springer, Heidelberg, pp 445–459

Schutt A, Feydy T, Stuckey PJ, Wallace MG (2009) Why cumulative decomposition is not as bad as it sounds. In: Gent IP (ed) Proceedings of principles and practice of constraint programming – CP 2009. Lecture notes in computer science, vol 5732. Springer, Heidelberg, pp 746–761

Schutt A, Feydy T, Stuckey PJ, Wallace MG (2011) Explaining the cumulative propagator. Constraints 16(3):250–282

Schutt A, Chu G, Stuckey PJ, Wallace MG (2012) Maximising the net present value for resource-constrained project scheduling. In: Beldiceanu N, Jussien N, Pinson E (eds) Integration of AI and OR techniques in contraint programming for combinatorial optimzation problems. Lecture notes in computer science, vol 7298. Springer, Heidelberg, pp 362–378

Schutt A, Feydy T, Stuckey P (2013a) Explaining time-table-edge-finding propagation for the cumulative resource constraint. In: Gomes CP, Sellmann M (eds) Integration of AI and OR techniques in constraint programming for combinatorial optimization problems. Lecture notes in computer science, vol 7874. Springer, Heidelberg, pp 234–250

Schutt A, Feydy T, Stuckey PJ, Wallace MG (2013b) Solving RCPSP/max by lazy clause generation. J Sched 16(3):273–289

Schwindt C (1995) ProGen/max: a new problem generator for different resource-constrained project scheduling problems with minimal and maximal time lags. WIOR 449, Universität Karlsruhe, Karlsruhe

Tsang E (1993) Foundations of constraint satisfaction. Academic, London

Vilím P (2011) Timetable edge finding filtering algorithm for discrete cumulative resources. In: Achterberg T, Beck JC (eds) Integration of AI and OR techniques in constraint programming for combinatorial optimization problems. Lecture notes in computer science, vol 6697. Springer, Heidelberg, pp 230–245

Walsh T (1999) Search in a small world. In: Proceedings of artificial intelligence – IJCAI 1999. Morgan Kaufmann, San Francisco, pp 1172–1177

Part III
Alternative Resource Constraints in Project Scheduling

Chapter 8
Time-Varying Resource Requirements and Capacities

Sönke Hartmann

Abstract This contribution discusses an extension of the classical resource-constrained project scheduling problem (RCPSP) in which the resource requests of the activities and the resource capacities may change over time. We present relationships to other variants of the RCPSP as well as some applications of this problem setting. Subsequently, we analyze the applicability of heuristics which were originally developed for the standard RCPSP and adapt one of them, a genetic algorithm, to this extension. The chapter closes with a few computational results and some remarks on research perspectives.

Keywords Genetic algorithm • Makespan minimization • Project scheduling • Time-varying resource constraints

8.1 Introduction

In the classical resource-constrained project scheduling problem (RCPSP), a set of activities has to be scheduled such that precedence and resource constraints are met. That is, activities may not start before their predecessors have finished, and the resource requests may not exceed the capacities in any period. The goal is to determine a schedule with the shortest possible project duration.

The RCPSP is simple to describe but hard to solve, which has it made attractive for many researchers. One important direction of research has been the development of better solution methods (see Kolisch and Hartmann 2006 for an overview of heuristics). Another main research direction has been the definition of alternative and more general problem settings (see Hartmann and Briskorn 2010 for a survey).

Popular variants and extensions of the standard RCPSP include multiple execution modes for activities (Chaps. 21 and 22 of this handbook), generalized precedence relations (Chaps. 5, 6, and 7 of this handbook), and alternative objectives such as maximization of the net present value (Chap. 14 of this handbook). Also

S. Hartmann (✉)
HSBA Hamburg School of Business Administration, Hamburg, Germany
e-mail: soenke.hartmann@hsba.de

C. Schwindt, J. Zimmermann (eds.), *Handbook on Project Management and Scheduling Vol. 1*, International Handbooks on Information Systems,
DOI 10.1007/978-3-319-05443-8_8

several approaches to generalize the resource constraints have been proposed. This includes alternative resource types such as storage resources (Chap. 9 of this handbook), continuous resources (Chap. 10 of this handbook), and partially renewable resources (Chap. 11 of this handbook).

In this contribution, we take a look at another generalization of the resource constraints. Resource requests and capacities are assumed to be constant over time in the standard RCPSP. This might be too restrictive for practical applications. Vacation of staff or planned maintenance of machines lead to capacities which vary over time. Also the resource request of an activity might not be constant during its processing time, depending on the actual application. Consequently, we consider an extension of the RCPSP in which resource availabilities are given for each period of the planning horizon, and resource demands are given for each period of an activity's duration. The resulting problem setting is referred to as RCPSP/t to indicate the time-dependency.

Resource capacities and requests varying with time have not yet gained much attention in the scientific literature. While the concept has been mentioned in a few papers (e.g., Bartusch et al. 1988; de Reyck et al. 1999; Sprecher 1994), the only dedicated approach is that of Hartmann (2013). There, a priority rule is developed for time-dependent capacities and requests and embedded into a general randomized scheduling framework. Tests based on a large set of test instances revealed that the priority rule performs marginally better than standard RCPSP rules such as the well-known latest start time rule (LST).

The goal of this contribution is twofold. First, we underscore the relevance of the RCPSP/t by pointing out to applications in medical research and aggregated production planning. Second, we discuss heuristics and their performance. We summarize findings concerning the applicability of heuristics that were designed for the standard RCPSP, and we exploit these results to adapt a genetic algorithm to the RCPSP/t.

8.2 Problem Setting

In this section we will give a more formal outline of the problem setting. We will also have a brief look at other project scheduling problems which are special cases of the problem discussed here or which include the problem discussed here as a special case, respectively.

8.2.1 Formal Problem Description

The RCPSP with time-varying resource requirements and capacities, the RCPSP/t, can be summarized as follows. As in the standard RCPSP, we consider n activities

$1, \ldots, n$ with a processing time or duration p_j for each activity $j \in \{1, \ldots, n\}$. Once started, an activity may not be interrupted. An activity j may not be started before all its predecessor activities $i \in Pred(j)$ have finished, where $Pred(j)$ denotes the set of the immediate predecessors of activity j. Two additional activities 0 and $n + 1$ are added. They are dummy activities with $p_0 = p_{n+1} = 0$ and reflect the start and the end of the project, respectively.

K resources are given. Resource k, $k = 1, \ldots, K$ has a capacity of $R_k(t)$ in period $t = 1, \ldots, T$, where T is the planning horizon. An activity j requires $r_{jk}(t)$ units of resource k in the t-th period of its duration, i.e., $t = 1, \ldots, p_j$. We usually assume the parameters such as processing times, capacities, and resource requests to be nonnegative and integer valued. The objective is to determine a schedule (i.e., a start time for each activity) with minimal total project duration (i.e., minimal makespan) such that both the temporal and the resource constraints are fulfilled.

A mathematical model can easily be obtained from incorporating the time-dependency of the resource parameters into the classical formulation of Pritsker et al. (1969). The resulting model can be found in Hartmann (2013). In the three-field classification scheme $\alpha|\beta|\gamma$ of Brucker et al. (1999), the RCPSP/t would be denoted by $PSt|prec|C_{max}$, where $\alpha = PSt$ stands for project scheduling with limited time-varying renewable resources.

8.2.2 *Relationships to Other Project Scheduling Problems*

Obviously, the RCPSP/t generalizes the standard RCPSP. If all resources have constant capacities and if all activities have constant resource requests, we obtain the standard RCPSP.

Furthermore, the RCPSP/t is a special case of the RCPSP/max, i.e., the RCPSP with generalized precedence constraints. As outlined by Bartusch et al. (1988), time-varying resource capacities can be transformed into constant ones by selecting the highest capacity and by defining a dummy activity for each drop in the capacity. Such a dummy activity is then fixed in time by adding a precedence relation with the source activity along with appropriate minimal and maximal time lags. Time-varying resource requests can be obtained in the RCPSP/max by splitting an activity whenever the request for a resource changes. The resulting sub-activities related to the original activity can now be stitched together using minimal and maximal time lags.

Finally, it should be mentioned that time-varying resource capacities are a special case of the so-called partially renewable resources (Böttcher et al. 1999 and Chap. 11 of this handbook). A partially renewable resource k is associated with a set of period subsets $\Pi_k = \{P_{k1}, \ldots, P_{k\nu}\}$ where $P_{k\mu} \subseteq \{1, \ldots, T\}$ for each $\mu = \{1, \ldots, \nu\}$. For each period subset $P_{k\mu} \in \Pi_k$ there is a total resource capacity $R_k^p(P_{k\mu})$. That is, over the periods $t \in P_{k\mu}$, the total capacity of resource k is $R_k^p(P_{k\mu})$. Time-varying resource capacities can obviously be captured by defining one period subset $\{t\}$ for each period t of the planning horizon and by setting $R_k^p(\{t\}) := R_k(t)$.

8.3 Applications

The applicability of time-varying resource capacities and requests is straightforward. The capacity of a resource may vary due to vacations of staff or maintenance of machines. Resource requests of an activity are not necessarily constant over time in case of longer or more complex activities. Case studies on project scheduling problems including such a time-dependency can occasionally be found in the literature, see, e.g., Kolisch and Meyer (2006). This section reviews two specific applications of time-varying resource capacities and requests.

8.3.1 Medical Research Projects

As reported by Hartmann (2013), the RCPSP/t can be applied to capture projects in medical research. Many projects in this field consist of experiments which require certain resources such as laboratory staff and equipment. The experiments can be viewed as activities which have to be scheduled such that an early end of the project is achieved. These characteristics suggest to apply project scheduling models like the RCPSP. A few properties of such medical research projects, however, go beyond the standard RCPSP, but they can easily be captured by the RCPSP/t.

The laboratory staff and the equipment often have capacities that vary over time. The staff might be absent due to vacation, or it might be present on certain weekends. Since often several projects are carried out in the same laboratory at the same time, the equipment might be available for a particular project only during certain weeks or on certain weekdays.

Moreover, the resource requests of the activities representing experiments need not be constant over time. Certain laboratory equipment might be required only on specific days, e.g., on the last day of an experiment. Also, laboratory staff might be required only on certain days of an experiment.

Finally, there are often specific requirements concerning the temporal arrangement of experiments. A typical constraint is that many repetitions of the same experiment are carried out in parallel because this makes it easier and more efficient to handle them. It also leads to a clearer schedule which helps to avoid mistakes when processing the experiments. On the other hand, it is important that not all repetitions are executed in parallel because otherwise possible mistakes in the handling of the experiments might not be detected and the results are distorted.

This can be captured by the RCPSP/t as follows: The repetitions of one experiment are grouped such that each group contains a number of repetitions that should be carried out in parallel. For each such group one activity is defined.

For example, if an experiment must be repeated 30 times, one would define three activities corresponding to 10 repetitions each, given that always 10 repetitions should be carried out in parallel. To make sure that these three activities are not carried out in parallel in the sense that they do not start on the same day, we define a fictitious resource with a constant capacity of one unit. Each of the three activities requests one unit this additional resource in the first period of its duration (and no units in the remaining periods).

More details on modeling medical research projects using the RCPSP/t as well as a case study based on real data can be found in Hartmann (2013).

8.3.2 Aggregated Production Scheduling

In what follows, we consider the case of a manufacturer of special machines located in Northern Germany. The company produces highly specialized machines in a make-to-order process. In addition to short-term planning at a detailed level, also long-term planning at an aggregated level is carried out.

The long-term planning approach takes all currently known orders into account. Each order corresponds to a machine of a particular type. While the production process of a machine is related to a project network with several activities, it is not necessary to consider all these individual activities in long-term planning.

Moreover, the only resources that need to be considered in long-term planning are the assembly areas. There are several different assembly areas, and each of these contains a number of so-called cells. The production of a machine type requires one or more cells from several assembly areas.

Each order is related to a deadline which must not be exceeded. The goal of aggregated planning is to determine a schedule in which the production of all ordered machines finishes as early as possible. The latter is of particular importance since it leads to free resource capacities for future orders.

This planning task can be captured as follows. We define one activity for each order. In other words, each project that is related to the production of one ordered machine is represented by a single activity. This activity is derived from the standard schedule that is associated with the production process of the related machine. Within this standard schedule, the start times are assumed to be fixed. The duration of an activity is defined as the total manufacturing time of the machine according to the standard schedule.

Obviously, each assembly area corresponds to one resource with a capacity given by the number of cells that are available. An activity requires some of the assembly areas, the request corresponds to the number of required cells. The request for an assembly area usually varies over time. For example, the first part of an activity might correspond to the manufacturing of certain parts in related assembly areas. The second part might correspond to the assembly of the machine using these parts. For this, another assembly area is needed whereas the assembly areas of the parts are no longer needed. This aggregation of activities to super-activities with time-varying resource requests has also been applied by Heimerl and Kolisch (2010) in a similar way.

We are looking for a start time for each activity (which must observe the given deadline of each activity). That is, we are looking for the time at which the production related to an order should start such that the makespan is minimized. Summing up, we obtain the RCPSP/t with an additional deadline constraint.

8.4 Heuristics for the RCPSP/t

This section deals with heuristics for the RCPSP/t. We discuss in general how heuristics developed for the standard RCPSP can be applied. Subsequently, we consider a genetic algorithm which is extended by so-called delays in order to match the search space of the RCPSP/t.

8.4.1 Applicability of Heuristics Designed for the Standard RCPSP

Over the last decades, a large number of heuristics have been developed for the classical RCPSP, for overviews refer to Kolisch and Hartmann (1999, 2006). Since the structure is very similar to the RCPSP/t, many of them can also be applied to the extended problem. One particular reason is that the so-called schedule-generation schemes (SGS) which are the backbone of most RCPSP heuristics also work for the RCPSP/t.

Two main SGS are available for the standard RCPSP, the serial and the parallel one (Kolisch 1996). The serial SGS picks an activity in each step and schedules it at the earliest precedence and resource feasible start time. The parallel SGS on the other hand considers a point in time and successively picks an activity that can start at this time without violating the precedence and resource constraints. Whenever there is no activity left that can be feasibly started, the next point in time is selected. Note that the SGS only guides the scheduling process. In both SGS, the decision which activity to pick next is made by a priority rule or by a metaheuristic representation.

As shown by Sprecher et al. (1995), the search space of the serial SGS always contains an optimal schedule for the RCPSP whereas the parallel one sometimes does not. Thus the parallel SGS might not be able to find an optimal solution for a given instance. This a drawback especially in small search spaces where the serial SGS is superior because it might be able to find an optimal solution. On the other hand, the parallel SGS is superior when applied to large instances because it produces schedules of better average quality, which is an advantage in large search spaces.

Generally, both SGS can be applied to the RCPSP/t. However, their properties change for this problem class. There are some instances of the RCPSP/t for which the search space of the serial SGS does not contain an (existing) optimal solution. This has been shown by counterexample in Hartmann (2013). This counterexample indicates that one may have to start an activity later than at the earliest possible start time to find an optimal solution.

Another important component of many RCPSP heuristics is the so-called justification or forward-backward improvement approach of Tormos and Lova (2001). The idea is to improve a schedule as follows. In a first step, the schedule is scanned from right to left. Thereby, each activity is shifted to the right as far as possible, but not beyond the dummy sink activity. Next, the activities are scanned from left to right and shifted as far to the left as possible, but not before the new start time of the dummy source activity. Valls et al. (2005) have demonstrated that this concept can be added to almost every heuristic and that it improves the results drastically.

This concept is not applicable when resource capacities are varying with time. Activities are shifted to the right to condense the project, but thereby also the dummy start activity is started later. In case of constant capacities, one can simply shift the entire project to the left and hence bring it back to the old start time while keeping the condensed schedule. The latter, however, is not possible if resource capacities vary over time.

Summing up, the SGS for the RCPSP are applicable to the RCPSP/t as well. This also holds for heuristics based on these SGS. Unfortunately, however, both SGS are unable to find an existing optimal solution for some instances of the RCPSP/t. Moreover, justification or forward-backward improvement is not applicable to the RCPSP/t. These issues must be considered when adapting heuristics from the RCPSP to the RCPSP/t.

8.4.2 An Adapted Genetic Algorithm

In this section, we adapt the genetic algorithm (GA) of Hartmann (1998) to the RCPSP/t. This GA was originally proposed for the standard RCPSP. It is based on the so-called activity list representation in which the non-dummy activities are given in some order. This order must be precedence feasible, that is, an activity may not appear before any of its predecessors. The serial SGS is applied to determine a schedule for an activity list. It simply takes the activities in the order given by the activity list and schedules each one at the earliest precedence and resource feasible time. The activity lists for the first generation are constructed using a randomized priority rule, namely the latest finish time rule (LFT).

The crossover operator takes parts of the activity lists from the mother and the father and combines them to form a child. We apply a two-point crossover for which two positions q_1 and q_2 with $1 \leq q_1 < q_2 \leq n$ are drawn randomly. The activities for the child in positions $1, \ldots, q_1$ are copied from the father. Child positions $q_1 + 1, \ldots, q_2$ are filled successively with activities from the mother. We always take the leftmost activity in the mother's list that does not yet appear in the child's current partial list. The remaining child positions $q_2 + 1, \ldots, n$ are taken from the father accordingly. Note that this crossover preserves the activities' relative positions in the parents and always produces precedence feasible offspring. The mutation operator swaps two activities with probability π_{mutation}, given that the result is still precedence feasible. For more details, refer to Hartmann (1998).

This GA has yielded better results than other genetic prepresentations for the standard RCPSP (Hartmann 1998). In the subsequent years, it has been extended by various researchers. The most noteworthy extension was the concept of justification or forward-backward improvement (see Sect. 8.4.1). As discussed above, however, it is not applicable to instances of the RCPSP/t. Therefore, we stick with the GA as described above.

In what follows, we discuss two ways of applying this GA to the RCPSP/t. The first approach is fairly straightforward. Considering that the serial SGS yields feasible solutions for the RCPSP/t, we can apply the GA described above. The only minor change is that we replace the method to calculate the first generation with an RCPSP/t-specific procedure, namely the tournament method and the critical path and resource utilization (CPRU) rule of Hartmann (2013). The main drawback of this GA approach is that it might not find an optimal solution because of the serial SGS (recall the discussion in Sect. 8.4.1).

This issue leads us to the second approach. Here we allow to delay activities, that is, an activity may be started at a time later than the earliest feasible start time. This extends the search space such that an optimal solution can be found.

The idea behind this is somewhat similar to that of Cho and Kim (1997) for the standard RCPSP. They proposed a simulated annealing (SA) heuristic in which solutions are represented by priority values and decoded by the parallel SGS. Recall that, like the serial SGS for the RCPSP/t, also the parallel SGS for the RCPSP might exclude all optimal solutions from the search space. To overcome this restriction, Cho and Kim (1997) suggest to allow to delay activities. They indicate a delay by a negative priority value (whereas an activity with a positive priority value is started as early as possible). Moreover, they test different rules to control the number of periods an activity may be delayed.

Our approach is similar as it allows to delay activities as well, albeit using the activity list representation and the serial SGS. Handling delays, however, follows a different concept. For each activity j we define a delay value $\delta(j)$ and include it in the representation. Now the activity list of an individual I includes a delay value for each activity:

$$I = \begin{pmatrix} j_1 & j_2 & \cdots & j_n \\ \delta(j_1) & \delta(j_2) & \ldots & \delta(j_n) \end{pmatrix}$$

The serial SGS is applied to determine the schedule for an individual. It schedules the activities in the order prescribed by the activity list. If an activity j_i has a delay value of $\delta(j_i) = 0$, then this activity is scheduled as early as possible. If it has a delay value of $\delta(j_i) = 1$, the earliest possible start time is ignored and the next feasible start time is selected. Higher delay values imply that more feasible start times are skipped. Generally, the earliest $\delta(j_i)$ feasible start times are skipped and the next feasible start time is selected. Note that $\delta(j_i)$ does not indicate the number of periods an activity is delayed. This is necessary because a feasible start time t for some activity does not mean that start time $t + 1$ is always feasible if the serial SGS is applied to the RCPSP/t. Also observe that, in contrast to Cho and Kim (1997), the magnitude of the delay is part of the representation and needs not be controlled by additional rules.

The GA based on this extended representation proceeds as follows. In the first generation, each individual is assigned an activity list using the tournament procedure and the CPRU rule. With a probability of π_{delay}, an activity is assigned a delay value of 1 and 0 otherwise (higher delay values can be produced by mutation). The crossover operator is extended in a straightforward way: Each activity j_i simply keeps its associated delay value $\delta(j_i)$. The mutation operator is expanded by changing a delay value with probability π_{mutation}. Then it is either increased by 1 or decreased by 1, each with a probability of 0.5 (a negative delay value is not accepted, though). This concept implies that the delay is subject to inheritance—if delaying an activity is beneficial, this will be passed on to the offspring.

Of course, it is not a priori clear whether or not delaying activities is a promising approach. On one hand, it helps to avoid the drawback of the serial SGS which might exclude all optimal solutions from the search space. On the other hand, scheduling activities later than necessary can be counterproductive if the objective is an early end of the project. Thus computational experiments can provide further insight.

8.5 Computational Results

In order to analyze the behavior of the heuristics and of the extended genetic algorithm, a computational study has been carried out. In what follows, we briefly describe the sets of test instances and present the computational results.

8.5.1 Test Sets

For the computational analysis we make use of the sets of test instances generated by Hartmann (2013). These sets are based on sets for the standard RCPSP which can be found in the internet-based project scheduling problem library PSPLIB, cf. Kolisch and Sprecher (1996). The original RCPSP sets were generated by ProGen (see Kolisch et al. 1995) and have been widely accepted as a standard test bed by the RCPSP community.

Hartmann (2013) added changes to the resource capacities and requests of these instances to adapt them to the RCPSP/t. These changes are based on parameters.

Probabilities P^R and P^r control whether or not a reduction is applied to the capacity and the request in a period, respectively. Higher probabilities imply more frequent changes in the resource availabilities and requests. Factors F^R and F^r determine the strength of the reduction for the availability and the request, respectively. The smaller the factor, the stronger the reduction.

The probabilities were set to 0.05, 0.1, and 0.2. The probabilities for changing the capacities and the requests were always kept the same, that is, $P^R = P^r$. The factors were set to 0 and 0.5. Also the factors for capacities and the requests are the same, that is, $F^R = F^r$. This led to six different instances derived from the set with $n = 30$ activities (and hence $6 \cdot 480 = 2{,}880$ instances in total) and six different instances derived from the set with $n = 120$ activities (thus $6 \cdot 600 = 3{,}600$ instances in total). More details can be found in Hartmann (2013).

8.5.2 Results

We tested the tournament heuristic of Hartmann (2013) with three different priority rules. We included the random rule (RND) as a benchmark, the latest start time rule (LST) which is one of the best performing rules for the standard RCPSP (Kolisch 1996), and the critical path and resource utilization rule (CPRU) which was developed for the RCPSP/t (Hartmann 2013). We also tested the GA of Hartmann (1998) in its standard form without delay as well as in the extended version designed for the RCPSP/t. In the latter version, different probabilities π_{delay} for controlling the distribution of delay values in the first generation were examined.

To obtain a basis for the comparison, all tested heuristics were stopped after 5,000 schedules were computed for an instance (in the GA, this corresponds to a population size of 100 over 50 generations). Table 8.1 provides a summary of the results obtained for the two test sets with $n = 30$ and $n = 120$ activities, respectively. It reports the average deviation from the lower bound LB/t (which makes use of the critical path and the time-varying resource parameters, see Hartmann 2013 for a definition), the percentage of instances for which a feasible solution is found, and the average computation time per instance in seconds.

The tested methods yield rather similar results for the set with $n = 30$, whereas the set with $n = 120$ shows differences between the heuristics. On the latter set, all methods clearly outperform the random approach. The CPRU rule is slightly better than the LST rule for the standard RCPSP, which confirms the findings of Hartmann (2013). The GAs lead to better results than the priority rule methods because they exploit learning effects during the search. This is in line with the results of Kolisch and Hartmann (2006).

For the set with $n = 30$, adding the possibility to delay activities in the GA can have a slight positive effect (the impact is fairly marginal, but it was confirmed in several repetitions of the experiment). For the set with $n = 120$ the GA without delays works best. In case of the much smaller solution space related to the $n = 30$

Table 8.1 Heuristic results, limit = 5,000 schedules

Heuristic	Configuration	$n = 30$			$n = 120$		
		Deviation	Feasible	CPU-sec	Deviation	Feasible	CPU-sec
Tournament	RND	11.9 %	98.3 %	0.101	39.1 %	100 %	0.471
Tournament	LST	11.3 %	98.3 %	0.103	31.7 %	100 %	0.627
Tournament	CPRU	11.3 %	98.3 %	0.104	31.1 %	100 %	0.643
GA	No delay	11.3 %	98.3 %	0.039	29.7 %	100 %	0.191
GA	$\pi_{\text{delay}} = 0.01$	11.2 %	98.5 %	0.041	30.3 %	100 %	0.209
GA	$\pi_{\text{delay}} = 0.05$	11.3 %	98.5 %	0.041	30.5 %	100 %	0.209
GA	$\pi_{\text{delay}} = 0.10$	11.3 %	98.5 %	0.042	30.8 %	100 %	0.216
GA	$\pi_{\text{delay}} = 0.20$	11.4 %	98.5 %	0.043	31.5 %	100 %	0.224

set, it makes sense to explore a search space that always contains an optimal solution. Considering the huge solution space for the $n = 120$ set, however, it seems to be more promising to reduce the search space to schedules of good average quality. The latter is achieved when activities are not delayed. This is very similar to the findings of Hartmann and Kolisch (2000) for the standard RCPSP, where a larger search space (due to the serial SGS) is better for smaller projects whereas a smaller search space (due to the parallel SGS) is more promising for larger projects.

The results also show that higher delay probabilities for setting up the first generation slightly worsens the results. Increasing the delay probability means that too many activities are delayed which leads to inferior solutions. But the results do not deteriorate too much because mutation and selection can eliminate unfavorable delay values from the gene pool. That indicates that the GA is robust because the evolution will discard delays sooner or later if they are not favorable.

The computation times of the priority rule methods are higher than those of the random method because of the time-consuming activity selection method. The lowest computation times are obtained for the GA because it simply picks the next activity from the activity list. Thus, there is no need for a more time-consuming activity selection. Increasing the delay probability leads to slightly higher computation times because in case of a delay the start time calculation of an activity has to be executed more than once. It should also be noted that the methods stop whenever the lower bound LB/t has been reached and thus an optimal solution has been found.

Tables 8.2 and 8.3 show the average deviations from the lower bound LB/t for the different instances subsets, that is, for the different probabilities and strengths of the resource capacity and request reduction. Likewise, Tables 8.4 and 8.5 display the percentages of those instances for which the lower bound LB/t was met (and hence the solution is proven to be optimal). Here, we restrict ourselves to the most important heuristics.

We observe that the solution gap between upper and lower bound is generally small for the sets with $n = 30$, and for many instances, the solution gap is 0 which means that the lower bound is met. Taking also the sets with $n = 120$ into account,

Table 8.2 Average deviation from lower bound LB/t in % ($n = 30$)

	$F^R = F^r$	0			0.5		
	$P^R = P^r$	0.05	0.1	0.2	0.05	0.1	0.2
Tournament	RND	11.7	8.7	5.9	14.7	15.2	14.4
Tournament	CPRU	11.2	8.4	5.8	13.9	14.5	13.6
GA	No delay	11.3	8.5	5.9	13.8	14.4	13.6
GA	$\pi_{\text{delay}} = 0.01$	11.2	8.4	5.8	13.9	14.3	13.3

Table 8.3 Average deviation from lower bound LB/t in % ($n = 120$)

	$F^R = F^r$	0			0.5		
	$P^R = P^r$	0.05	0.1	0.2	0.05	0.1	0.2
Tournament	RND	35.6	27.7	15.9	49.8	51.4	54.0
Tournament	CPRU	28.3	21.3	12.5	40.5	41.3	43.0
GA	No delay	27.1	20.6	12.3	38.3	39.3	40.8
GA	$\pi_{\text{delay}} = 0.01$	27.5	20.9	12.2	39.2	40.1	41.7

Table 8.4 Percentage of instances for which lower bound LB/t is met ($n = 30$)

	$F^R = F^r$	0			0.5		
	$P^R = P^r$	0.05	0.1	0.2	0.05	0.1	0.2
Tournament	RND	57.9	66.5	69.4	38.5	34.0	34.2
Tournament	CPRU	58.3	67.3	69.6	39.0	34.8	35.2
GA	No delay	57.7	67.7	69.0	38.8	34.8	34.8
GA	$\pi_{\text{delay}} = 0.01$	58.1	66.9	69.6	39.0	34.6	34.6

Table 8.5 Percentage of instances for which lower bound LB/t is met ($n = 120$)

	$F^R = F^r$	0			0.5		
	$P^R = P^r$	0.05	0.1	0.2	0.05	0.1	0.2
Tournament	RND	25.3	36.7	59.5	6.7	4.5	5.5
Tournament	CPRU	34.3	43.8	63.8	16.3	12.3	9.3
GA	No delay	35.7	44.3	63.2	16.0	12.5	9.8
GA	$\pi_{\text{delay}} = 0.01$	36.0	45.7	65.3	16.3	12.8	10.3

we see that generally the reduction factors of $F^R = F^r = 0$ lead to lower solution gaps and more optimal solutions than factors of $F^R = F^r = 0.5$. Among the sets for $F^R = F^r = 0$, the solution gap is lowest for a high reduction probability ($P^R = P^r = 0.2$). Also note that for the sets with $F^R = F^r = 0$ and $P^R = P^r = 0.2$ there is only a small difference between the random method and the best GA ($n = 120$) or hardly any difference at all ($n = 30$).

A capacity reduction down to 0 means that less degrees of freedom exist for scheduling activities. In such a case, there may be only a few resource feasible start times, especially for activities with a long duration. If such an activity can only start rather late due to the resource capacities, this may determine the makespan to a large degree, and better heuristics cannot find better schedules than simple random methods. Taking these observations into account, future research might focus more on the sets with $F^R = F^r = 0.5$ because this setting leaves more degrees of freedom for scheduling, which helps to identify performance differences between different heuristics.

8.6 Conclusions

In this contribution, we have summarized the current state of research concerning the RCPSP with time-dependent resource availabilities and requests. We have also adapted a well-known genetic algorithm which was originally proposed for the standard RCPSP. By allowing activities to start later than necessary, we have extended the search space of the genetic algorithm in a way than an optimal solution can be found for the RCPSP/t. The computational experiments showed, however, that delaying activities only leads to better results in case of small project instances.

Future research directions for the RCPSP/t can be promising in two directions. First, further methods tailored for this problem class might lead to improved results. Second, research on real-world applications and case studies can be useful to assess the relevance of the RCPSP/t. It can also be a good idea to use case studies to identify promising combinations of the RCPSP/t with other extensions of the RCPSP. The case study concerning aggregated production planning has shown that additional deadlines can be relevant in practice. Further extensions might also be possible, e.g., multiple modes to reflect alternative speeds of the production processes. Multiple modes can also be useful when the selection of orders shall be included, since one mode can reflect the decision not to carry out the production. When order selection is included, also a different objective function such as the maximization of the net present value would be needed.

Acknowledgements The author was supported by the HSBA foundation.

References

Bartusch M, Möhring RH, Radermacher FJ (1988) Scheduling project networks with resource constraints and time windows. Ann Oper Res 16:201–240

Böttcher J, Drexl A, Kolisch R, Salewski F (1999) Project scheduling under partially renewable resource constraints. Manage Sci 45:543–559

Brucker P, Drexl A, Möhring R, Neumann K, Pesch E (1999) Resource-constrained project scheduling: notation, classification, models, and methods. Eur J Oper Res 112:3–41

Cho JH, Kim YD (1997) A simulated annealing algorithm for resource-constrained project scheduling problems. J Oper Res Soc 48:736–744

de Reyck B, Demeulemeester EL, Herroelen WS (1999) Algorithms for scheduling projects with generalized precedence relations. In: Węglarz J (ed) Project scheduling: recent models, algorithms and applications. Kluwer, Boston, pp 77–106

Hartmann S (1998) A competitive genetic algorithm for resource-constrained project scheduling. Nav Res Logist 45:733–750

Hartmann S (2013) Project scheduling with resource capacities and requests varying with time: a case study. Flex Serv Manuf J 25:74–93

Hartmann S, Briskorn D (2010) A survey of variants and extensions of the resource-constrained project scheduling problem. Eur J Oper Res 207:1–14

Hartmann S, Kolisch R (2000) Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. Eur J Oper Res 127:394–407

Heimerl C, Kolisch R (2010) Scheduling and staffing multiple projects with a multi-skilled workforce. OR Spectr 32:343–368

Kolisch R (1996) Serial and parallel resource-constrained project scheduling methods revisited: theory and computation. Eur J Oper Res 90:320–333

Kolisch R, Hartmann S (1999) Heuristic algorithms for solving the resource-constrained project scheduling problem: classification and computational analysis. In: Węglarz J (ed) Project scheduling: recent models, algorithms and applications. Kluwer, Boston, pp 147–178

Kolisch R, Hartmann S (2006) Experimental investigation of heuristics for resource-constrained project scheduling: an update. Eur J Oper Res 174:23–37

Kolisch R, Meyer K (2006) Selection and scheduling of pharmaceutical research projects. In: Jozefowska J, Węglarz J (eds) Perspectives in modern project scheduling. Springer, New York, pp 321–344

Kolisch R, Sprecher A (1996) PSPLIB – a project scheduling problem library. Eur J Oper Res 96:205–216

Kolisch R, Sprecher A, Drexl A (1995) Characterization and generation of a general class of resource-constrained project scheduling problems. Manage Sci 41:1693–1703

Pritsker AAB, Watters LJ, Wolfe PM (1969) Multiproject scheduling with limited resources: a zero-one programming approach. Manage Sci 16:93–107

Sprecher A (1994) Resource-constrained project scheduling: exact methods for the multi-mode case. Lecture notes in economics and mathematical systems, No. 409. Springer, Berlin

Sprecher A, Kolisch R, Drexl A (1995) Semi-active, active and non-delay schedules for the resource-constrained project scheduling problem. Eur J Oper Res 80:94–102

Tormos P, Lova A (2001) A competitive heuristic solution technique for resource-constrained project scheduling. Ann Oper Res 102:65–81

Valls V, Ballestín F, Quintanilla MS (2005) Justification and RCPSP: a technique that pays. Eur J Oper Res 165:375–386

Chapter 9
Storage Resources

Jacques Carlier and Aziz Moukrim

Abstract This chapter looks at project scheduling problems with storage resources. Activities can produce or consume resources at their start or at their completion. We consider projects composed of events subject to precedence constraints and resource constraints. We describe briefly the exact methods of Neumann and Schwindt and of Laborie, which solve the problem when stocks of resources have to be between minimum and maximum levels. We then suppose that there are no maximum stocks. We report the shifting algorithm which solves in polynomial time the financing problem where resources are produced at given dates. We also explain how an earliest schedule corresponding to a linear order of consumption events can be built. The enumeration of linear orders then becomes sufficient for building an exact method.

Keywords Linear order • Makespan minimization • Project scheduling • Storage resources

9.1 Introduction

Most papers on scheduling problems consider activities that use *renewable* resources, meaning that the activities require certain quantities of resources when they start, and then give them back at their completion (see Chaps. 1, 2, and 3 of this handbook, Graham et al. 1979, and Neumann et al. 2003). This is the case, for instance, for the Resource Constrained Project Scheduling Problem, where activities are also subject to precedence constraints. We assume that all data take the form of integer values. Renewable resources can model manpower or machines. In this chapter we consider non-renewable resources that can be consumed or produced by activities when they start or when they complete. We use the term *storage resources*, because when a quantity of resources is produced it is stocked and when it is consumed it is de-stocked. Such resources can represent money, raw materials,

J. Carlier (✉) • A. Moukrim
Heudiasyc, CNRS UMR 7253, Université de Technologie de Compiègne, Compiègne, France
e-mail: jacques.carlier@utc.fr; aziz.moukrim@utc.fr

C. Schwindt, J. Zimmermann (eds.), *Handbook on Project Management and Scheduling Vol. 1*, International Handbooks on Information Systems,
DOI 10.1007/978-3-319-05443-8_9

mechanical parts etc. Non-renewable resources were introduced in Błażewicz, cf. Chap. 1: Błażewicz et al. (1986) and in Carlier and Rinnooy Kan (1982). Carlier and Rinnooy Kan consider the particular case of money by looking at the financing problem where different amounts of money arrive at given dates and can be used to finance activities. More recently Neumann and Schwindt (2003) consider the project scheduling problem with inventory constraints, where there are several resources, the level of whose stocks must always respect minimum and maximum values. The problem consists in computing a schedule that is resource-feasible, time-feasible, and that minimizes makespan. A schedule is time-feasible if precedence constraints are satisfied and resource-feasible if at any given time the stock of each resource is between the minimum and maximum permitted levels. Neumann and Schwindt propose a branch and bound method for solving this scheduling problem. It is an exact method that can solve most of the instances of a benchmark with up to 100 events. We will be describing the Neumann and Schwindt (2003) method briefly below. Laborie (2003) also proposes a particularly innovative exact method that can solve all the instances of Neumann and Schwindt (2003). Other authors have studied the particular case when there is no maximum stock. This is known as the problem with consumption and production of resources. Carlier et al. (2009) explain how to build an earliest schedule associated with a linear order on events that enables schedules to be enumerated by enumerating linear orders. They also prove that linear orders can be limited to consumption events or production events. Koné et al. (2013) propose solving the problem using a linear programming model. Given that the project scheduling problem with storage resources is interesting from both the practical and theoretical standpoints, more research into this subject is clearly desirable. The chapter is organized as follows. After this introduction, we describe, in Sect. 9.2, the project scheduling problem with inventory constraints and the exact methods of Neumann and Schwindt (2003) and of Laborie (2003). Section 9.3 is devoted to the financing problem and Sect. 9.4 to the project scheduling problem with consumption and production of resources. Section 9.5 presents our conclusion.

9.2 The Project Scheduling Problem with Inventory Constraints

Neumann and Schwindt first introduced the project scheduling problem with inventory constraints. Inventory constraints refer to non-renewable resources, which can be stocked and have minimum and maximum prescribed stocks. Neumann and Schwindt proposed a branch and bound method for solving this problem and truncated it to a filtered beam search heuristic. Laborie subsequently proposed a constraint programming method for solving the problem. In this section, we introduce the model and briefly describe both methods.

9.2.1 The Model

In this model, activities are replaced by events. An event might be, for example, the start or the completion of an activity. There are also storage resources. r_{ik} is the demand for storage resource k by event i. If r_{ik} is strictly positive, it corresponds to the production by event i of r_{ik} units of resource k. If it is strictly negative, it corresponds to the consumption of -r_{ik} units of resource k. A project consists of n events $1, \ldots, n$. In addition the fictitious events 0 and $n+1$ which represent the start and the completion of the project are introduced. The nodes $V = \{0, 1, \ldots, n, n+1\}$ of a graph are associated with the events. There are minimum and maximum time lags between the events. A project network $N = (V, E, \delta)$ is obtained by introducing arcs between pairs of events. If there is a minimum time lag d_{ij}^{min} between the occurrences of two events i and j, an arc (i, j) is introduced with $\delta_{ij} := d_{ij}^{min}$. If there is a maximum time lag d_{ij}^{max} between the occurrences of two events i and j, an arc (j, i) is introduced with $\delta_{ji} := -d_{ij}^{max}$. A schedule S is a function of V into $\mathbb{Z}_{\geq 0}$ giving the starting times of the events. It has to be time-feasible and resource-feasible. S is time-feasible if it satisfies the inequality $S_j - S_i \geq \delta_{ij}$ for every arc (i, j) of E. It is resource-feasible if, for every time t and for every resource k, the inventory of the resource is between a minimum level R_k^{min} and a maximum level R_k^{max}. Given a schedule S, let $\mathscr{A}(S,t) = \{i \in V \mid t \geq S_i\}$ and $r_k(S,t) = \sum_{i \in \mathscr{A}(S,t)} r_{ik}$. The schedule S is resource-feasible if, for any time t and for every resource k: $R_k^{max} \geq r_k(S,t) \geq R_k^{min}$.

A schedule is time-feasible if it satisfies the inequality constraints, resource-feasible if it satisfies the resource constraints and it is said to be *feasible* if it is both time-feasible and resource-feasible. We remark that it can be supposed without loss of generality that $R_k^{min} = 0$, if it is supposed that, for any k, event 0 consumes R_k^{min} units of resource k. In literature, the problem is denoted $PSs|temp|C_{max}$.

9.2.2 The Exact Method of Neumann and Schwindt

Neumann and Schwindt (2003) propose a branch and bound method which enumerates alternatives for avoiding stock shortages and surpluses by introducing *disjunctive precedence constraints* between some disjoint sets of events A and B: A is before B if $\min\{S_i \mid i \in B\} \geq \min\{S_i \mid i \in A\}$. So a node in the tree corresponds to a set of these disjunctive precedence constraints. The authors show that if the set of schedules of a node is not empty, there exists an earliest schedule S respecting the initial precedence constraints and the disjunctive precedence constraints. Moreover they explain how to compute this earliest schedule by adjusting the starting times of events. If this schedule is resource-feasible, it is feasible, so we backtrack. Otherwise, there exists a time t such that $\mathscr{A}(S,t)$ is a surplus set or a shortage set.

That is: $r_k(S,t) > R_k^{max}$ or $r_k(S,t) < R_k^{min}$. They consider the smallest t satisfying the previous condition.

Let us explain the case of a surplus set. Here it is necessary to postpone a minimal subset B of events (B included in $\mathscr{A}(S,t)$). B is composed of events i with r_{ik} strictly positive (production events) and it is minimal in the sense that the surplus conflict is solved, but it is not solved for some proper subset of B. Of course it is necessary to enumerate several alternatives for B. To postpone B, they consider the set A of events which are not in $\mathscr{A}(S,t)$ and with r_{ik} strictly negative (consumption events).

A first lower bound of the makespan is associated with the earliest schedule. A second lower bound uses the earliest schedule S and also a latest schedule LS defined by $LS_i := \overline{d}_i$, where $\overline{d}_i$ is a deadline of event i. A lower bound of the stock is obtained by starting the consumption events as early as possible, given by S, and the production events as late as possible, given by LS. Similarly an upper bound of the stock is obtained by starting the consumption events as late as possible, given by LS, and the production events as early as possible, given by S. If at some time t it can be proved that the stock will be insufficient or too large, there is no solution and the corresponding node of the tree can be cancelled.

The experimental analysis of Neumann and Schwindt (2003) shows that their method can solve problem instances with 100 events and five storage resources. These data were generated by the authors, and are composed of 360 projects. Twelve projects have not been solved optimally. The larger problems are solved by replacing the minimal subset B by a subset of cardinality one.

9.2.3 The Exact Method of Laborie

The scheduling problems which are considered by Laborie (2003) are very general. They include the Resource Constrained Project Scheduling Problem and the project scheduling problem with inventory constraints. His method propagates resource constraints within a constraint programming approach. Its application to the project scheduling problem with inventory constraints is presented below. Laborie (2003) refers to a resource reservoir, because the maximum and minimum levels are given. Most of the techniques in the literature refine the execution intervals of activities. These are the cases of edge finding or energy based reasoning devoted to renewable resources. But generally, at the start of the search, no activity intervals can yet be deduced. So Laborie (2003) focuses on the precedence relations between events rather on their absolute positions in time, as we explain below. It is a method that is complementary to the aforementioned techniques.

The search space consists of a global search tree. The method consists in iteratively refining a partial schedule. A partial schedule is composed of a set of events, temporal constraints and resource constraints. The main tools of Laborie (2003) are time-tabling, the resource graph, and the balance constraints.

Time-tabling is a propagation technique which relies on the computation of upper and lower bounds at any time t for the use of every resource k. It can limit the domains of the start and completion times of activities by removing the dates that would necessarily lead to an over-consumption or under-consumption of some resource by an event.

The resource graph RG is composed of two sets of arcs: $RG = (V, E_{\leq}, E_{<})$, where $E_{<}$ is included in $E_{\leq}$, and

- $E_{\leq}$ is the set of couples (i, j) such that: $S_i \leq S_j$,
- $E_{<}$ is the set of couples (i, j) such that: $S_i < S_j$.

The resource graph expresses precedence relations between events. The graph on a resource is designed to gather together all the precedence relations between events on the resource. They may come from initial temporal constraints, from deductions, and from branching decisions. When new precedence relations are introduced, the transitive closure of the resource graph is maintained thanks to a matrix.

This graph means that balance constraints associated with events can be evaluated. The basic idea is to compute, for each event using a specific resource, upper and lower bounds on the resource level just before and just after this event. An event i is safe for some resource if the upper bound of the resource level just before i and just after i is smaller than the minimum capacity and the lower bound of the resource level before i and after i is larger than the minimum capacity. When all events of a resource are safe, the reservoir constraint of this resource is satisfied.

The balance constraint can reveal three types of information: dead-ends, new bounds for time variables, and new precedence relations. For instance, when the upper bound of the resource level just before event i is strictly smaller than the minimal level, we get a dead-end. We can also get new bounds on time variables. For instance, if the resource level before i in the partial schedule is smaller than the minimum level, production events need to be scheduled before i. The earliest dates can be computed at which sufficient resources might be available for processing i. It will depend on the earliest dates at which production events can be scheduled. Finally if the processing of event j after event i would provoke a dead-end, i must be scheduled before j. So we can add the corresponding precedence constraint.

Branching is based on precedence relations. It involves choosing two events i and j. Laborie chooses either to process i before j or j before i, one of both precedences being strict. i is chosen as a critical event, for instance an event consuming or producing a large quantity of resources. The choice which is sophisticated is explained in the paper.

Laborie's method has been implemented in the ILOG Scheduler, a C++ library for constraint-based scheduling (see Le Pape 1994). It can solve to optimality all the instances of Neumann and Schwindt (2003), including the 12 previously open instances in less than 10 s. To resume this method is elaborated and innovative. It is also very efficient in practice.

9.3 The Financing Problem

The financing problem was a subject of study prior to the project scheduling problem with inventory constraints (see Carlier and Rinnooy Kan 1982; Słowiński 1984). This problem aims to model the financing of some project being realized. It is a special case, insofar as the dates of production events are given whenever there are precedence constraints between consumption events. It is solved using a polynomial algorithm known as the shifting algorithm. Deadlines can also be taken into account. The model, the shifting algorithm and a discussion are reported below.

9.3.1 The Model

Dependent tasks have to be scheduled in a minimal makespan. Task i consumes at the outset a quantity r_i of a non-renewable resource, which can represent money. Task i is replaced by its starting event, denoted without ambiguity i. Initially at time $\tau_1 = 0$, b_1 units of the resource are available. $b_2, b_3, \ldots, b_q$ additional units of the resource become available at dates $\tau_2, \tau_3, \ldots, \tau_q$. A precedence graph $N = (V, E, \delta)$ is associated with the problem. V contains the set of consumption events and the two fictitious events 0 and $n+1$. We suppose that the minimum capacity is 0 and the maximum capacity is infinite. When money is involved, it is better to have as large a stock as possible! At first we will suppose that N does not contain any arc $(i, 0)$. Such an arc will model a deadline $\overline{d}_i$.

9.3.2 Time-Feasible Schedules

Let us denote d_{ij} the value of a maximal path from i to j in N. $ES = \{ES_i = d_{0i} \mid i \in V\}$ is the earliest time-feasible schedule and $LS = \{LS_i = d_{0,n+1} - d_{i,n+1} \mid i \in V\}$ the latest time-feasible schedule. It is also well known that there exists a latest schedule with makespan $d_{0,n+1} + \Delta$ which is defined by $LS(\Delta) = \{LS_i + \Delta \mid i \in V\}$ when Δ is positive.

9.3.3 Feasible Schedule

A time-feasible schedule $S = (S_0, S_1, \ldots, S_{n+1})$ is resource-feasible if the following condition is satisfied: for any t : $RE(t) = \sum_{\{i \mid S_i \leq t\}} r_i \leq A(t) = \sum_{\{\mu \in \{1,\ldots,q\} \mid \tau_\mu \leq t\}} b_\mu$. In other words, the requirement curve is below the availability curve (see Fig. 9.1). Note that $r(S, t)$ introduced in Sect. 9.2.1 is equal to $A(t) - RE(t)$.

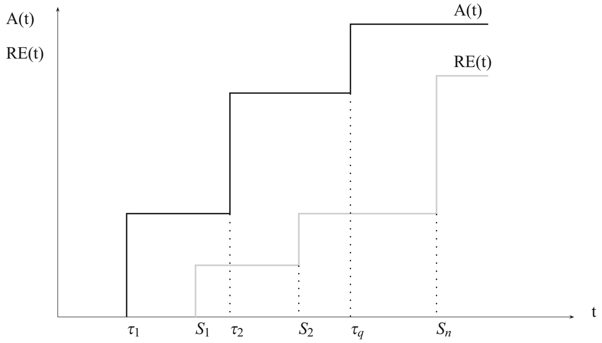


Fig. 9.1 Graphical interpretation of feasible schedules

9.3.4 The Shifting Algorithm

In the shifting algorithm, the latest schedule is shifted in order to satisfy the feasibility condition.

Algorithm 9.1: The shifting algorithm

```
begin
    if ($\sum_{i \in V} r_i > \sum_{\mu \in \{1,\dots,q\}} b_\mu$) then
        write there is no feasible schedule
    else
        Compute the latest schedule $LS := \{LS_i = d_{0,n+1} - d_{i,n+1} / i \in V\}$;
        $A(\tau_1) := b_1$;
        for $\mu := 2$ to $q$ do
            $A(\tau_\mu) := A(\tau_{\mu-1}) + b_\mu$
        $\mu := 1; \Delta := 0; RE := 0$;
        for $i := 1$ to $n$ do
            $RE := RE + r_i$;
            while $A(\tau_\mu) < RE$ do
                $\mu := \mu + 1$
            if $\Delta < (\tau_\mu - LS_i)$ then
                $\Delta := (\tau_\mu - LS_i)$
```

9.3.5 An Instance

We consider a precedence graph with four tasks 1, 2, 3 and 4 with processing times 7, 8, 9 and 6. Task 1 precedes task 4, task 2 precedes tasks 3 and 4. Assume that

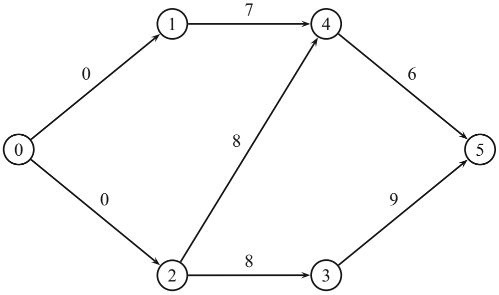


Fig. 9.2 Example with four tasks such that $r_1 = r_2 = r_3 = r_4 = 7$ and $\tau_1 = 0, \tau_2 = 5$ and $\tau_3 = 10, b_1 = b_2 = b_3 = 10$

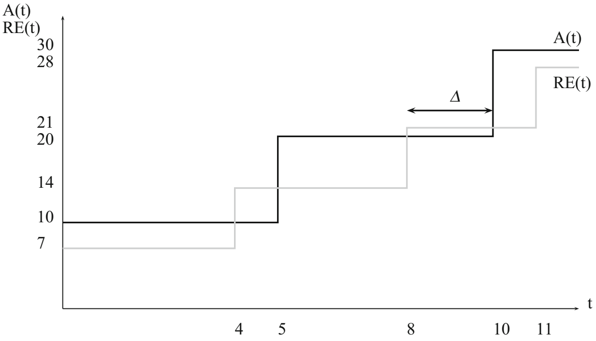


Fig. 9.3 Applying the shifting algorithm

any task consumes seven units of resource. Moreover, at times $\tau_1 = 0, \tau_2 = 5$ and $\tau_3 = 10, b_1 = b_2 = b_3 = 10$ units of resource become available (see Fig. 9.2).

The shifting algorithm computes the latest schedule $LS : LS_0 = 0, LS_1 = 4$, $LS_2 = 0, LS_3 = 8, LS_4 = 11$ and $LS_5 = 17$. Next, we determine for any task i, the smallest time τ_μ such that $A(\tau_\mu) \geq RE(LS_i)$. Then Δ is the smallest value such that $A(LS_i + \Delta) \geq RE(LS_i)$ for any i. Therefore, $\Delta = 2$ and the optimal schedule built by the algorithm is $S_0 = 0, S_1 = 6, S_2 = 2, S_3 = 10, S_4 = 13$ and $S_5 = 19$ (see Fig. 9.3).

9.3.6 Discussion

The shifting algorithm computes an optimal schedule when $(\sum_{i \in V} r_i \leq \sum_{1 \leq \mu \leq q} b_\mu)$. This is because it is better to finance activities as late as possible, which is ensured by restricting the research to schedules $LS(\Delta)$. The algorithm computes the lowest possible positive value of Δ. If we have several resources, the generalization is straightforward. We can also take into account deadlines (see

Carlier 1989). Some other generalizations are presented in Carlier (1989) and Carlier (1984).

9.4 The Project Scheduling Problem with Consumption and Production of Resources

We now consider the particular case where there are no maximum stocks. It can model the RCPSP and the project scheduling problem with inventory constraints. So it remains $\mathcal{NP}$-hard. The case with one resource can be polynomially solved when the precedence graph is series-parallel. In the general case, when a linear order on all events is given, there exists an earliest schedule which can be computed efficiently because we can replace the linear order by precedence constraints. If we have a linear order on consumption events, there also exists an earliest schedule, which can be computed efficiently when valuations of arcs are positive or null. If we have a linear order on production events, then a latest schedule exists. We report all these results below.

9.4.1 The Model

We will now explain how we can model a project scheduling problem with inventory constraints by a project scheduling problem with consumption and production of resources. Let us consider an instance of the project scheduling problem with inventory constraints and k a resource with $R_k^{min} = 0$. We replace resource k by two new resources denoted k_1 and k_2. If event i produces a quantity r_{ik} in the original instance, it will produce a quantity $r_{ik_1} = r_{ik}$ of resource k_1 in the new instance and produce a quantity $r_{ik_2} = -r_{ik}$ of resource k_2. The initial availability of resource k_2 is R_k^{max}.

We can also model the resource constrained project scheduling problem (RCPSP). Let us consider an instance of RCPSP. We have to schedule a set of activities subject to precedence constraints and to renewable resource constraints. An activity i is replaced by two events i_1 and i_2. i_1 corresponds to the start of activity i, and i_2 to its completion. Of course, between i_1 and i_2 there is a time lag equal to the processing time of activity i. If activity i requires a quantity r_{ik} of the renewable resource k, event i_1 will consume a quantity r_{ik} of the renewable resource k and event i_2 will produce a quantity r_{ik}.

Abdel-Wahab and Kameda (1978) have proved that the existence scheduling problem with consumption and production of resources becomes polynomial when there is one resource only and the precedence graph is series-parallel. But it remains $\mathcal{NP}$-hard even if the consumption and production of resources are equal to one and the time lags are equal to one (see Abdel-Wahab and Kameda 1978; Sethi 1975).

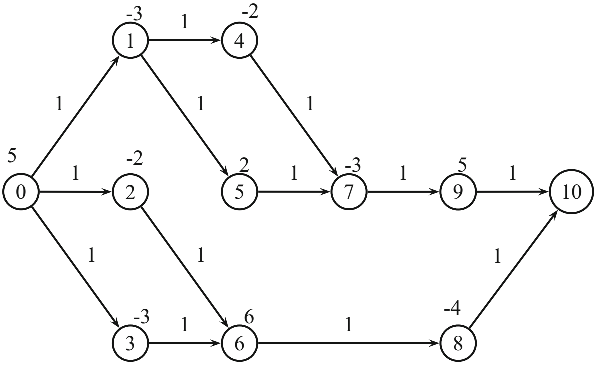


Fig. 9.4 An instance of the project scheduling problem with production and consumption of resources

9.4.2 Example

Figure 9.4 reports an example where there are nine events. The precedence graph therefore has eleven nodes when we add the two fictitious nodes 0 and 10. It can be seen that there is only one resource with initial availability 5. There are also three consumption events without predecessors, and no production events without predecessors. If we start at first event 1, event 5 can be started and the availability of the resource becomes equal to 4. It is not sufficient to start both events 2 and 3. Consequently, event 6 cannot start and it is not possible to build a schedule in this way. So, events 2 and 3 need to be started before event 1, because event 6 can start after them. The schedule can be completed in a straightforward manner by starting after events $1, 5, 4, 7, 9$, and 8.

9.4.3 The Earliest Schedule Associated with a Complete Linear Order

A complete linear order $\rho = (i_1, i_2, \ldots, i_n)$ on events is said to be feasible if there exists a schedule S such that $S_{i_1} \leq S_{i_2} \leq \ldots \leq S_{i_n}$. To simplify the presentation, we assume throughout this section that there is only one resource and that $\rho = (1, 2, \ldots, n)$. We first introduce the set of precedence arcs $E_\rho = \{(1,2), \ldots, (n-1, n)\}$ valued by 0. We will now show that the linear order can be replaced by a set E_γ of precedence arcs valued by 0.

Let us suppose we want to start event j and that $\sum_{i \in \{0,1,\ldots,j\}} r_i < 0$. If there is no production event after j in the linear order, there is no feasible solution, because it is necessary to start such a production event at the same time as j. We have to force some production event to start at the same time as j by introducing an arc (j', j)

with valuation 0. j' is the smallest event larger than j such that $\sum_{i \in \{0,1,\ldots,j'\}} r_i \geq 0$. We therefore replace the linear order by a set of precedence arcs E_γ. The problem is modeled by the graph $G_{\rho\gamma} = (V, E + E_\rho + E_\gamma)$. So an earliest schedule can be computed in $\mathcal{O}(nm)$ if there are n events and m arcs by a modified label correcting algorithm.

9.4.4 The Earliest Schedule Associated with a Linear Order on Consumption Events

The drawback of the previous method is that there are a large number of complete linear orders. But as we will now see, the operation can be restricted to consumption events only. We denote by ρ_c a complete linear order of consumption events. To simplify the presentation we assume that $\rho_c = (1, 2, \ldots, c)$. A first result is that, provided ρ_c is feasible, there exists an earliest schedule. Moreover this earliest schedule can be computed efficiently when the valuations of arcs are strictly positive or null. In the general case, that is to say when valuations can also be negative, we get only a pseudo-polynomial algorithm.

Let us first suppose that the valuations of arcs are strictly positive. We then associate a cut $Cut(j)$ with a consumption event j as the partition of nodes in the graph $G_{\rho_c} = (V, E + E_{\rho_c})$: $\overline{Succ}(j)$ and $V \setminus \overline{Succ}(j)$ (an event i is in $\overline{Succ}(j)$ if there exists a path from j to i in the graph). j does not belong to $\overline{Succ}(j)$. We say that $Cut(j)$ is feasible if: $\sum_{i \in V \setminus \overline{Succ}(j)} r_i \geq 0$. Of course if there exists an infeasible cut there is no feasible schedule. We have proved that conversely if all these cuts are feasible, there exists a feasible schedule. In order for a graph to be feasible, it must not contain any directed cycle. So there is at least one event without a predecessor. The strategy for calculating the earliest start time schedule is to start the production events as soon as possible. A consumption event without a predecessor may be able to start if the resource availability is sufficient. Otherwise it has to wait for the starting time of some production event. It can be proved that the method works if no infeasible cut exists. The complexity of the method is $\mathcal{O}(n^2)$.

Let us suppose now that the valuations of arcs are strictly positive or null. Of course if there are directed cycles with null valuations, all the events in such a cycle can be replaced by a new event. So it can be assumed that there are no such cycles. The previous algorithm cannot be applied because production events which are successors of an event with a valuation 0 can start at the same time as the consumption event. An algorithm which takes into account this latter property solves this case in $\mathcal{O}(n^3)$.

The remaining case is when there exist negative weight arcs. The earliest schedule can be computed if it exists in pseudo-polynomial time. It is a question of adjusting dates. In a first step the previous algorithms are used without taking into account the negative arcs. Then in a second step negative arcs are taken into account by modifying the starting dates. Finally we iterate both steps for as long as required. We do not know if the corresponding problem is $\mathcal{NP}$-hard in the weak sense. Determining the status of this problem is something of a challenge.

9.5 Conclusions

This chapter has been devoted to the storage scheduling problem. We have considered three models: the project scheduling problem with inventory constraints, the financing problem, and the project scheduling problem with production and consumption of resources. The exact methods for the project scheduling problem with inventory constraints are very powerful but so far they have been applied to one benchmark of instances only. It is important to improve them and test them on larger or different benchmarks. One perspective would be to consider the project scheduling problem with consumption and production of resources by the implicit enumeration of linear orders of consumption events. Classical lower bounds from the literature for renewable resources (Carlier et al. 2010) might also be integrated. Our current work is taking us in this direction. Another active field is that of the financing model. Recent papers have looked at this with additional renewable resources, and polynomial algorithms and complexity results have been presented (see Briskorn et al. 2013; Drotos and Kis 2013). Determining the frontier between easy and hard problems is also very challenging. We firmly believe that further research on the storage scheduling problem is needed because of its great potential for modeling and consequently for use in different applications.

Acknowledgements This work was carried out within the framework of the Labex MS2T, which was funded by the French Government, through the program Investments for the future managed by the National Agency for Research (Reference ANR-11-IDEX-0004-02).

References

Abdel-Wahab HM, Kameda T (1978) Scheduling to minimize maximum cumulative costs subject to series-parallel precedence constraints. Oper Res 26:141–158

Błażewicz, cf. Chap. 1: Błażewicz J, Cellary W, Słowiński R, Węglarz J (1986) Scheduling under resource constraints: deterministic models. Baltzer, Basel

Briskorn D, Jaehn F, Pesch E (2013) Exact algorithms for inventory constrained scheduling on a single machine. J Sched 16:105–115

Carlier J (1984) Problèmes d'ordonnancements à contraintes de ressources: algorithmes et complexité. Habilitation thesis, Université de Paris VI, France

Carlier J (1989) Scheduling under financial constraints. In: Słowiński R, Węglarz J (eds) Advances in project scheduling. Elsevier, Amsterdam, pp 187–224

Carlier J, Rinnooy Kan AHG (1982) Financing and scheduling. Oper Res Lett 1:52–55

Carlier J, Moukrim A, Xu H (2009) The project scheduling problem with production and consumption of resources: a list-scheduling based algorithm. Discrete Appl Math 157:3631–3642

Carlier J, Moukrim A, Xu H (2010) A branch and bound method for the generalized resource constrained project scheduling problem. In: Proceedings of the 10th international workshop on project management and scheduling, Poznań, pp 135–140

Drotos M, Kis T (2013) Scheduling of inventory releasing jobs to minimize a regular objective function of delivery times. J Scheduling 16:337–346

Graham R, Lawler EL, Lenstra JK, Rinnooy Kan AHG (1979) Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann Discrete Math 5:287–326

Koné O, Artigues C, Lopez P, Mongeau M (2013) Comparison of mixed integer linear programming models for the resource constrained project scheduling problem with consumption and production of resources. Flex Serv Manuf J 25:25–47

Laborie P (2003) Algorithms for propagating resource constraints in AI planning and scheduling: existing approaches and new results. Artif Intell 143:151–188

Le Pape C (1994) Implementation of resource constraints in ILOG Schedule: a library for the development of constraint-based scheduling systems. Intell Syst Eng 3:55–66

Neumann K, Schwindt C (2003) Project scheduling with inventory constraints. Math Method Oper Res 56:513–533

Neumann K, Schwindt C, Zimmermann J (2003) Project scheduling with time windows and scarce resources: temporal and resource-constrained project scheduling with regular and nonregular objective functions. Springer, Berlin

Sethi R (1975) Complete register allocation. SIAM J Comput 4:226–248

Słowiński R (1984) Preemptive scheduling of independent jobs on parallel machines subject to financial constraints. Eur J Oper Res 15:366–373

Chapter 10
Continuous Resources

Grzegorz Waligóra and Jan Węglarz

Abstract In this chapter project scheduling under an additional continuous resource is considered. In particular, we deal with discrete-continuous project scheduling problems to minimize the project duration. These problems are characterized by the fact that activities of a project simultaneously require discrete and continuous resources for their execution. A class of the problems is considered, where the number of discrete resources is arbitrary, and there is one continuous, renewable resource, whose total amount available at a time is limited. Activities are nonpreemptable, and the processing rate of an activity is a continuous, increasing function of the amount of the continuous resource allotted to the activity at a time. Theoretical results for the cases of convex and concave processing rate functions of activities are presented, and the methodology developed for solving the problems with concave functions is described in detail. Some conclusions and final remarks are given.

Keywords Continuous resources • Demand division • Feasible sequence • Makespan minimization • Processing rate • Project scheduling

10.1 Introduction

In the classical project scheduling problems it is assumed that resources can be assigned to activities in amounts from a given finite set only (i.e., in discrete numbers of units). Such resources are called *discrete* (or discretely-divisible). However, in many practical situations resources can be allotted to activities in arbitrary numbers from a given interval (i.e., in real numbers). Such resources are called *continuous* (or continuously-divisible). Situations of this type occur when, e.g., activities are processed by parallel processing units driven by a common (electric, pneumatic, hydraulic) power source, like commonly supplied grinding or mixing machines, electrolytic tanks, or refueling terminals. Also in computer systems, where multiple

G. Waligóra (✉) • J. Węglarz
Institute of Computing Science, Poznan University of Technology, Poznan, Poland
e-mail: grzegorz.waligora@cs.put.poznan.pl; jan.weglarz@cs.put.poznan.pl

C. Schwindt, J. Zimmermann (eds.), *Handbook on Project Management and Scheduling Vol. 1*, International Handbooks on Information Systems,
DOI 10.1007/978-3-319-05443-8_10

processors share a common primary memory, if it is a paged-virtual memory system and the number of pages goes into hundreds, then primary memory can be treated as a continuous resource (Węglarz 1980). On the other hand, the processors themselves can be considered as a continuous resource in scalable (SPP) or massively parallel (MPP) systems when the number of them is huge (hundreds or even thousands). More recently, so called power- (or energy-) aware scheduling problems have been considered, where a set of processors is a discrete resource and power (energy) is a continuous resource (see, e.g., Różycki and Węglarz 2014).

In general, two activity processing models appear in the literature. In the first model—the *processing time vs. resource amount*, the activity duration is a function of the amount of a continuous resource allotted to this activity. This model is a straightforward generalization of the well-known discrete time-resource tradeoff model. It is implicitly assumed that the resource amount allocated to an activity does not change during its execution. Within the approach based on this model, the existence of some polynomially solvable cases of machine scheduling problems for linear functions have been proved. In the second model—the *processing rate vs. resource amount*, the processing rate of an activity is a function of the amount of a continuous resource allotted to this activity at a time. In this case, the amount of the continuous resource allotted to an activity may change during its execution. A fundamental result for this model and a renewable resource can be found in Węglarz (1976), whereas for a doubly constrained resource in Węglarz (1981). In both these papers activities are assumed to be (continuously-)preemptable.

From between the two abovementioned models, the processing rate vs. resource amount model is more natural in the majority of practical situations, since it reflects directly the temporary nature of renewable resources. As examples, functions like rotational speed vs. electric current, or progress rate vs. number of primary memory pages allotted to a program can be given. This temporary character is vital, as it is often ignored in practice in the case of some doubly constrained resources, which can be then treated as nonrenewable ones. Money is a good example of such a resource where usually only the total consumption is taken into account, whereas it also has a temporary nature as it may be limited in a given period. Even the most typical continuous, renewable resource, which is power, is also, in general, doubly constrained, since its consumption, i.e., energy, is also limited. Moreover, the processing rate vs. resource amount model enables to perform a deeper analysis of the properties of optimal schedules, and can even lead to analytical results in some cases. Because of that, it is sometimes reasonable to treat a discrete resource as a continuous one in order to use this model. Such an approach may be applied when there are sufficiently many allotments of the discrete resource for processing an activity, e.g., in SPP or MPP systems.

The chapter is organized as follows. In Sect. 10.2 we recall the most important results concerning the problem of allocating the continuous renewable resource among independent activities to minimize the project duration, in the case when it is the only limited resource (in the absence of limited discrete resources). In Sect. 10.3 we define the discrete-continuous project scheduling problem in which both discrete and continuous resources are limited, and the activities are precedence-related.

Section 10.4 reports the basic results for the defined problem obtained for two classes of the processing rate functions of activities: convex and concave functions. Since it is shown that the case of convex functions is trivial, we focus on the case with concave functions in Sect. 10.5, and present the methodology for solving the problem developed for this class of functions. Section 10.6 contains some conclusions and final remarks.

10.2 Continuous Resource Allocation

In this section we recall very briefly main theoretical results concerning the continuous resource allocation. The results relate to independent activities, the processing rate vs. resource amount activity processing model, and the minimization of the project duration.

We assume that one continuous, renewable resource is available. The availability of the resource over time is constant and equal to 1. The resource can be allotted to activities in (arbitrary) amounts from the interval [0, 1]. The amount (unknown in advance) of the continuous resource allotted to activity i at time t is denoted by $u_i(t)$, and $\sum_{i=1}^{n} u_i(t) = 1$ for any t. The resource amount $u_i(t)$ determines the processing rate of activity i, which is described by the following equation:

$$\dot{x}_i(t) = \frac{dx_i(t)}{dt} = f_i[u_i(t)], \quad x_i(0) = 0, \quad x_i(C_i) = \tilde{x}_i \tag{10.1}$$

where:

$x_i(t)$ is the state of activity i at time t;
f_i is a continuous, increasing function, such that $f_i(0) = 0$;
$u_i(t)$ is the continuous resource amount allotted to activity i at time t;
C_i is the completion time (unknown in advance) of activity i;
$\tilde{x}_i$ is the processing demand (final state) of activity i.

State $x_i(t)$ of activity i at time t is an objective measure of work related to the processing of activity i up to time t. It may denote, e.g., the number of man-hours already spent on processing activity i, the volume (in cubic meters) of a constructed building, the number of standard instructions in processing computer program i etc.

The problem is to find an allocation of the continuous resource to activities that minimizes the project duration. The continuous resource allocation is defined by a piecewise continuous, nonnegative vector function $\mathrm{u}(t) = [u_1(t), u_2(t), \ldots, u_n(t)]$, whose values $\mathrm{u}^* = \left[u_1^*, u_2^*, \ldots, u_n^*\right]$ are (continuous) resource allocations corresponding to C_{max}^*—the minimal value of C_{max}. Completion of activity i requires that:

$$x_i\ (C_i) = \int_0^{C_i} f_i\ [u_i\ (t)]\ dt = \tilde{x}_i \tag{10.2}$$

The following result, proved by Węglarz in (1976), is fundamental for the continuous resource allocation problem:

Theorem 10.1. *The minimum project duration* C^*_{max} *as a function of final states of activities* $\tilde{\mathrm{x}} = (\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_n)$ *can always be given by:*

$$C^*_{max}(\tilde{\mathrm{x}}) = \min\{C_{max} > 0 : \tilde{\mathrm{x}}/C_{max} \in conv(Y)\}$$

where $conv(Y)$ *is the convex hull of* Y*, and set* Y *is defined as:*

$$Y = \{\mathrm{y} : y_i = f_i\left(u_i\right), u_i \geq 0, i = 1, 2, \ldots, n, and \sum_{i=1}^{n} u_i \leq 1\}$$

$C^*_{max}(\tilde{\mathrm{x}})$ *is a convex function.*

Two immediate corollaries follow (Węglarz 1976):

Corollary 10.1. *For convex processing rate functions of activities the project duration is minimized by sequential processing of all activities, each of them using the total available amount of the continuous resource.*

Corollary 10.2. *For concave functions* f_i*,* $i = 1, 2, \ldots, n$*, the project duration is minimized by fully parallel processing of all activities using the following resource amounts:*

$$u_i^* = f_i^{-1}\left(\tilde{x}_i / C^*_{max}\right) \quad (i = 1, 2, \ldots, n) \tag{10.3}$$

where C^*_{max} *is the unique positive root of the equation:*

$$\sum_{i=1}^{n} f_i^{-1}\left(\tilde{x}_i / C_{max}\right) = 1 \tag{10.4}$$

Let us comment briefly on both the Corollaries. Firstly, Corollary 10.1 holds, in fact, for all functions fulfilling the condition $f_i \leq b_i u_i, b_i = f_i\left(1\right), i = 1, 2, \ldots, n$, i.e., functions not greater than a linear function. In the sequel, the results presented for convex functions hold for functions fulfilling the above condition.

Secondly, Corollary 10.2 identifies very important cases in which an optimal resource allocation can be found in an efficient way. Generally speaking, these are the cases when Eq. (10.4) can be solved analytically. From among them the ones in which Eq. (10.4) is an algebraic equation of an order $\leq$ 4 are of special importance. This is, for example, the case of power processing rate functions of the form: $f_i\left(u_i\right) = b_i u_i^{1/\alpha_i}, \alpha_i \in \{1, 2, 3, 4\}, i = 1, 2, \ldots, n$. Using these functions we can model activity processing rates in a variety of practical problems, e.g., those arising in multiprocessor scheduling with memory allocation (see Węglarz 1980).

It should also be noticed that in both the above Corollaries preemptability of activities is of no importance. In Corollary 10.1 activities are processed sequentially, each of them using the total available amount of the continuous resource. In Corollary 10.2 activities are processed using constant resource amounts [given by Eq. (10.3)] from their starts to their completions. As a result, allowing activity preemptions does not affect optimal schedules. In the remainder of the chapter we deal with nonpreemptable activities, however, preemptable ones can be considered as well, especially in the context of computer systems.

In the next sections we will show how the results recalled above can be applied to the case when activities are nonpreemptable and precedence-related, and additional discrete, renewable resources occur.

10.3 Discrete-Continuous Project Scheduling

In Sect. 10.2 properties of optimal schedules have been given, proved for the case where a single continuous resource is the only limited resource, and the independent activities may be performed in parallel. However, also discrete limited resources can appear, as well as precedence constraints between activities, which can restrict the execution order of the activities. Discrete-continuous project scheduling problems arise when activities of a project simultaneously require discrete and continuous resources for their execution.

The *discrete-continuous resource-constrained project scheduling problem* (DCRCPSP) is defined as follows (Waligóra 2011). Given is a project consisting of n precedence-related, nonpreemptable activities, which require resources of two types: discrete and continuous ones. We assume that K discrete resources are available and r_{ik}, $i = 1, 2, \ldots, n$; $k = 1, 2, \ldots, K$, is the (fixed) discrete resource request of activity i for resource k. The total number of units of discrete resource k available in each time period is R_k, $k = 1, 2, \ldots, K$. A project is represented by an activity-on-node (AoN) directed, acyclic, and topologically ordered graph $G = (V, E)$, where the set of nodes V corresponds to the set of activities, and the set of arcs E represents precedence constraints. The activities are subject to finish-to-start precedence constraints of the type (i, j) with zero minimum time lags. The precedence constraints of activity i with other activities are defined by two sets: set $Pred(i)$ of direct predecessors of activity i, and set $Succ(i)$ of direct successors of activity i. One continuous resource is available, and the processing rate of each activity at a time is defined by the amount of the continuous resource allotted to the activity, according to Eq. (10.1). Thus, each activity of the project is characterized by its processing demand, processing rate function, discrete resource requests, and precedence constraints with other activities. It is assumed that all activities and resources are available from the start of the project. The problem is to find a precedence- and discrete resource-feasible schedule and, simultaneously, a continuous resource allocation, that minimize the project duration.

Using the project scheduling classification scheme presented in Brucker et al. (1999), the notation of the considered problem is $PSc|prec|C_{max}$.

All the parameters of the DCRCPSP are summarized in Table 10.1.

Table 10.1 Parameters of the DCRCPSP

Symbol	Definition
n	Number of activities
$G = (V, E)$	Directed graph with node set V and arc set E
$(i, j) \in E$	Precedence constraint between activities i and j
$Pred(i)$	Set of immediate predecessors of activity i
$Succ(i)$	Set of immediate successors of activity i
K	Number of discrete resources
R_k	Number of available units of discrete resource k
r_{ik}	Request for discrete resource k by activity i
f_i	Processing rate function of activity i
$\tilde{x}_i$	Processing demand of activity i
S_i	Starting time of activity i
C_i	Completion time of activity i
$p_i = C_i - S_i$	Duration of activity i

10.4 Basic Results for DCRCPSP

In this section we recall after Waligóra (2014) the most important theoretical results for the DCRCPSP. We will show how the results presented in Sect. 10.2 can be applied to the problem defined in Sect. 10.3 for the considered classes of processing rate functions of activities: convex (Sect. 10.4.1) and concave (Sect. 10.4.2) functions.

10.4.1 Convex Processing Rate Functions

In this case the following lemma was proved, which follows directly from Corollary 10.1.

Lemma 10.1. *In the DCRCPSP with convex processing rate functions, the project duration is minimized by a sequential configuration of activities, in which activities are processed one after another in a precedence-feasible order, each of them using the total available amount of the continuous resource.*

Thus, for convex processing rate functions the solution of the DCRCPSP is trivial, since any precedence-feasible sequence of activities leads to an optimal schedule. Obviously, if the condition $r_{ik} \leq R_k, i = 1, 2, \ldots, n; k = 1, 2, \ldots, K$, holds, then the discrete resource constraints are not violated because only one activity is performed at a time.

10.4.2 Concave Processing Rate Functions

In this case another lemma holds, following directly from Corollary 10.2:

Lemma 10.2. *In the DCRCPSP with concave processing rate functions, the project duration is minimized by a parallel precedence- and discrete resource-feasible configuration of activities, where the activities are processed using the resource amounts given by Eq. (10.3), and C^*_{max} is the unique positive root of Eq. (10.4). The parallel configuration means that as many activities as possible are processed in parallel, respecting precedence and discrete resource constraints.*

Thus, for concave functions the problem becomes more complicated since many different parallel schedules can be constructed. In order to solve the DCRCPSP for this class of functions, a special methodology has been developed which is described in Sect. 10.5.

10.5 Methodology Based on Feasible Sequences

The methodology for the DCRCPSP with concave processing rate functions of activities, based on Lemma 10.2, uses the idea of a *feasible sequence* introduced in Józefowska and Węglarz (1998) for discrete-continuous machine scheduling. Observe that a feasible schedule (i.e., a solution of a discrete-continuous project scheduling problem) can be divided into $\nu \leq n$ intervals defined by the completion times of the consecutive activities. Let Z_μ denote the combination of activities processed in parallel in the μ-th interval. Thus, a feasible sequence Θ of combinations $Z_\mu, \mu = 1, 2, \ldots, \nu$, is associated with each feasible schedule. The feasibility of such a sequence requires that:

- Each activity appears in at least one combination, i.e.:

$$\bigwedge_{i \in \{1,\ldots,n\}} \bigvee_{\mu \in \{1,\ldots,\nu\}} i \in Z_\mu$$

- Nonpreemptability of each activity is guaranteed, i.e.:

$$\bigwedge_{i \in \{1,\ldots,n\}} \bigwedge_{(\mu,\pi): i \in Z_\mu, i \in Z_\pi, \pi \geq \mu} (\mu = \pi) \vee \big(i \in Z_\rho, \rho = \mu + 1, \ldots, \pi - 1\big)$$

  which means that each activity appears in exactly one or in successive combinations in Θ

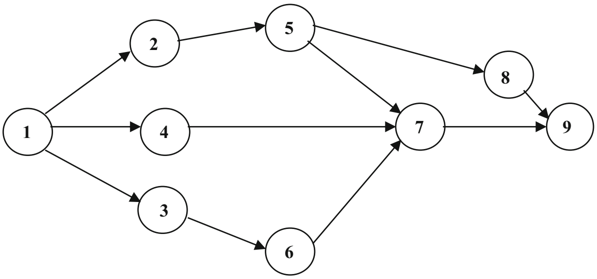


Fig. 10.1 An exemplary project

- Precedence constraints between activities are satisfied, i.e.:

$$\bigwedge_{(i,j)\in E} \left(i \in Z_\mu \wedge j \in Z_\pi\right) \Rightarrow (\pi > \mu)$$

- The number of units of each discrete resource $k, k = 1, 2, \ldots, K$, assigned to all activities in combination $Z_\mu, \mu = 1, 2, \ldots, \nu$, does not exceed R_k, i.e.:

$$\bigwedge_{k\in\{1,\ldots,K\}} \bigwedge_{\mu\in\{1,\ldots,\nu\}} \sum_{i\in Z_\mu} r_{ik} \leq R_k$$

Example 10.1. Consider a nine-activity project ($n = 9$) presented in Fig. 10.1. Assume that there is one discrete resource ($K = 1$) available in four units ($R_1 = 4$). The resource requests of activities are defined by vector $\mathrm{r}_1 = [3, 2, 1, 1, 3, 1, 4, 2, 1]$.

There are many feasible sequences for this exemplary project. One of them is, e.g.:

$$\Theta = \{1\}, \{2, 3, 4\}, \{3, 4\}, \{3, 5\}, \{5, 6\}, \{6, 8\}, \{7\}, \{9\}$$

The above form of a feasible sequence means that activity 1 is processed alone in the first interval, and this interval ends with the completion of this activity. The corresponding combination is $Z_1 = \{1\}$. Then activities 2, 3, and 4 are processed in parallel, and the completion of activity 2 ends the second interval. The corresponding combination is $Z_2 = \{2, 3, 4\}$. Next, only activities 3 and 4 are processed in the third interval, since the resource request of activity 5 does not allow it to be executed in this interval (although all its direct predecessors are finished). Thus, the corresponding third combination is $Z_3 = \{3, 4\}$. All the next combinations also fulfill the precedence and discrete resource constraints. The last interval ends with the completion of activity 9, and this is the end of the schedule. The resulting last combination is $Z_8 = \{9\}$. A schedule corresponding to the considered feasible sequence Θ is shown in Fig. 10.2.

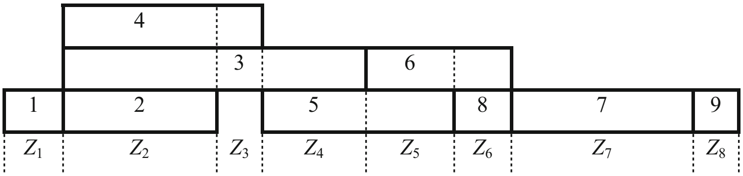


Fig. 10.2 Schedule corresponding to feasible sequence Θ

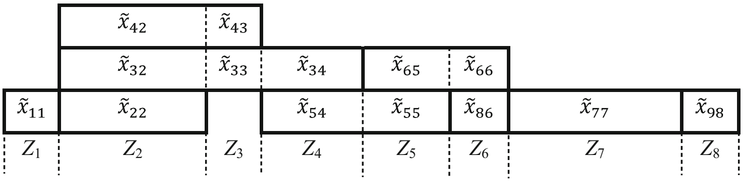


Fig. 10.3 Demand division for feasible sequence Θ

It is important to stress that at this moment the actual durations of activities are still unknown, since the continuous resource has not yet been allocated. However, the form of a feasible sequence gives the information which activities are processed in parallel in consecutive intervals. The continuous resource will be allocated on a basis on that information.

Next, for a given feasible sequence Θ, the processing demand $\tilde{x}_i$ of each activity i, $i = 1, 2, \ldots, n$, can be divided into parts $\tilde{x}_{i\mu} \geq 0$ (unknown in advance) corresponding to particular time intervals (combinations), i.e., $\tilde{x}_{i\mu}$ is a part of activity i processed in combination Z_μ. Such a division of processing demands of activities among successive intervals (combinations) for a given feasible sequence is called a *demand division*. The number of such divisions is, in general, infinite. The demand division for the feasible sequence Θ considered in the example is shown in Fig. 10.3.

Obviously, the sum of all parts of the processing demand of an activity must be equal to its total processing demand, i.e., $\tilde{x}_{11} = \tilde{x}_1, \tilde{x}_{22} = \tilde{x}_2, \tilde{x}_{32} + \tilde{x}_{33} + \tilde{x}_{34} = \tilde{x}_3$ etc. Then, an optimal division of processing demands of activities $\tilde{x}_i$, $i = 1, 2, \ldots, n$, among combinations in Θ can be found, i.e., a division that leads to a minimum-length schedule from among all feasible schedules generated by Θ. To this end, a nonlinear mathematical programming problem can be formulated in which the sum of the minimum-length intervals generated by consecutive combinations in Θ, as functions of the vector $\tilde{x}_\mu = \{\tilde{x}_{i\mu}\}_{i \in Z_\mu}$, is minimized subject to the constraints that each activity has to be completed.

Let $C^*_{max}(\tilde{\mathrm{x}}_\mu)$ be the minimum length of the part of the schedule generated by $Z_\mu \in \Theta$, and let Ψ_i be the set of all indices of Z_μ's such that $i \in Z_\mu$. The following mathematical programming problem (Problem P) finds an optimal demand division (and, in consequence, an optimal continuous resource allocation) for a given feasible sequence:

Problem P

$$\text{Min.}\quad C_{max} = \sum_{\mu=1}^{\nu} C^{*}_{max}\left(\tilde{\mathrm{x}}_{\mu}\right) \tag{10.5}$$

$$\text{s.t.}\quad \sum_{\mu \in \Psi_i} \tilde{x}_{i\mu} = \tilde{x}_i \;\; (i = 1, 2, \ldots, n) \tag{10.6}$$

$$\tilde{x}_{i\mu} \geq 0 \qquad (i = 1, 2, \ldots, n; \mu \in \psi_i) \tag{10.7}$$

where $C^{*}_{max}\left(\tilde{\mathrm{x}}_{\mu}\right)$ is the unique positive root of the equation:

$$\sum_{i \in Z_{\mu}} f_i^{-1}\left[\tilde{x}_{i\mu}/C^{*}_{max}(\tilde{\mathbf{x}}_{\mu})\right] = 1 \tag{10.8}$$

Constraints (10.6) correspond to the condition of fulfilling the processing demands of all activities, whereas constraints (10.7) ensure that the $\tilde{x}_{i\mu}$'s are nonnegative. As a sum of convex functions (see Theorem 10.1), the objective function (10.5) is also a convex function. Thus, the problem is to minimize a convex function subject to linear constraints.

After finding an optimal division $\tilde{x}_{i\mu}, i = 1, 2, \ldots, n, \mu \in \Psi_i$ of $\tilde{x}_i$'s, the corresponding optimal continuous resource allocation for combination Z_{μ} is given as:

$$u^{*}_{i\mu} = f_i^{-1}\left[\tilde{x}_{i\mu}/C^{*}_{max}\left(\tilde{x}_{\mu}\right)\right] \;\left(i \in Z_{\mu}\right) \tag{10.9}$$

Thus, the solution of Problem P allows to find an optimal continuous resource allocation for a given feasible sequence. Consequently, the DCRCPSP decomposes into two interrelated subproblems: (1) to construct a precedence- and resource-feasible sequence of activities with respect to discrete resources only, i.e., a feasible sequence as defined earlier, and (2) to allocate the continuous resource optimally among activities in the feasible sequence. It should be stressed that this decomposition of the DCRCPSP is of huge importance. Firstly, notice that an optimal solution can be found by solving Problem P for all feasible sequences, and choosing the one with the minimum project duration. This full enumeration approach can be applied for small problem sizes, and guarantees finding an optimal schedule. In general, the number of all feasible sequences grows exponentially with the number of activities, and therefore, searching for an optimal feasible sequence may be performed by various search algorithms, e.g., local search metaheuristics. Secondly, the decomposition into the discrete and the continuous part allows to incorporate the knowledge on the properties of solutions to both the subproblems, and, in that way, identify cases which are easier to solve. An alternative approach could be a formulation of the DCRCPSP as a mixed integer nonlinear programming (MINLP) problem, as some problem variables are discrete and some are continuous. However, as it is known, MINLP problems are typically very difficult to solve (see, e.g., Floudas 1995), and therefore this approach has not been taken into consideration so far.

10.6 Conclusions

In this chapter the discrete-continuous resource-constrained project scheduling problem (DCRCPSP) has been considered. The problem is an extension of the classical discrete resource-constrained project scheduling problem (RCPSP), well-known from the literature. The extension consists in the presence of an additional continuous, renewable resource whose total amount available at a time is limited. As it has been discussed, continuous resources can appear very often in real life situations. The methodology for solving the DCRCPSP has been presented, based on the results obtained for the continuous resource allocation problem. The methodology critically depends on the form of the processing rate functions of activities. It has been shown that for convex functions the problem becomes easy, since sequential execution of activities, each one using the total available amount of the continuous resource, leads to optimal schedules. On the other hand, for concave processing rate functions it is desirable to perform activities in parallel. In this case the special methodology based on feasible sequences is required. In the first step a feasible sequence of activity sets is constructed, and in the second step an optimal continuous resource allocation is calculated by solving a convex mathematical programming problem. Unfortunately, in order to find an optimal schedule all feasible sequences have to be examined in general, and, in consequence, this approach becomes computationally inefficient for larger problem sizes. Moreover, in such cases, when the number of variables in the problem grows rapidly, solving the mathematical programming problem in the second step using specialized nonlinear solvers may take a long time. Therefore some heuristic approaches to allocating the continuous resource, based on procedures presented in Józefowska et al. (2002) and Waligóra (2009) for discrete-continuous machine scheduling, have already been proposed in Waligóra (2011).

In Józefowska et al. (2000) and Waligóra (2011) another heuristic approach has been proposed, where continuous resource allotments are discretized. As a result, the classical discrete multi-mode resource-constrained project scheduling problem (MRCPSP) (see Węglarz et al. 2011 and Chap. 21 of this handbook for surveys) is obtained in a version without nonrenewable resources. The number of modes for the activities depends on the discretization level, i.e., the number of discretized allotments. It should be stressed that in this approach the processing rate functions of activities need not be concave, but may be arbitrary continuous, increasing functions. The resulting MRCPSP can then be solved using one of the numerous existing methods known from the literature, e.g., efficient metaheuristics. This will generate an approximate schedule for the original DCRCPSP, however the calculations will be much less time consuming with no need for solving the nonlinear mathematical programming problem. Thus, the importance of the discretization idea follows from the fact that it allows to convert a discrete-continuous project scheduling problem into a purely discrete one. On the other hand, as mentioned earlier, it can be sometimes useful to continuize a discrete resource, in order to take advantage of the results proved for the processing rate vs. resource amount model. Discretization and continuization are opposite approaches, and the decision on using them depends on the problem knowledge and, particularly, processing rate functions.

Acknowledgements This chapter is a part of the project no. 2013/11/B/ST6/00970, funded by the Polish National Science Centre.

References

Brucker P, Drexl A, Möhring R, Neumann K, Pesch E (1999) Resource-constrained project scheduling: notation, classification, models and methods. Eur J Oper Res 112(1):3–41

Floudas CA (1995) Nonlinear and mixed-integer optimization: fundamentals and applications. Oxford University Press, Oxford

Józefowska J, Węglarz J (1998) On a methodology for discrete-continuous scheduling. Eur J Oper Res 107(2):338–353

Józefowska J, Mika M, Różycki R, Waligóra G, Węglarz J (2000) Solving the discrete-continuous project scheduling problem via its discretization. Math Method Oper Res 52(3):489–499

Józefowska J, Mika M, Różycki R, Waligóra G, Węglarz J (2002) A heuristic approach to allocating the continuous resource in discrete-continuous scheduling problems to minimize the makespan. J Sched 5(6):487–499

Różycki R, Węglarz J (2014) Power-aware scheduling of preemptable jobs on identical parallel processors to minimize makespan. Ann Oper Res 213(1):235–252

Waligóra G (2009) Tabu search for discrete-continuous scheduling problems with heuristic continuous resource allocation. Eur J Oper Res 193(3):849–856

Waligóra G (2011) Heuristic approaches to discrete-continuous project scheduling problems to minimize the makespan. Comput Optim Appl 48(2):399–421

Waligóra G (2014) Discrete-continuous project scheduling with discounted cash inflows and various payment models – a review of recent results. Ann Oper Res 213(1):319–340

Węglarz J (1976) Time-optimal control of resource allocation in a complex of operations framework. IEEE T Syst Man Cyb 6(11):783–788

Węglarz J (1980) Multiprocessor scheduling with memory allocation – a deterministic approach. IEEE T Comput 29(8):703–709

Węglarz J (1981) Project scheduling with continuously-divisible, doubly constrained resources. Manag Sci 27(9):1040–1052

Węglarz J, Józefowska J, Mika M, Waligóra G (2011) Project scheduling with finite or infinite number of activity processing modes – a survey. Eur J Oper Res 208(3):177–205

Chapter 11
Partially Renewable Resources

Ramon Alvarez-Valdes, Jose Manuel Tamarit, and Fulgencia Villa

Abstract In recent years, in the field of project scheduling the concept of partially renewable resources has been introduced. Theoretically, it is a generalization of both renewable and non-renewable resources. From an applied point of view, partially renewable resources allow us to model a large variety of situations that do not fit into classical models, but can be found in real problems in timetabling and labor scheduling. In this chapter we define this type of resource, describe an integer linear formulation and present some examples of conditions appearing in real problems which can be modeled using partially renewable resources. Then we introduce some preprocessing procedures to identify infeasible instances and to reduce the size of the feasible ones. Some exact, heuristic, and metaheuristic algorithms are also described and tested.

Keywords GRASP • Makespan minimization • Partially renewable resources • Project scheduling • Scatter search

11.1 Introduction

The classical RCPSP basically includes two types of resources: renewable resources, in which the availability of each resource is renewed at each period of the planning horizon, and non-renewable resources, whose availability are given at the beginning of the project and which are consumed throughout the processing of the activities requiring them. However, in the last decade new types of resources have been proposed to allow the model to include new types of constraints which appear in industrial problems: allocatable resources (Schwindt and Trautmann

R. Alvarez-Valdes (✉) • J.M. Tamarit
Department of Statistics and Operations Research, University of Valencia, Valencia, Spain
e-mail: ramon.alvarez@uv.es; jose.tamarit@uv.es

F. Villa
Department of Applied Statistics and Operations Research, and Quality, Polytechnic University of Valencia, Valencia, Spain
e-mail: mfuvilju@eio.upv.es

C. Schwindt, J. Zimmermann (eds.), *Handbook on Project Management and Scheduling Vol. 1*, International Handbooks on Information Systems,
DOI 10.1007/978-3-319-05443-8_11

2003, Mellentien et al. 2004), storage or cumulative resources (Neumann et al. 2002, 2005, Chap. 9 of this handbook), spatial resources (de Boer 1998), recycling resources (Shewchuk and Chang 1995), time-varying resources (Chap. 8 of this handbook) or continuous resources (Chap. 10 of this handbook).

Another new type of resource is partially renewable resources. The availability of this resource is associated to a subset of periods of the planning horizon and the activities requiring the resource only consume it if they are processed in these periods. Although these resources may seem strange at first glance, they can be a powerful tool for solving project scheduling problems. On the one hand, from a theoretical point of view, they include renewable and non-renewable resources as particular cases. In fact, a renewable resource can be considered as a partially renewable resource with an associated subset of periods consisting of exactly one period. Non-renewable resources are partially renewable resources where the associated subset is the whole planning horizon. On the other hand, partially renewable resources make it possible to model complicated labor regulations and timetabling constraints, therefore allowing us to approach many labor scheduling and timetabling problems as special cases of project scheduling problems. The RCPSP with partially renewable resources is denoted by RCPSP/π or in the three-field classification by $PSp|prec|C_{max}$.

Partially renewable resources were first introduced by Böttcher et al. (1999), who proposed an integer linear formulation and developed exact and heuristic algorithms. Schirmer (2000) studied this new type of resource thoroughly in his book on project scheduling problems. He presented many examples of special conditions which can be suitably modeled using partially renewable resources and made a theoretical study of the problem. He also proposed several families of heuristic algorithms for solving the problem. More recently, Alvarez-Valdes et al. (2006, 2008) have developed some preprocessing procedures, as well as GRASP/Path Relinking and Scatter Search algorithms, which efficiently solve the test instances published in previous studies.

In this chapter we introduce the concept of partially renewable resources and review all the existing algorithms. In Sect. 11.2 we define first this type of resource and describe an integer linear formulation, adapted from Böttcher et al. (1999). Then we present and discuss some examples of conditions appearing in real problems which can be modeled using partially renewable resources, and briefly review the first proposed algorithms. Section 11.3 contains the preprocessing procedures and Sects. 11.4 and 11.5 describe the GRASP/PR and the Scatter Search algorithms in more detail. Finally, Sect. 11.6 contains the computational results and Sect. 11.7 the conclusions.

11.2 Solving Problems with Partially Renewable Resources

The RCPSP/π can be defined as follows: Let n be the number of activities to schedule. The project consists of $n+2$ activities, numbered from 0 to $n+1$, where activities 0 and $n+1$ represent the beginning and the end of the project, respectively.

Let $Pred(i)$ be the set of immediate predecessors of activity i. Each activity i has a processing time of p_i and once started cannot be interrupted. Let $\mathscr{R}^p$ be the set of partially renewable resources. Each resource $k \in \mathscr{R}^p$ has a total availability R_k and an associated set of periods Π_k. An activity i requiring resource k will consume r_{ik} units of it at each period $t \in \Pi_k$ in which it is processed. Finally, let T be the planning horizon in which all the activities must be processed. For each activity i we obtain its earliest and latest start times, ES_i, LS_i, by critical path analysis. We denote by $W_i = \{ES_i, \ldots., LS_i\}$, the set of possible starting times, and $Q_{it} = \{t - p_i + 1, \ldots, t\}$.

11.2.1 Formulation of the Problem

The RCPSP/π consists of sequencing the activities so that the precedence and resource constraints are satisfied and the makespan is minimized.

If we define the variables:

$$x_{it} = \begin{cases} 1 & \text{if activity } i \text{ starts at time } t \\ 0 & \text{otherwise.} \end{cases}$$

the problem can be formulated as follows:

$$\text{Min.} \sum_{t \in W_{n+1}} t\, x_{n+1,t} \tag{11.1}$$

$$\text{s. t.} \sum_{t \in W_i} x_{it} = 1 \qquad (i = 0, \ldots, n+1) \tag{11.2}$$

$$\sum_{t \in W_j} (t + p_j) x_{jt} \le \sum_{t \in W_i} t\, x_{it} \qquad (i = 0, \ldots, n+1, j \in Pred(i)) \tag{11.3}$$

$$\sum_{i=1}^{n} r_{ik} \sum_{t \in \Pi_k} \sum_{q \in Q_{it} \bigcap W_i} x_{iq} \le R_k \qquad (k \in \mathscr{R}^p) \tag{11.4}$$

$$x_{it} \in \{0, 1\} \qquad (i = 0, \ldots, n+1, \;\; t \in W_i) \tag{11.5}$$

The objective function (11.1) to be minimized is the starting time of the last activity and hence the makespan of the project. According to constraints (11.2), each activity must start once. Constraints (11.3) are the precedence constraints and constraints (11.4) are the resource constraints. Note that in this problem there is only one global constraint for each resource $k \in \mathscr{R}^p$. An activity i consumes r_{ik} units of a resource k for each period in which it is processed within Π_k. Another special characteristic of this problem is that all the activities must finish within the planning horizon T in which sets Π_k are defined. The starting times of each activity i form a finite set W_i and therefore the existence of feasible solutions is not guaranteed. In fact, Schirmer (2000) has shown that the feasibility variant of the RCPSP/π is $\mathcal{NP}$-complete in the strong sense.

The above formulation is called the normalized formulation by Böttcher et al. (1999) and Schirmer (2000). Both consider an alternative, more general, formulation in which several subsets of periods are associated with each resource $k \in \mathcal{R}^p$. $\Pi_k = \{P_{k1}, \ldots, P_{k\nu}\}$ is a set of subsets of periods where each subset of periods $P_{k\mu} \in \Pi_k$ has a resource capacity $R_{k\mu}$. This general formulation can be transformed into the above formulation by a "normalization" procedure in which a new resource k' is defined for each subset of periods $P_{k\mu}$. Therefore, the normalized formulation can be used without loss of generalization.

11.2.2 Modeling Capabilities

In this section we describe several situations in which partially renewable resources can be used to model conditions which classical renewable or non-renewable cannot accommodate.

11.2.2.1 Lunch Break Assignments

In this example, taken from Böttcher et al. (1999), we have five workers in a 09:00 to 18:00 shift with a lunch break of 1 h. If the lunch period is fixed to period 13 for all workers, the availability profile would be that depicted in Fig. 11.1. We would get more flexibility if each worker could have the break either in period 13 or 14. However, if we use renewable resources, the number of workers taking the break at each period must have been decided beforehand. An example appears in Fig. 11.2, in which three workers take the break at period 13 and two at period 14.

The situation can be more appropriately modeled by using a partially renewable resource k with $\Pi_k = \{13, 14\}$ and total availability of $R_k = 5$ units. Each task consumes $r_{ik} = 1$. Therefore, the total number of working hours assigned to these periods cannot exceed five, leaving time for the five workers to have their lunch break.

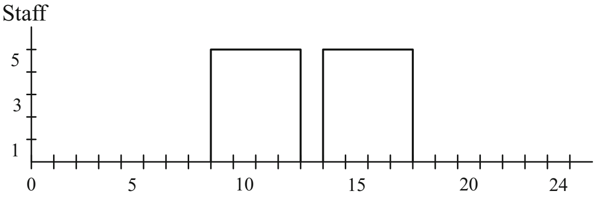


Fig. 11.1 Fixed break at period 13

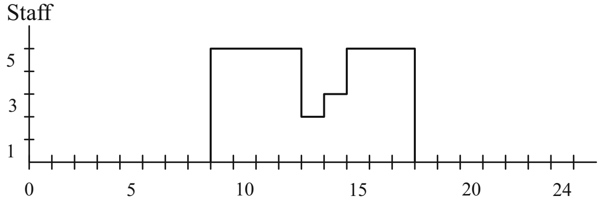


Fig. 11.2 Fixed distribution of breaks between periods 13 and 14

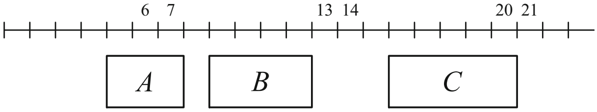


Fig. 11.3 Weekend days-off assignment

11.2.2.2 Weekend Days-Off Assignments

Let us consider a contractual condition like that of *working at most two weekend days out of every three consecutive weeks* (Alvarez-Valdes et al. 2008). This condition cannot be modeled as a renewable resource because this type of resource considers each period separately. It cannot be modeled as a non-renewable resource because this type of resource considers the whole planning horizon. We model this condition for each worker as a partially renewable resource k with a set of periods $\Pi_k = \{6, 7, 13, 14, 20, 21\}$ for the first three weekends and a total availability of $R_k = 2$ units. Each task i assigned to this worker consumes $r_{ik} = 1$ unit of this resource for each weekend day in which it is processed. In Fig. 11.3 we see three activities A, B, and C scheduled within the timescale depicted above. Activity A is in process at periods 6 and 7 and then it consumes two units of the resource. Activity B does not consume the resource and activity C consumes one unit in period 20. If these three activities had to be done by the same worker, the solution in the figure would not be possible because it would exceed the resource availability.

11.2.2.3 Special Conditions in School Timetabling Problems

When building their timetables, some schools include conditions such as that each teacher must have classes at least one afternoon per week and at most two afternoons per week. Let us assume for simplicity that the morning sessions are numbered 1, 3, 5, 7, 9 and the afternoon sessions are numbered 2, 4, 6, 8, 10. For each teacher we define two partially renewable resources. In both cases, $\Pi_1 = \Pi_2 = \{2, 4, 6, 8, 10\}$. In order to ensure that the teacher attends at least one afternoon session, $R_1 = -1$ and for each of the teacher's classes, $r_{i1} = -1$. The second condition is ensured defining $R_2 = 2$, $r_{i2} = 1$.

A complete study of logical conditions that can be modeled by using partially renewable resources and then applied to different real situations can be found in Schirmer (2000).

11.2.3 A Branch & Bound Algorithm

Böttcher et al. (1999) proposed an exact algorithm for the RCPSP/π that is based upon the branch and bound method introduced by Talbot and Patterson (1978) for the classical RCPSP, because this was one of the few approaches allowing time-varying capacity profiles to be accommodated. The basic scheme of the algorithm is straightforward. Starting with an empty schedule, partial schedules are feasibly augmented by activities, one at a time. The enumeration tree is built with a depth-first strategy, where the depth of a branch is the number of activities scheduled. Backtracking occurs when an activity cannot be scheduled within its feasible interval. In order to reduce the computational effort, Böttcher et al. (1999) developed several resource-based feasibility bounds tailored to the specifics of the problem.

The authors employed a sample of 1,000 test instances generated from a modification of ProGen (Kolisch et al. 1995). They generated four sets with 250 instances each one and with 15, 20, 30, and 60 activities, respectively. The number of resources for each set was 30. Applying the branch and bound algorithm in a truncated version (TBB), using a CPU time limit of 5 min per instance on a C-language implementation on an IBM RS/6000 workstation at 66 MHz, produced the results shown in Table 11.1.

The study by Böttcher et al. (1999) also included several priority rules designed to obtain good quality solutions in short computing times. First, they tested some classical rules: MINEFT (minimum earliest finishing time), MINLFT (minimum latest finishing time), MINSLK (minimum slack), MTSUCC (most total successors). Defining the maximum resource usage of each activity i as $SRU_i = \sum_{k \in \mathcal{R}^p} r_{ik}$, they included the priority rules MAXSRU and MINSRU, considering the maximum and minimum of the activities resource consumptions. In order to obtain a more accurate calculation of resource usage they defined SC_{ikt}, the consumption of resource k by activity i when started at period t, $MC_{ikt} = \min\{SC_{ik\tau} | t \leq \tau \leq LS_i\}$, the minimum resource consumption of resource k by activity i when started not earlier than t and not later than LS_i, and MCI_{ikt}

Table 11.1 Effectiveness of TBB algorithm proposed by Böttcher et al. (1999)

Activities	Solved to optimality	Solved to feasibility	Unsolved
15	240	10	–
20	229	21	–
30	217	14	19
60	229	11	10

which is the sum of MC_{ikt} of all the successors of activity i considering their corresponding time intervals. These elements were used to calculate $TRU_{it} = \sum_{k \in \mathscr{R}^p} (SC_{ikt} + MCI_{ikt})$ and $RRU_{it} = \sum_{k \in \mathscr{R}^p : R_k > 0} (SC_{ik} + MCI_{ikt})/R_k$ which are lower bounds on the absolute and relative resource usage, producing rules MAXTRU, MINTRU, MAXRRU, MINRRU. These rules are static because they do no take into account the resources already consumed in a partial solution. If R_k^0 if the left-over capacity of resource k with respect to a partial schedule, $DRRU_{jt} = \sum_{k \in \mathscr{R}^p : R_k^0 > 0} (SC_{ikt} + MCI_{ikt})/R_k^0$ is the dynamic relative resource usage and $TRC_{jt} = \sum_{k \in \mathscr{R}^p} (R_k^0 - SC_{ikt} + MCI_{ikt})/R_k^0$ an upper bound of the total remaining capacity. These values were used to define rules MAXDRRU, MINDRRU, MAXTRC, MINTRC. Böttcher et al. (1999) concluded that classical rules worked well for easy instances, while the new dynamic rules did a better job on hard instances. In summary, rule MINLFT seemed to be the most promising one.

11.2.4 Schirmer's Algorithms

In his book in 2000, Schirmer made a complete study of partially renewable resources, their modeling capabilities and theoretical properties. He also developed a series of progressively more complex algorithms. First, he collected known priority rules and added some new ones. The list appears in Table 11.2. In order to test them, Schirmer used Progen to generate new instances, but previously he analyzed the parameter combinations that generally produce infeasible solutions. So he selected the most promising parameter combinations to generate the new set of test instances. For each of the 96 promising clusters, ten instances were generated of sizes 10, 20, 30, and 40 with 30 resources.

In Table 11.2, RK_{ks} is the remaining availability or resource k at stage s; RD_{ikt} is the relevant demand (consumption) of resource k if activity i starts at time t; MDE_{ikt} is the minimum relevant demand on resource k of all the successors of activity i when it starts at time t; and D_s is the set of possible assignments (i, t) at stage s. The 12 rules involving resources can use all the resources or only those which are potentially scarce. Therefore, a total of 32 rules were defined.

In a second phase he introduced randomization, using the Modified Regret-Based Biased Random Sampling, in which the probability assigned to each activity is calculated using the regret, that is the worst possible consequences that might result from selecting another candidate. In a third phase, Schirmer developed a local search, based on local and global left-shifts of the activities. Finally, he proposed a Tabu Search algorithm. He compared the results with those obtained with the *TBB* algorithm by Böttcher et al. (1999) and concluded that good construction methods solved significantly more instances optimally than tabu search, so these perform better on easier instances. Yet, on very hard instances, tabu search outperformed all construction methods. Indeed, the multi-start version of the tabu search method was the only one that managed to solve all the instances attempted, except for five that he conjectured to be infeasible.

Table 11.2 Priority rules proposed by Schirmer (2000)

	Measure	Definition	Static vs. dynamic	Local vs. global
MIN	EF_i	$ES_i + p_i$	S	L
MIN	ES_i	ES_i	S	G
MIN	LF_i	$LS_i + p_i$	S	L
MIN	LS_i	LS_i	S	G
MAX	MTS_i	$\lvert\{i' \mid i < i'\}\rvert$	S	L
MIN	SLK_i	$LS_i - EF_i$	D	L
MIN	SPT_i	p_i	S	L
MIN	$SSLK_i$	$LS_i - ES_i$	S	L
MAX	DRC_{it}	$\sum_k (RK_{ks} - RD_{ikt})$	D	L
MAX	DRC/E_{it}	$\sum_k (RK_{ks} - RD_{ikt} - MDE_{ikt})$	D	G
MIN	DRD_{it}	$\sum_k RD_{ikt}/max\{RD_{i'kt'} \mid (i', t') \in D_s$	D	G
MIN	DRD/E_{it}	$\sum_k (RD_{ikt} + MDE_{ikt})/max\{RD_{i'kt'}$ $+MDE_{i'kt'} \mid (i', t') \in D_s)$	D	G
MIN	DRS_{it}	$\sum_k RD_{ikt}/RK_{ks}$	D	L
MIN	DRS/E_{it}	$\sum_k (RD_{ikt} + MDE_{ikt})/RK_{ks}$	D	G
MIN	RRD_{it}	$\sum_k RD_{ikt}/(r_{ik} p_i)$	S	L
MIN	RRD/E_{it}	$\sum_k RD_{ikt}/(r_{ik} p_i) + \sum_{i<j} MDE_{jkEST_j}/(r_{jk} p_k)$	D	G
MIN	TRD_{it}	$\sum_k RD_{ikt}$	S	L
MIN	TRD/E_{it}	$\sum_k (RD_{ikt} + MDE_{ikt})$	S	G
MIN	TRS_{it}	$\sum_k RD_{ikt}/R_k$	S	L
MIN	TRS/E_{it}	$\sum_k (RD_{ikt} + MDE_{ikt})/R_k$	S	G

11.3 Preprocessing

Preprocessing has two objectives. First, helping to decide whether a given instance is infeasible or if it has feasible solutions. If the latter is the case, a second objective is to reduce the number of possible starting times of the activities and the number of resources. If these two objectives are satisfactorily achieved, the solution procedures will not waste time trying to solve infeasible problems and will concentrate their efforts on the relevant elements of the problem.

The preprocessing we have developed includes several procedures:

1. *Identifying trivial problems*
   If the solution in which the starting time of each activity i is set to ES_i is resource-feasible, then it is optimal.
2. *Reducing the planning horizon T*
   For each instance, we are given a planning horizon $(0, T)$. This value plays an important role in the time-indexed problem formulation. In fact, late starting times of the activities, LS_i are calculated starting from T in a backward recursion. Therefore, the lower the value T, the fewer variables the problem will have. In

order to reduce T, we try to build a feasible solution for the given instance, using a GRASP algorithm, which will be described later. The GRASP iterative process stops as soon as a feasible solution is obtained or after 200 iterations. The new value T is updated to the makespan of the feasible solution obtained. Otherwise, T is unchanged.

If the makespan of the solution equals the length of the critical path in the precedence graph, the solution is optimal and the process stops and returns the solution.

3. *Eliminating idle resources*

   Each resource $k \in \mathscr{R}^p$ is consumed only if the activities requiring it are processed in periods $t \in \Pi_k$. Each activity can only be processed in a finite interval. It is therefore possible that no activity requiring the resource can be processed in any period of Π_k. In this case, the resource is idle and can be eliminated. More precisely, if we denote the points in time where activity i can be in progress by $\overline{W_i} = \{ES_i, \ldots, LS_i + p_i - 1\}$, and $\forall i \mid r_{ik} > 0 : \ \Pi_k \bigcap \overline{W_i} = \emptyset$, the resource $k \in \mathscr{R}^p$ is idle and can be eliminated.

4. *Eliminating non-scarce resources*

   Schirmer (2000) distinguishes between scarce and non-scarce resources. He considers a resource $k \in \mathscr{R}^p$ as scarce if $\sum_{i=1}^{n} r_{ik} p_i > R_k$, that is, if an upper bound on the maximum resource consumption exceeds the resource availability. In this case, the upper bound is computed by supposing that all the activities requiring the resource are processed completely inside Π_k.

   We have refined this idea by taking into account the precedence constraints. Specifically, we calculate an upper bound on the maximum consumption of resource k by solving the following linear problem:

$$\text{Max.} \sum_{i=1}^{n} r_{ik} \sum_{t \in \Pi_k} \sum_{q \in Q_{it} \bigcap W_i} x_{iq} \tag{11.6}$$

$$\text{s. t.} \quad \sum_{t \in W_i} x_{it} = 1 \qquad (i = 1, \ldots, n) \tag{11.7}$$

$$\sum_{\tau=t}^{T} x_{j\tau} + \sum_{\tau=1}^{t+p_j-1} x_{i\tau} \leq 1 \qquad (i = 1, \ldots, n; j \in Pred(i); t \leq T) \tag{11.8}$$

$$x_{it} \geq 0 \qquad (i = 1, \ldots, n; t \in W_i) \tag{11.9}$$

The objective function (11.6) maximizes the resource consumption over the whole project. Constraints (11.7) ensure that each activity starts once. Constraints (11.8) are the precedence constraints. We use this expression, introduced by Christofides et al. (1987), because it is more efficient than the usual precedence constraint. In fact, the reformulation of these constraints produces a linear problem whose coefficient matrix is totally unimodular and thus all the vertices of the feasible region are integer-valued (Chaudhuri et al. 1994). If the solution value is not greater than the resource availability, this resource will not cause any conflict and can be skipped in the solution process.

5. *A filter for variables based on resources*

   For each activity i and each possible starting time $t \in W_i$, we compute a lower bound LB_{kit} on the consumption of each resource k if activity i starts at time t. We first include the resource consumption of activity i when starting at that time t and then for each other activity in the project we consider all its possible starting times, determine for which of them the resource consumption is minimum and add that amount to LB_{kit}. Note that this minimum is calculated over all the periods in W_j for each activity j not linked with i by precedence constraints, but for an activity h which is a predecessor or successor of i, the set W_h is reduced by taking into account that i is starting at time t. If for some resource k, $LB_{kit} > R_k$, time t is not feasible for activity i to start in, and the corresponding variable x_{it} is set to 0.

   When this filter is applied to an activity i, some of its possible starting times can be eliminated. From then on, the set of possible starting times is no longer W_i. We denote by $\underline{W_i}$ the set of starting times passing the filter.

   This filter is applied iteratively. After a first run on every activity and every starting time, if some of the variables are eliminated the process starts again, but this time computing LB_{kit} on the reduced sets. As the minimum resource consumptions are calculated over restricted subsets, it is possible that new starting times will fail the test and can be eliminated. The process is repeated until no starting time is eliminated in a complete run.

6. *Consistency test for finishing times*

   When the above filter eliminates a starting time of an activity i, it is possible that some of the starting times of its predecessors and successors are no longer feasible. For an activity i, $\tau_i = \max\{t \mid t \in \underline{W_i}\}$. Then, for each $j \in Pred(i)$ the starting times $t \in \underline{W_j}$ such that $t > \tau_i - p_j$ can be eliminated. Analogously, $\gamma_i = \min\{t \mid t \in \underline{W_i}\}$. Then, for each $j \in Succ(i)$, the starting times $t \in \underline{W_j}$ such that $t < \gamma_i + p_i$ can also be eliminated.

   This test is also applied iteratively until no more starting times are eliminated. If, after applying these two procedures for reducing variables, an activity i has $\underline{W_i} = \emptyset$, the problem is infeasible. If the makespan of the initial solution built by GRASP equals the minimum starting time of the last activity $n + 1$, then this solution is optimal.

7. *A linear programming bound for the starting time of activity* $n + 1$

   We solve the linear relaxation of the integer formulation of the problem given in Sect. 11.2.1, using only variables and resources not eliminated in previous preprocessing procedures and replacing expression (11.3) with expression (11.8) for the precedence constraints. The optimal value of the linear program, opl_1, is a lower bound for the optimal value of the starting time of activity $n + 1$. Therefore we can set $x_{n+1,t} := 0$ for all $t < \lceil opl_1 \rceil$. If that eliminates some variables with non-zero fractional values, we solve the modified problem and obtain a new solution opl_2, strictly greater than opl_1, which can in turn be used to eliminate new variables, and the process goes on until no more variables are removed. If the solution value obtained by GRASP equals the updated minimum starting time for activity $n + 1$, this solution is optimal. Otherwise, the lower bound can be used to check the optimality of improved solutions obtained in the GRASP process.

11.4 GRASP Algorithm

In this section we describe the elements of our GRASP implementation in detail. The first two subsections contain the constructive randomized phase and the improvement phase. The last two subsections describe an enhanced GRASP procedure and a Path Relinking algorithm operating over the best GRASP solutions obtained. A comprehensive review of GRASP can be found in Resende and Ribeiro (2003).

11.4.1 The Constructive Phase

11.4.1.1 A Deterministic Constructive Algorithm

We have adapted the serial schedule-generation scheme (SSS) proposed by Schirmer (2000), which in turn is an adaptation of the serial schedule-generation scheme commonly used for the classical RCPSP. We denote by S_i the starting time assigned to activity i. At each stage of the iterative procedure an activity is scheduled by choosing from among the current set of decisions, defined as the pairs (i, t) of an activity i and a possible starting time $t \in \underline{W_i}$. The selection is based on a priority rule.

Step 0. Initialization

$S_0 := 0$ (sequencing dummy activity 0)

$\mathscr{C} := \{0\}$ (activities already scheduled)

$\forall k \in \mathscr{R}^p \; : \; RK_k := R_k$ (remaining capacity of resource k)

$$TD_k := \sum_{i=1}^{n} r_{ik} p_i \text{ (maximum possible demand for } k \text{)}$$

$SR := \{k \in \mathscr{R}^p \mid TD_k > RK_k\}$ (set of possible scarce resources)

$EL :=$ set of eligible activities, i.e. those activities for which activity 0 is the only predecessor

Step 1. Constructing the set of decisions

$\mathscr{D} := \{(i, t) \mid i \in EL, \; t \in \underline{W_i}\}$

Step 2. Choosing the decision

Select the best decision (i^*, t^*) in $\mathscr{D}$, according to a priority rule

```
Step 3.   Feasibility test
    if $(i^*, t^*)$ is resource-feasible, go to Step 4.
    else
        $\mathscr{D} := \mathscr{D} \setminus \{(i^*, t^*)\}$
        if $\mathscr{D} = \emptyset$, STOP. The algorithm does not find a feasible solution.
        else, go to Step 2.
Step 4.   Update
    $S_{i^*} := t^*$
    $\mathscr{C} := \mathscr{C} \cup \{i^*\}$
    $EL := (EL \setminus \{i^*\}) \cup \{i \,|\, Pred(i) \subseteq S\}$
    $\forall h \,|\, i \in Pred(h)\ :\ \underline{W_h} = \underline{W_h} \setminus \{\tau \mid t^* + p_{i^*} > \tau\}$
    $\forall k \in \mathscr{R}^p\ :$   $RK_k := RK_k - RD_{ikt}$
                  $TD_k := TD_k - r_{i^*k}\, p_{i^*}$
                  if $TD_k \leq RK_k$ , then $SR = SR \setminus \{k\}$
    if $|S| = n + 2$, STOP. The sequence is completed.
    else, go to Step 1.
```

At Steps 1 and 2, we follow Schirmer's design, working with the set of decisions $\mathscr{D}$ and choosing from it both the activity to sequence and the time at which it starts. An alternative could have been to first select the activity and then the time, choosing, for instance, the earliest resource-feasible time. However, this strategy has serious drawbacks. First, it is less flexible. Priority rules which take into account the resource consumption could not be used, because this consumption varies depending on the periods in which the activity is processed. Second, as Schirmer has shown in his book, scheduling each activity at its earliest resource-feasible time may not only fail to produce an optimal solution but may also fail to produce feasible solutions. As he says: "delaying activities from their earliest feasible start time is crucial to finding good—even feasible—solutions for some instances".

At Step 3, we perform a complex feasibility test which does not involve only activity i^* but also the unscheduled activities. Following a process which is quite similar to the filter described in Sect. 11.3, we try to assess the effect of scheduling i^* at time t^* by computing an estimation of the global resource consumptions, also considering the minimum consumption of the activities not yet scheduled. If this estimation exceeds the remaining resource availability, t^* is labeled as non-feasible for i^*. This test allows us to avoid decisions which will inevitably produce infeasible solutions in the later stages of the constructive process and our computational experience has told us that significantly improves the proportion of feasible solutions we obtain. However, it has a larger computational cost. Therefore, we do not perform the feasibility test for each decision of $\mathscr{D}$ at Step 1, as in Schirmer's algorithm, but only for the decision chosen at Step 2. In problems with a large number of possible finishing times for the activities, this strategy is more efficient. If this decision fails the feasibility test of Step 3, the second best decision is tested and so on. Few tests are usually required and therefore it is more convenient to take the feasibility test out of Step 1.

The resource availabilities are updated, subtracting RD_{ikt}, the consumption of activity i when starting at period t from the remaining capacity R_k. We also keep the set of possible scarce resources SR updated because some priority rules based on resource consumption only take this type of resource into account.

11.4.1.2 Priority Rules

We have tested the 32 priority rules used by Schirmer (2000). The first eight are based on the network structure, including classical rules such as EFT, LFT, SPT or MINSLK. The other 24 rules are based on resource utilization. Twelve of them use all the resources and the other 12 only the scarce resources. A preliminary computational experience allowed us to choose the most promising rules and use them in the next phases of the algorithm's development. These preliminary results also showed that even with the best performing rules, the deterministic constructive algorithm failed to obtain a feasible solution for many of the ten-activity instances generated by Böttcher et al. (1999). Therefore, the objective of the randomization procedures included in the algorithm was not only to produce diverse solutions but also to ensure that for most of the problems the algorithm would obtain a feasible solution.

11.4.1.3 Randomization Strategies

We introduce randomization procedures for selecting the decision at Step 2 of the constructive algorithm. Let sc_{it} be the score of decision (i,t) on the priority rule we are using, let $sc_{max} = \max\{sc_{it} | (i,t) \in \mathscr{D}\}$, and let δ be a parameter to be determined ($0 < \delta < 1$). We have considered three alternatives:

1. *Random selection on the Restricted Candidate List, RCL*
   Select decision (i^*, t^*) at random in set $RCL = \{(i,t) \mid sc_{it} \geq \delta sc_{max}\}$
2. *Biased selection on the Restricted Candidate List, RCL*
   We build the Restricted Candidate List as in alternative 1, but instead of choosing at random from among its elements, the decisions involving the same activity i are given a weight which is inversely proportional to the order of their finishing times. For instance, if in RCL we have decisions $(2,4), (2,5), (2,7), (2,8)$ involving activity 2 and ordered by increasing finishing times, then decision $(2,4)$ will have a weight of 1, decision $(2,5)$ weight $1/2$, decision $(2,7)$ weight $1/3$ and decision $(2,8)$ weight $1/4$. The same procedure is applied to the decisions corresponding to the other activities. Therefore, the decisions in RCL corresponding to the lowest starting times of the activities involved will be equally likely and the randomized selection process will favor them.
3. *Biased selection on the set of decisions $\mathscr{D}$*
   We have also implemented the Modified Regret-Based Biased Random Sampling (MRBRS/δ) proposed by Schirmer (2000), in which the decision (i,t) is chosen

from among the whole set $\mathscr{D}$ but with its probability proportional to its regret value. The regret value is a measure of the worst possible consequence that might result from selecting another decision.

11.4.1.4 A Repairing Mechanism

The randomization strategies described above significantly improve the ability of the constructive algorithm to find feasible solutions for tightly constrained instances. However, a limited computational experience showed that not even with the best priority rule and the best randomization procedure could the constructive algorithm obtain feasible solutions for all of the ten-activity instances generated by Böttcher et al. (1999). Therefore, we felt that the algorithm was not well-prepared for solving larger problems and we decided to include a repairing mechanism for infeasible partial schedules.

In the construction process, if at Step 3 all the decisions in $\mathscr{D}$ fail the feasibility test and $\mathscr{D}$ finally becomes empty, instead of stopping the process and starting a new iteration, we try to re-assign some of the already sequenced activities to other finishing times in order to free some resources that could be used for the first of the unscheduled activities to be processed. If this procedure succeeds, the constructive process continues. Otherwise, it stops.

11.4.2 The Improvement Phase

Given a feasible solution obtained in the constructive phase, the improvement phase basically consists of two steps. First, identifying the activities whose starting times must be reduced in order to have a new solution with the shortest makespan. These activities are labeled as *critical*. Second, moving critical activities to lower finishing times in such a way that the resulting sequence is feasible according to precedence and resource constraints. We have designed two types of moves: a simple move, involving only the critical activity, and a double move in which, apart from the critical activity, other activities are also moved.

11.4.3 An Aggressive Procedure

The standard version of our heuristic algorithm starts by applying the preprocessing procedure in Sect. 11.3. The reduced problem then goes through the iterative GRASP algorithm described above, combining a constructive phase and an improvement phase at each iteration, until the stopping criterion, here a fixed number of iterations, is met.

An enhanced version of the heuristic algorithm combines preprocessing and GRASP procedures in a more *aggressive* way. After a given number of iterations (stopping criterion), we check whether the best known solution has been improved. If this is the case, we run the preprocessing procedures again, setting the planning horizon T to the makespan of the best-known solution and running the filters for variable reduction. The GRASP algorithm is then applied to the reduced problem. Obtaining feasible solutions is now harder, but if the procedure succeeds we will get high quality solutions.

The stopping criterion combines two aspects: the number of iterations since the last improvement and the number of iterations since the last call to preprocessing. For the first aspect a low limit of 50 iterations is set. We do not call preprocessing every time the solution is improved, but we do not wait too long to take advantage of the improvements. For the second aspect we set a higher limit of 500 iterations to give the process enough possibilities of improving the best current solution and, at the same time, do not allow it to run for too long a time.

11.4.4 Path Relinking

If throughout the iterative procedures described above we keep a set of the best solutions, usually denoted as *elite solutions*, we can perform a Path Relinking procedure. Starting from one of these elite solutions called the *initiating solution*, we build a path towards another elite solution called the *guiding solution*. We progressively impose the attributes of the guiding solution onto the intermediate solutions in the path, so these intermediate solutions evolve from the initiating solution until they reach the guiding solution. Hopefully, along these paths we will find solutions which are better than both the initiating and the guiding solutions.

We keep the ten best solutions obtained in the GRASP procedure. We consider each of them in turn as the initiating solution and another as the guiding solution. We build a path from the initiating to the final solution with $n-1$ intermediate solutions. The jth solution will have the finishing times of the first j activities taken from the guiding solution, while the remaining $n-j$ finishing times will still correspond to those of the initiating solution. Therefore, along the path, the intermediate solutions will become progressively more similar to the guiding solution and more different from the initiating one. In many cases these intermediate solutions will not be feasible. If this is the case, a repairing mechanism similar to that described in Sect. 11.4 is applied. We proceed from activity 1 to activity n, checking for each activity j whether the partial solution from 1 to j is feasible. If it is not, we first try to find a feasible finishing time for activity j, keeping previous activities unchanged. If that is not possible, we try to re-assign some of the previous activities to other finishing times in order to obtain some resources which are necessary for processing activity j at one of its possible finishing times. If this procedure succeeds, we consider activity $j+1$. Otherwise, the solution is discarded and we proceed to the next intermediate solution. If we obtain a complete intermediate solution which is feasible, we apply to it the improvement phase described in the GRASP algorithm.

11.5 Scatter Search Algorithm

A Scatter Search algorithm is an approximate procedure in which an initial population of feasible solutions is built and then the elements of specific subsets of that population are systematically combined to produce new feasible solutions which will hopefully improve the best known solution (see the book by Laguna and Marti (2004) for a comprehensive description of the algorithm). The basic algorithmic scheme is composed of five steps:

1. Generation and improvement of solutions
2. Construction of the Reference Set
3. Subset selection
4. Combination procedure
5. Update of the Reference Set

This basic algorithm stops when the Reference Set cannot be updated and then no new solutions are available for the combination procedure. However, the scheme can be enhanced by adding a new step in which the Reference Set is regenerated and new combinations are possible. The following enumeration describes each step of the algorithm in detail.

1. *Generation and improvement of solutions*
   The initial population is generated by using the basic version of the GRASP algorithm.
2. *Generation of the Reference Set*
   The Reference Set, *RefSet*, the set of solutions which will be combined to obtain new solutions, is built by choosing a fixed number b of solutions from the initial population. Following the usual strategy, b_1 of them are selected according to a quality criterion: the b_1 solutions with the shortest makespan, with ties randomly broken. The remaining $b_2 = b - b_1$ solutions are selected according to a diversity criterion: the solutions are selected one at a time, each one of them the most diverse from the solutions currently in *RefSet*. That is, select solution S' for which the $\text{Min}_{S \in RefSet}\{dist(S, S')\}$ is maximum. The distance between two solutions S^1 and S^2 is defined as

$$dist(S^1, S^2) = \sum_{i=1}^{n+1} |S_i^1 - S_i^2|$$

   where S_i^μ is the starting time of the i-th activity in solution S^μ.
3. *Subset selection*
   Several combination procedures were developed and tested. Most of them combine two solutions, but one of them combines three solutions. The first time the combination procedure is called, all pairs (or trios) of solutions are considered and combined. In the subsequent calls to the combination procedure, when the

Reference Set has been updated and is composed of new and old solutions, only combinations containing at least one new solution are studied.

4. *Combining solutions*

   Eight different combination procedures have been developed. Each solution S^{μ} is represented by the vector of the starting times of the activities of the project: $S^{\mu} = (S_0^{\mu}, S_1^{\mu}, S^{\mu_2}, \ldots, S_n^{\mu}, S_{n+1}^{\mu})$. When combining two solutions S^1 and S^2 (or three solutions S^1, S^2 and S^3), the solutions will be ordered by nondecreasing makespan. Therefore, S^1 will be a solution with a makespan lower than or equal to the makespan of S^2 (and the makespan of S^2 will be lower than or equal to the makespan of S^3).

   Combination 1

   The starting times of each activity in the new solution, S^{new}, will be a weighted average of the corresponding starting times in the two original solutions:

   $$S_i^{new} = \lfloor \frac{k_1 S_i^1 + k_2 S_i^2}{k_1 + k_2} \rfloor \quad \text{where } k_1 = (1/S_{n+1}^1)^2 \text{ and } k_2 = (1/S_{n+1}^2)^2$$

   Combination 2

   A crosspoint $m \in \{1, 2, .., n\}$ is taken randomly. The new solution, S^{new}, takes the first m starting times from S^1. The remaining starting times $m+1, m+2, .., n+1$ are selected at random from S^1 or S^2.

   Combination 3

   A crosspoint $m \in \{1, 2, .., n\}$ is taken randomly. The new solution, S^{new}, takes the first m starting times from S^1. The remaining starting times $m+1, m+2, .., n+1$ are selected from S^1 or S^2 with probabilities, π_1 and π_2, inversely proportional to the square of their makespan:

   $$\pi_1 = \frac{(1/S_{n+1}^1)^2}{(1/S_{n+1}^1)^2 + (1/S_{n+1}^2)^2} \quad \text{and} \quad \pi_2 = \frac{(1/S_{n+1}^2)^2}{(1/S_{n+1}^1)^2 + (1/S_{n+1}^2)^2}$$

   Combination 4

   The combination procedure 2 with $m = 1$. Only the first starting time is guaranteed to be taken from S^1.

   Combination 5

   The combination procedure 3 with $m = 1$. Only the first starting time is guaranteed to be taken from S^1.

   Combination 6

   Two crosspoints $m_1, m_2 \in \{1, 2, .., n\}$ are taken randomly. The new solution, S^{new}, takes the first m_1 starting times from S^1. The starting times $m_1 + 1, m_1 + 2, .., m_2$ are taken from S^2 and the starting times $m_2 + 1, m_2 + 2, .., n + 1$ are taken from S^1.

Combination 7
The starting times of S^1 and S^2 are taken alternatively to be included in S^{new}. Starting from the last activity $n+1$, $S^{new}_{n+1} = S^1_{n+1}$, then $S^{new}_n = S^2_n$ and so on until completing the combined solution.
Combination 8
Three solutions S^1, S^2 and S^3 are combined by using a voting procedure. When deciding the value of S^{new}_i, the three solutions vote for their own starting time S^1_i, S^2_i, S^3_i. The value with a majority of votes is taken as S^{new}_i. If the three values are different, there is a tie. In that case, if the makespan of S^1 is strictly lower than the others, the vote of quality of S^1 imposes its finishing time. Otherwise, if two or three solutions have the same minimum makespan, the starting time is chosen at random from among those of the tied solutions.

$$S^{new}_i = \begin{cases} S^1_i & \text{if } S^1_i = S^2_i \\ S^2_i & \text{if } S^1_i \neq S^2_i = S^3_i \\ S^1_i & \text{if } S^1_i \neq S^2_i \neq S^3_i \text{ and } S^1_{n+1} < S^2_{n+1} \\ random\{S^1_i, S^2_i\} & \text{if } S^1_i \neq S^2_i \neq S^3_i \text{ and } S^1_{n+1} = S^2_{n+1} < S^3_{n+1} \\ random\{S^1_i, S^2_i, S^3_i\} & \text{if } S^1_i \neq S^2_i \neq S^3_i \text{ and } S^1_{n+1} = S^2_{n+1} = S^3_{n+1} \end{cases}$$

Most of the solutions obtained by the combination procedures do not satisfy all the resource and precedence constraints. The non-feasible solutions go through a repairing process that tries to produce feasible solutions which are as close as possible to the non-feasible combined solution. This process is composed of two phases. First, the starting times S^{new}_i are considered in topological order to check if the partial solution $(S^{new}_0, S^{new}_1, .., S^{new}_i)$ satisfies precedence and resource constraints. If that is the case, the next time S^{new}_{i+1} is studied. Otherwise, S^{new}_i is discarded as the starting time of activity i and a new time is searched for from among those possible starting times of i. The search goes from times close to S^{new}_i to times far away from it. As soon as a time t^i is found which could be included in a feasible partial solution $(S^{new}_0, S^{new}_1, .., t^i)$, the search stops and the next time S^{new}_{i+1} is considered. If no feasible time is found for activity i, the process goes to the second phase which consists of a repairing procedure similar to that of the constructive algorithm. This procedure tries to change the starting times of previous activities, $1, 2, .., i-1$, in order to give activity i more chances of finding a starting time satisfying precedence and resource constraints. If this repairing mechanism succeeds, the process goes back to the first phase and the next time S^{new}_{i+1} is considered. Otherwise, the combined solution is discarded.

5. *Updating the Reference Set*
The combined solutions which were initially feasible and the feasible solutions obtained by the repairing process described above go through the improvement phase in Sect. 11.4.2. The improved solutions are then considered for inclusion in the Reference Set. The Reference Set *RefSet* is updated according to the quality

criterion: the best b solutions from among those currently in *RefSet* and from those coming from the improvement phase will form the updated set *RefSet*.
If the set *RefSet* is not updated because none of the new solutions qualify, then the algorithm stops, unless the regeneration of *RefSet* is included in the algorithmic scheme.

6. *Regenerating the Reference Set*
The regeneration of Reference Set *RefSet* has two objectives: on the one hand, introducing diversity into the set, because the way in which *RefSet* is updated may cause diverse solutions with a high makespan to be quickly substituted with new solutions with a lower makespan but more similar to solutions already in *RefSet*; on the other hand, obtaining high quality solutions, even better than those currently in *RefSet*.
The new solutions are obtained by again applying the GRASP algorithm with a modification. We take advantage of the information about the optimal solution obtained up to that point in order to focus the search on high quality solutions. More precisely, if the best known solution has a makespan S_{n+1}^{best}, we set the planning horizon $T = S_{n+1}^{best}$ and run the preprocessing procedures again, reducing the possible starting times of the activities. When we run the GRASP algorithm, obtaining solutions is harder, because only solutions with a makespan lower than or equal to S_{n+1}^{best} are allowed, but if the algorithm succeeds we will get high quality solutions.
For the regenerated set *RefSet* we then consider three sources of solutions: the solutions obtained by the GRASP algorithm, the solutions currently in *RefSet* and the solutions in the initial population. From these solutions, the new set *RefSet* is formed. Typically, the b_1 quality solutions will come from the solutions obtained by the GRASP, completed if necessary by the best solutions already in *RefSet*, while the b_2 diverse solutions will come from the initial population.

11.6 Computational Results

This section describes the test instances used and summarizes the results obtained on them by the preprocessing procedures, the priority rules, and the metaheuristic algorithms developed in previous sections.

11.6.1 Test Instances

Böttcher et al. (1999) generated an instance generator PROGEN 2. Taking as their starting point PROGEN (Kolisch et al. 1995), an instance generator for the classical RCPSP with renewable resources, they modified and enlarged the set of parameters and generated a set of 2,160 instances with ten non-dummy activities, ten replications for each one of the 216 combinations of parameter values. As most of the problems were infeasible, they restricted the parameter values to the 25 most promising combinations and generated 250 instances of sizes 15, 20, 30 and 60 of non-dummy activities, always keeping the number of resources to 30.

Later Schirmer (2000) developed PROGEN 3, an extension of PROGEN 2, and generated some new test instances. He generated 960 instances of sizes 10, 20, 30, and 40, with 30 resources. Most of them have a feasible solution, while a few of those with ten activities are infeasible. Additionally, we generated a new set of test instances of 60 activities using Schirmer's PROGEN 3, with the same parameter values he used to generate his problems. This new set gave us an indication of the performance and running times of our algorithms on larger problems.

We applied all the algorithms and procedures described in previous sections to both sets of instances, obtaining similar results. In the next subsections we present and comment on the results obtained on Schirmer's problems.

11.6.2 Preprocessing Results

Table 11.3 shows the detailed results of the preprocessing process. A first filter identifies the trivial instances. For the remaining instances, a first feasible solution is obtained. Let us suppose that the value of this solution is S_{n+1}^{start}. If S_{n+1}^{start} equals the length of the critical path, the solution is optimal. The number of instances proven optimal by this test appears as *CPM_bound*. Otherwise, the filters for resources and variables are used, sometimes eliminating some feasible finishing times for the last activity $n + 1$. If S_{n+1}^{start} equals the minimum possible finishing time of $n + 1$, the solution is optimal. The number of instances proven optimal by this test appears as *MIN_PFT_bound*. Finally, the linear bound is calculated. If S_{n+1}^{start} equals the linear bound, the solution is optimal. The number of instances proven optimal by this test appears as *LP_bound*. However, the linear bound is only applied if the number of remaining variables is fewer than 1,500, to avoid lengthy runs. The sum of the instances proven to be optimal by these three tests appears as *Solved to optimality by preprocessing*. The remaining problems are shown on the last line of the table. For more than 60 % of the non-trivial problems, the preprocessing procedures are able to provide a provably optimal solution.

A characteristic of PROGEN 3 is that it tends to produce large values of the planning horizon T. For this set of problems the reduction of T described in Sect. 11.3 is especially useful. Table 11.4 shows the reduction of T obtained by that procedure on the non-optimally solved problems.

The reductions in the planning horizon T, together with the procedures for reducing possible finishing times for the activities, produce significant decreases in the final number of variables to be used by solution procedures, as can be seen in Table 11.5.

Table 11.3 Schirmer problems—optimal solutions identified in the preprocessing

Activities	10	20	30	40	60
Problems	951	960	960	960	960
Feasible problems	946	960	960	960	960
Trivial problems	143	395	507	574	614
Solved to optimality by pre-processing	502	341	291	221	202
CPM_bound	259	287	279	212	201
MIN_PFT_bound	209	46	10	8	1
LP_bound	34	8	2	1	0
Remaining problems	301	224	162	165	144

Table 11.4 Schirmer problems—reductions of planning horizon T

Activities	10	20	30	40	60
Problems	301	224	162	165	144
Average initial T	43	82	120	158	230
Average reduction	21 %	38 %	43 %	48 %	53 %
Maximal reduction	54 %	66 %	68 %	73 %	73 %

Table 11.5 Schirmer problems—reductions of resources and variables

Activities	10	20	30	40	60
Problems	301	224	162	165	144
Initial resources	30	30	30	30	30
Remaining resources (average)	15 (50 %)	15 (50 %)	18 (60 %)	18 (60 %)	20 (67 %)
Initial variables (average)	211	971	2,281	4,256	9,962
Remaining variables (average)	103 (49 %)	338 (35 %)	718 (31 %)	1,188 (28 %)	2,616 (26 %)

11.6.3 Computational Results of Constructive Algorithms

The 32 priority rules described by Schirmer (2000) were coded and embedded in the constructive algorithm in Sect. 11.4.1. In a preliminary computational study these rules were tested on the 879 feasible instances of size 10 generated by Böttcher et al. (1999). Table 11.6 shows the results obtained by the six best performing rules. The first three rules are based on the network structure of the problems. The last three rules are based on resource consumption. In them, SR indicates that the rules require the use of only scarce resources, indexed by k. RK_{ks} is the remaining capacity of resource k at stage s. RD_{ikt} is the relevant demand, defined as $RD_{ikt} = r_{ik}|Q_{it} \cap \Pi_k|$. MDE_{ikt} is the minimum relevant demand entailed for resource k by all successors of activity i when started at period t. The most important feature of Table 11.6 is that even the best rules fail to produce a feasible solution for 20 % of these small instances of size 10. Therefore, we need randomizing strategies and repairing mechanisms to significantly increase the probability of finding feasible solutions in the constructive phase of the GRASP algorithm.

Table 11.6 Results of priority rules

Rule	Definition	Feasible solutions (%)	Optimal solutions (%)
LFT	$\text{Min}\{LFT_j\}$	80.09	64.28
MTS	$\text{Max}\{\|\{i \| j \in P_i'\}\|\}$	79.64	69.98
SLK	$\text{Min}\{LST_j - EFT_j\}$	76.22	61.66
DRC/*SR*	$\text{Max}\{\sum_r (RK_{rs} - RD_{jrt})\}$	81.57	27.08
DRS/*SR*	$\text{Min}\{\sum_r (RK_{rs}/RD_{jrt})\}$	79.29	27.53
TRS/*SR*	$\text{Min}\{\sum_r (RD_{jrt} + MDE_{jrt})\}$	79.41	28.56

The randomization procedures allowed us to get an important increase in the number of feasible solutions. However, not all these small problems could be solved. That was the reason for the development of a repairing mechanism to help the constructive algorithm to find feasible solutions for the more tightly constrained problems. After this preliminary study we decided to use the second randomization procedure, a biased selection on the Restricted Candidate List, and the priority rule *LFT*.

11.6.4 Computational Results of GRASP Algorithms

Table 11.7 contains the results on the 2,553 non-trivial Schirmer instances, including the 60-activity instances we generated. The first part of the table shows the average distance to optimal solutions. However, not all the optimal solutions are known. The optimal solution is not known for one instance of size 30 and five instances of size 40. In these cases, the comparison is made with the best-known solution, obtained by a time-limited run of the CPLEX integer code, by the best version of GRASP algorithms in any of the preliminary tests or by the best version of the Scatter Search algorithm. The second part of the table shows the number of times the best solution does not match the optimal or best known solution, while the third part shows the average computing times in seconds.

Columns 6 to 9 of Table 11.7 show the results of the GRASP algorithms. Four versions have been tested: *GRASP*, the basic GRASP algorithm, *GR+PR*, in which the best solutions obtained in the GRASP iterations go through the Path Relinking phase, *AG-GR*, the aggressive GRASP procedure, and *AG+PR*, combining aggressive GRASP and Path Relinking. The GRASP algorithms use priority rule *LFT* and the second randomization procedure with $\delta = 0.85$. The GRASP algorithm runs for a maximum of 2,000 iterations while for the aggressive procedure in Sect. 11.4.3 we use the limits described there.

The results in Table 11.7 allow us to observe the different performance of the four algorithms more clearly. The aggressive GRASP procedure does not guarantee a better solution than the basic GRASP algorithm for each instance, but it tends to

Table 11.7 Comparison of Scatter Search and GRASP algorithms on Schirmer problems

Non-trivial		Scatter Search			GRASP			
Activities	instances	Regen 0	Regen 1	Regen 6	GRASP	GR+PR	AG-GR	AG+PR
		Average deviation from optimal solution (%)						
10	803	0.02	0.00	0.00	0.00	0.00	0.00	0.00
20	565	0.15	0.03	0.04	0.40	0.33	0.13	0.12
30	453	0.32	0.19	0.10	1.00	0.88	0.24	0.21
40	386	0.59	0.36	0.25	2.03	1.82	0.67	0.59
60	346	1.22	0.90	0.71	3.68	3.31	1.38	1.16
Total	2,553	0.35	0.04	0.03	0.27	0.24	0.07	0.06
		Non-optimal solutions						
10	803	3	0	0	1	1	1	1
20	565	22	10	5	43	32	22	19
30	453	41	29	21	68	63	35	33
40	386	61	41	31	89	84	59	54
60	346	85	74	67	110	105	91	80
Total	2,553	212	154	124	311	285	208	187
		Average running time						
10	803	1.2	1.8	2.5	0.9	0.9	0.3	0.3
20	565	3.0	3.4	17.0	1.4	1.4	1.1	1.2
30	453	7.2	11.8	28.8	2.9	3.1	3.4	3.7
40	386	15.7	25.8	51.1	5.7	6.2	2.9	7.2
60	346	55.8	105.2	175.9	8.7	10.3	10.6	13.4
Total	2,553	18.3	33.6	60.1	3.6	4.1	4.0	4.9

produce better results especially for large problems. The Path Relinking algorithm adds little improvement to the good results obtained by GRASP procedures.

The last lines of the table provide the running times of the algorithms, in CPU seconds. In all cases preprocessing is included as a part of the solution procedure. The algorithms have been coded in *C*++ and run on a Pentium IV at 2.8 GHz. The average running times are very short, though some problems would require quite long times. In order to avoid excessively long runs, we have imposed a time limit of 60 s for the GRASP procedures. This limit negatively affects the solution of some hard instances, which could have obtained better solutions in longer times, but in general it ensures a good balance between solution quality and required running time. The addition of the Path Relinking procedure does not increase the running times very much, in part because we have included a mechanism to detect solutions which have already been explored and avoid submitting them to the improvement procedure. Therefore it seems convenient to keep it in the final implementation. Therefore, the aggressive GRASP algorithm with Path Relinking seems to be the best option for this type of metaheuristic algorithm.

11.6.5 Computational Results of Scatter Search Algorithms

In order to obtain the initial population, the GRASP algorithm is run until 100 different feasible solutions are obtained or the limit of 2,000 iterations is reached. From the initial population, a reference set *RefSet* of $b = 10$ solutions is built, with $b_1 = 5$ quality solutions and $b_2 = 5$ diverse solutions.

In a preliminary experience, the eight combination procedures described in Sect. 11.5 have been tested on Schirmer's problems. In this case, no regeneration of the reference set is included and the algorithm stops whenever no new solutions can be added to the Reference Set after a combination phase. We could see that all the combinations except for Combination 7 obtained similar results. So for further testing we keep Combinations 1 and 8, which produce the best results and have completely different structures. We could observe that the basic Scatter Search algorithm is very efficient, obtaining optimal solutions for most of the 3,826 feasible Schirmer test instances. Therefore, an additional step in which the reference set is regenerated will only be justified if it helps to solve the hardest problems, those not solved by the basic algorithm. The regeneration procedure depends on three parameters: the number of iterations of the modified GRASP algorithm, the number of new solutions obtained, and the number of times the regeneration process is called. We have considered the following values for these parameters:

1. Maximum number of iterations: 500–1,000
2. Maximum number of new solutions: 20–50
3. Calls to regenerate: three times—Only when the solution is improved after the last call to regenerating

Table 11.7 shows the results for three versions of the Scatter Search algorithm: Regen 0 (without regeneration), Regen 1 (500 iterations, 20 solutions, the regeneration is called while the solution is improved), Regen 6 (1,000 iterations, 50 solutions, regeneration is called three times), and compare them with the best version of the GRASP algorithms. The table shows that while the GRASP algorithm is very efficient and can obtain better solutions on small problems, the Scatter Search procedure with increasingly complex regeneration strategies can significantly improve the overall results with a moderate increase in the running times. Similar results were obtained for the test instances of Böttcher et al. (1999).

11.7 Conclusions

Partially renewable resources, in which the availability of the resource is associated to a given set of periods and the activities only consume it when they are processed in these periods, can be seen as a generalization of renewable and non-renewable resources, but their main interest comes from their usefulness for modeling complex situations appearing in timetabling and labor scheduling problems, which can be approached as project scheduling problems.

Preprocessing techniques which help to determine the existence of feasible solutions and to reduce the number of variables and constraints are specially useful for this type of problems because of the existence of a time horizon in which the partially renewable resources are defined.

In this chapter we have reviewed existing formulations and exact algorithms but we have focused on preprocessing techniques and heuristic algorithms, ranging form simple priority rules to sophisticated GRASP/Path Relinking and Scatter Search procedures. The computational results show that these metaheuristic algorithms are able to produce high quality solutions in moderate computing times.

Acknowledgements This work has been partially supported by the Spanish Ministry of Education and Science DPI2011-24977.

References

Alvarez-Valdes R, Crespo E, Tamarit JM, Villa F (2006) GRASP and path relinking for project scheduling under partially renewable resources. Eur J Oper Res 189:1153–1170

Alvarez-Valdes R, Crespo E, Tamarit JM, Villa F (2008) A scatter search algorithm for project scheduling under partially renewable resources. J Heuristics 12:95–113

Böttcher J, Drexl A, Kolisch R, Salewski F (1999) Project scheduling under partially renewable resource constraints. Manage Sci 45:544–559

Chaudhuri S, Walker RA, Mitchell JE (1994) Analyzing and exploiting the structure of the constraints in the ILP approach to the scheduling problem. IEEE T VLSI Syst 2:456–471

Christofides N, Alvarez-Valdes R, Tamarit JM (1987) Project scheduling with resource constraints: a branch and bound approach. Eur J Oper Res 29:262–273

de Boer(1998) Resource-constrained multi-project management: a hierarchical decision support system. Ph.D. dissertation, University of Twente, Twente, The Netherlands

Kolisch R, Sprecher A, Drexl A (1995) Characterization and generation of a general class of resource-constrained project scheduling problems. Manage Sci 41:1693–1703

Laguna M, Marti R (2004) Scatter search. Kluwer, Boston

Mellentien C, Schwindt C, Trautmann N (2004) Scheduling the factory pick-up of new cars. OR Spectr 26:579–601

Neumann K, Schwindt C, Trautmann N (2002) Advanced production scheduling for batch plants in process industries. OR Spectr 24:251–279

Neumann K, Schwindt C, Trautmann N (2005) Scheduling of continuous and discontinuous material flows with intermediate storage restrictions. Eur J Oper Res 165:495–509

Resende, MGC, Ribeiro CC (2003) Greedy randomized adaptive search procedures. In: Glover F, Kochenbeger G (eds) Handbook of metaheuristics. Kluwer, Boston, pp 219–249

Schirmer A (2000) Project scheduling with scarce resources. Dr. Kovac, Hamburg

Schwindt C, Trautmann N (2003) Scheduling the production of rolling ingots: industrial context, model and solution method. Int Trans Oper Res 10:547–563

Shewchuk JP, Chang TC (1995) Resource-constrained job scheduling with recyclable resources. Eur J Oper Res 81:364–375

Talbot FB, Patterson JH (1978) An efficient integer programming algorithm with network cuts fo solving resource-constrained project scheduling problems. Manage Sci 24:1163–1174

Part IV
Preemptive Project Scheduling

Chapter 12
Integer Preemption Problems

**Sacramento Quintanilla, Pilar Lino, Ángeles Pérez,
Francisco Ballestín, and Vicente Valls**

Abstract A fundamental assumption in the basic *RCPSP* is that activities in progress are non-preemptable. Some papers reveal the potential benefits of allowing activity interruptions in the schedule when the objective is the makespan minimization. In this chapter we consider the *Maxnint_PRCPSP* in which it is assumed that activities can be interrupted at any integer time instant with no cost incurred, that each activity can be split into a maximum number of parts, and that each part has a minimum duration established. We show how some procedures developed for the *RCPSP* can be adapted to work with the *Maxnint_PRCPSP* and we introduce some procedures specifically designed for this problem. Furthermore, precedence relationships between activities can refer to portions of work content or periods of time. In single-modal project scheduling when interruption is not allowed, both are equivalent but not when preemption is considered. We present a study of generalized work and time precedence relationships and all conversions amongst them.

Keywords Integer preemption • Makespan minimization • Project scheduling • Time and work generalized precedence relationships

12.1 Introduction

The interruption of an activity of a project while being executed is a situation that often appears in practice and that many commercial software packages contemplate. However, there is not much research work dealing with the interruption of activities in the *RCPSP* context.

S. Quintanilla (✉) • P. Lino • Á. Pérez • F. Ballestín
Department of Mathematics for the Economy, University of Valencia, Valencia, Spain
e-mail: maria.quintanilla@uv.es; pilar.lino@uv.es; angeles.perez@uv.es; francisco.ballestin@uv.es

V. Valls
University of Valencia, Valencia, Spain
e-mail: vicente.valls@uv.es

C. Schwindt, J. Zimmermann (eds.), *Handbook on Project Management and Scheduling Vol. 1*, International Handbooks on Information Systems,
DOI 10.1007/978-3-319-05443-8_12

The interruption of an activity can be considered discrete (it is only accepted at any integer time instant) or continuous (the activity can be interrupted at whatever moment during its execution). In this chapter we focus on the first case (discrete interruption), while the following chapter deals with the continuous case. The generalized precedence relationships (GPR) have been studied in Chaps. 5–7 of this handbook and Ballestín et al. (2013) in the context of single-modal and multi-modal project scheduling, respectively. When preemption is allowed and/or the duration of an activity depends on the mode, the lag of a GPR cannot equivalently be expressed in terms of elapsed time or in terms of work content. This chapter shows how to deal with this situation.

Kaplan (1988, 1991) was the first to study the preemptive resource-constrained project scheduling problem (*PRCPSP*) where preemption at integer points is allowed. Demeulemeester and Herroelen (1996) proposed a Branch and Bound algorithm for the *PRCPSP* and proved that the procedure proposed by Kaplan was wrong. Ballestín et al. (2008) presented a procedure to solve the *RCPSP* in the case where one interruption is allowed per activity (*1_PRCPSP*), always at integer points. This was the first paper which made clear that preemption does significantly help to decrease the project length with respect to the non-interruption case. Previous studies did not reach this conclusion probably because the computational tests were made on the Patterson instance set (Patterson 1984). Li et al. (2011) developed a particle swarm optimization method to solve the *RCPSP* and the *1_PRCPSP* and the authors concluded that preemption helps to reduce the project duration significantly. The *1_PRCPSP* can be generalized to the *ρ_PRCPSP* where ρ interruptions are allowed per activity, always at integer points.

Ballestín et al. (2009) introduced a model for discrete preemption in the *RCPSP* where the maximum number of times that each activity can be interrupted and the minimum length of each subactivity can be fixed. The problem was named *Maxnint_PRCPSP* and the authors introduced an evolutionary algorithm with a suitable codification and crossover to solve the problem and they studied preemption in the presence of due dates. The conclusion was that preemption is useful and helps to find better solutions when due dates are present. Also, the greater the number of interruptions per activity that are allowed the higher the quality of the solutions. However, it is shown that allowing more than two interruptions does not seem to add much to the solution quality.

Van Peteghem and Vanhoucke (2010) presented a genetic algorithm for the multi-modal *RCPSP* and *PRCPSP*, concluding that activity preemption always leads to better solutions. Quintanilla et al. (2012) worked with a complex model in the context of a Service Centre. The problem was modeled as a multi-modal project scheduling problem with due dates, each activity needed a single unit of resource, preemption was allowed and time and work generalized precedence relationships with minimal and maximal time lag between the tasks were considered. In this context, time and work relationships are not equivalent and the authors presented a complete study of work and time GPRs which included proper definitions, a new notation and all possible conversions among them. A review of project scheduling with generalized precedence relationships can be found in Neumann et al. (2003).

The rest of the chapter is organized as follows: Sect. 12.2 models the *Maxnint_PRCPSP*. Section 12.3 introduces a codification to the solutions of the *Maxnint_PRCPSP* in such a way that some procedures developed for the *RCPSP* can be directly used for the *Maxnint_PRCPSP*. Section 12.4 adapts some of these procedures to the new problem with the aim of obtaining better solutions. The modifications consist in introducing new breaks in the activities and changes in the initial duration of each part or subactivity. Sections 12.5 and 12.6, respectively, describe and compare 11 algorithms to solve the *Maxnint_PRCPSP*. In Sect. 12.7 we present a study of work and time generalized precedence relationships and all conversions amongst them. Finally, conclusions are given in Sect. 12.8.

12.2 The *Maxnint_PRCPSP*

The preemptive resource-constrained project scheduling problem, *PRCPSP* ($PS|prec, pmtn/int|C_{max}$), is defined as the *RCPSP* with the assumption of the non-preemption of activities relaxed. A project consists of $n + 2$ activities numbered 0 to $n + 1$ where dummy activities $i = 0$ and $i = n + 1$ represent the beginning and completion of the project. There are precedence relations between the activities. $E = \{(i, j) \in V \times V | i \text{ must finish before } j \text{ can start}\}$ is the set of those relationships where V is the set of activities. Each activity i has a duration p_i and needs some renewable resources to ensure it is carried out. There are K different resource types with availability in each time period of R_k units, $k = 1, \ldots, K$. Each activity i requires r_{ik} units of resource k during each period of time in which it is processed. Dummy activities $i = 0$ and $n+1$ have zero duration and resource usage. In the *PRCPSP*, activities are allowed to be preempted at any integer time instant and resumed later on with no cost incurred. This problem can be solved by splitting each activity i in p_i parts (subactivities) of one unit duration each and introducing a precedence relationship between two consecutive parts. The *Maxnint_PRCPSP* adds the restriction that each activity i can be preempted a maximum number of times ($maxnint_i$) and each part should have a minimum duration (ε_i). The *RCPSP* is the *Maxnint_PRCPSP* with $maxnint_i = 0$ or equivalently $\varepsilon_i = p_i$ and the *PRCPSP* is the *Maxnint_PRCPSP* with $maxnint_i = p_i - 1$. All parameters are assumed to be non-negative integer values.

We consider $maxnint_0 = 0$ and $maxnint_{n+1} = 0$ and we can update $maxnint_i = \min(maxnint_i, \lfloor p_i/\varepsilon_i \rfloor - 1)$ $\forall i = 1, \ldots, n$. In the rest of the chapter we will consider that this update has been made.

The conceptual model of the *Maxnint_PRCPSP*, was introduced in Ballestín et al. (2009) as follows:

$$\text{Min. } S_{n+1,1} \tag{12.1}$$

$$\text{s.t. } S_{01} = 0 \tag{12.2}$$

$$S_{j1} \geq S_{i,maxnint_i+1} + p_{i,maxnint_i+1} \quad ((i,j) \in E) \tag{12.3}$$

$$\sum_{i \in \mathscr{A}(S,t)} r_{ik} \leq R_k \qquad (t = 0, \ldots, UB; k = 1, \ldots, K) \tag{12.4}$$

$$S_{i,\mu+1} \geq S_{i\mu} + p_{i\mu} \; (i \in V \text{ with } maxnint_i > 0; \mu = 1, \ldots, maxnint_i) \tag{12.5}$$

$$\sum_{j=1}^{maxnint_i+1} p_{i\mu} = p_i \qquad (i \in V) \tag{12.6}$$

$$p_{i\mu} \geq \varepsilon_i y_{i\mu} \quad (i \in V; \mu = 1, \ldots, maxnint_i + 1) \tag{12.7}$$

$$y_{i,\mu+1} \leq y_{i\mu} \qquad (i \in V \text{ with } maxnint_i > 0; \mu = 1, \ldots, maxnint_i) \tag{12.8}$$

$$S_{i\mu}, p_{i\mu} \in \mathbb{Z}_{\geq 0} \qquad (i \in V; \mu = 1, \ldots, maxnint_i + 1) \tag{12.9}$$

$$y_{i\mu} \in \{0, 1\} \quad (i \in V; \mu = 1, \ldots, maxnint_i + 1) \tag{12.10}$$

where the variables used have the following meaning:

- $S_{i\mu}$ = start time of the μ-th part of activity i, $\mu \leq maxnint_i + 1$
- $p_{i\mu}$ = processing time of the μ-th part of activity i, $\mu \leq maxnint_i + 1$
- $y_{i\mu}$ is 1 if part μ of activity i is a non-dummy part, i.e., does not have duration 0, $\mu \leq maxnint_i + 1$. If an activity is split into fewer parts than $maxnint_i + 1$, the last parts of the activity will be dummy parts with duration 0. Their start and finish times will be the finish time of the last non-dummy part.

The objective of the problem is to minimize the makespan. Equation (12.2) assigns the start time 0 to the dummy activity 0. The precedence constraints given by (12.3) indicate that the start of an activity j must wait for the end of the last subactivity of all the predecessor activities of j. For each renewable resource type, Eq. (12.4) indicates that the resource amount required by the activities in progress cannot exceed the resource availability, $t = 0, \ldots, UB$ where UB is an upper bound on the project length, e.g., the sum of the activities' durations. Equation (12.5) ensures that a subactivity of an activity i does not start sooner than the end of the previous subactivity of the same activity. Conditions on the durations of subactivities are reflected in (12.6)–(12.8). In (12.6) the sum of the durations of all parts of an activity i must be equal to the processing time of i. In Eq. (12.7) the duration of each subactivity must be at least the minimum duration ε_i, but only if the part is a non-dummy one. Restrictions (12.8) ensure that the non-dummy parts of each activity are the first parts. The variables $y_{i\mu}$ are binary (12.10) and the rest of the variables must be non-negative integers, as stated in (12.9).

Since the *RCPSP* is a particular case of the *Maxnint_PRCPSP*, our problem is $\mathscr{NP}$-hard in the strong sense (see Błażewicz et al. 1983 for the $\mathscr{NP}$-hardness of the *RCPSP*).

The most important aspect of the problem is that we can work at the same time with non-preemptable activities ($maxnint_i = 0$) and preemptable activities ($maxnint_i > 0$), in addition to activities with different numbers of allowed interruptions.

It is obvious that if we know the value of $S_{i\mu}$ and $p_{i\mu}$ for non-dummy subactivities we can easily obtain the value of the remaining variables ($y_{i\mu}$). For this reason, a schedule or solution is determined knowing only these values.

12.3 Using Existing Procedures for the RCPSP to Solve the *Maxnint_PRCPSP*

Some good heuristic algorithms developed for the *RCPSP* (Kolisch and Hartmann 2006) use the activity list as a codification of a solution, the serial schedule-generation scheme (SGS) to decode it, the rapid Double Justification local search, and the one-point or two-point crossovers.

The serial SGS is capable of generating every active schedule, therefore it is able to find an optimal solution for each feasible instance, as the set of active schedules always contains an optimal schedule.

This section shows how to adapt the codification of a solution to directly use these three procedures for the *Maxnint_PRCPSP*.

12.3.1 Codification of a Solution

A codification of a solution for the *Maxnint_PRCPSP* has been introduced in Ballestín et al. (2009) by means of a pair of vectors $\gamma = (\ell, \Omega)$, a subactivity list vector ℓ, and a duration vector Ω. ℓ contains each activity as many times as subactivities of this activity are considered. $\Omega(u)$ stores the processing time of a subactivity of the activity $\ell(u)$. Each activity i will always appear behind their predecessors in the vector ℓ and i can appear a maximum of $maxnint_i + 1$ times. A precedence relationship is considered to exist between the subactivities corresponding to two consecutive occurrences of the same activity in ℓ. As $\Omega(u)$ corresponds to the processing time of one subactivity of activity $\ell(u)$, $\Omega(u)$ has to be greater than or equal to $\varepsilon_{\ell(u)}$. The sum of the components of Ω corresponding to the same activity i in ℓ must be equal to p_i.

Given a solution of the *Maxnint_PRCPSP* we define the representation of it as the pair $\gamma = (\ell, \Omega)$ constructed as follows: The lengths of ℓ and Ω are equal to the number of non-dummy parts in the solution $\left(\sum_{i=1}^{n} \sum_{\mu=1}^{maxnint_i+1} y_{i\mu}\right)$. Instants $S_{i\mu}$ corresponding to non-dummy parts are sorted in increasing order. If $S_{i\mu}$ occupies position v, then i and $p_{i\mu}$ occupy position v in ℓ and Ω respectively.

As an example, the representation of the last schedule in Fig. 12.1 is:

$$\ell = (1, 3, 2, 4, 1, 3, 2, 5, 4, 3, 5)$$

$$\Omega = (3, 7, 4, 5, 4, 1, 2, 3, 3, 4, 3)$$

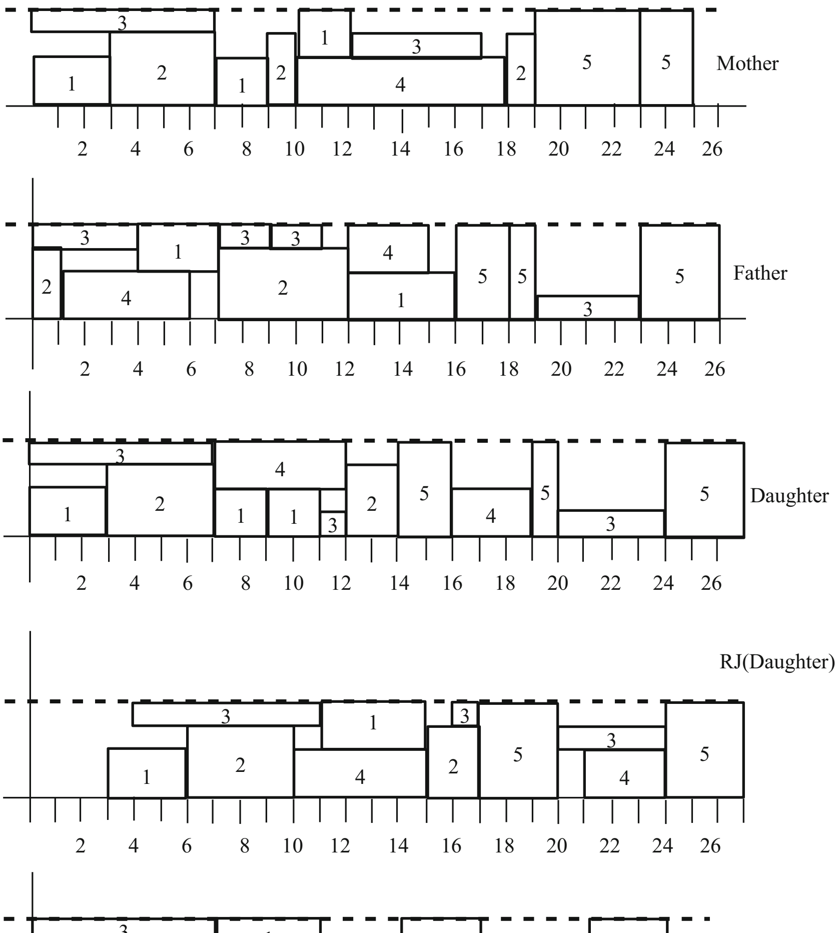


Fig. 12.1 Schedules obtained by applying serial SGS to the mother, father, and daughter. Right justification and double justification applied to the schedule associated with the daughter

12.3.2 The Serial SGS

Given $\gamma = (\ell, \Omega)$, the serial SGS can be used to obtain a feasible schedule S for the *Maxnint_PRCPSP*. First, the dummy activity 0 is started at time 0. Then the subactivities are scheduled following the order established in ℓ during the processing time given by Ω. Note that a subactivity can be started only if all previous subactivities for the same activity in the list have been completed.

Note that if S is the schedule obtained by application of the serial SGS to the pair (ℓ, Ω) and (ℓ', Ω') is a representation of S, then the serial SGS applied to (ℓ', Ω') yields S. Note also that ℓ' and Ω' are not necessarily equal to ℓ and Ω respectively and they may even have different dimensions.

As an example, let us consider $\ell = (1, 2, 1, 3, 2, 4, 1, 2, 5, 3, 5)$ and $\Omega = (3, 4, 2, 7, 1, 8, 2, 1, 4, 5, 2)$. The serial SGS applied to (ℓ, Ω) produces the schedule $serialSGS(\ell, \Omega)$ that corresponds to the first schedule in Fig. 12.1. However, the representation of that schedule is (ℓ', Ω') where $\ell' = (1, 3, 2, 1, 2, 4, 1, 3, 2, 5)$ and $\Omega' = (3, 7, 4, 2, 1, 8, 2, 5, 1, 6)$.

12.3.3 The Double Justification

In the *RCPSP* context, the double justification is a technique that acts on a feasible schedule, it never increases the makespan of the schedule and in many cases it shortens it. This technique (Valls et al. 2005) is equivalent to applying the serial SGS twice. Firstly, in the reverse project, it is applied to the activity list (for the reverse project) made up of the activities in non-increasing order of their finishing times. Secondly, in the original project, it is applied to the activity list composed of the activities in non-decreasing order of their starting times in the last calculated schedule. The reverse project is the network obtained by simply reversing the precedence relationships of the original project. Therefore, double justification can be used for the *Maxnint_PRCPSP* using the content from the previous two sections. Both the project and the reverse project must consider the relationships linking any two consecutive subactivities of the schedule.

12.3.4 One-Point and Two-Point Crossovers

The one-point and two-point crossovers proposed by Hartmann (1998) act on vectors of equal length. As the vectors in two different codifications can have a different number of components, to work with the Hartmann crossovers, we construct a pair of vectors of constant length for each codification. The crossovers are applied on the new vectors and finally we transform the resulting vectors into codifications, as follows:

Given (ℓ, Ω), we construct $V1$ and $V2$ of a constant length equal to $maxnsub = \sum_{i \in V}(maxnint_i + 1)$, the maximum number of subactivities in any feasible schedule. $V1$ is constructed following the order in ℓ but adding dummy activities to ensure that each activity i appears $maxnint_i + 1$ times. The dummy activities associated with activity i appear in the new vector after the last occurrence of i in ℓ. $V2$ is constructed by copying Ω and inserting a zero every time a dummy activity is inserted into $V1$.

To apply the crossovers we consider the mother $\gamma^M = (\ell^M, \Omega^M)$, the father $\gamma^F = (\ell^F, \Omega^F)$, and the pair of vectors corresponding to the mother $(V1^M, V2^M)$ and the father $(V1^F, V2^F)$. As a result we will obtain two children: a daughter $(V1^D, V2^D)$ and a son $(V1^S, V2^S)$. To construct the children, firstly, we apply the Hartmann crossover to the subactivity lists in $V1^M$ and $V1^F$ obtaining $V1^D$ and $V1^S$. Then we calculate component by component the second vector of each child as follows:

Let u be $1 \leq u \leq maxnsub$ and let i be the activity stored in $V1^D(u)$. If i comes from the mother then $V2^D(u)$ belongs to the set $\{0, \varepsilon_i, \varepsilon_i + 1, \varepsilon_i + 2, \ldots, V2^M(u)\}$. To decide the value, we first calculate t as:

$$t = \sum_{v|V1^M(v)=i, v\leq u} V2^M(v) - \sum_{v|V1^D(v)=i, v<u} V2^D(v)$$

Then, if $t < \varepsilon_i$, $V2^D(u) = 0$, otherwise $V2^D(u) = \min(V2^M(u), t)$.

Similarly, if activity i stored in $V1^D(u)$ comes from the father, then $V2^D(u)$ belongs to the set $\{0, \varepsilon_i, \varepsilon_i + 1, \varepsilon_i + 2, \ldots, V2^F(u)\}$. To decide the value, we first calculate t as:

$$t = \sum_{v|V1^F(v)=i, v\leq u} V2^F(v) - \sum_{v|V1^D(v)=i, v<u} V2^D(v)$$

Then, if $t < \varepsilon_i$, $V2^D(u) = 0$, otherwise $V2^D(u) = \min(V2^F(u), t)$.

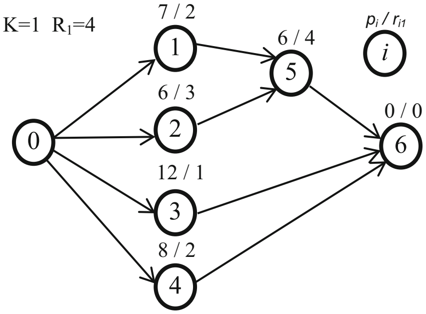
After obtaining $V1^D$ and $V2^D$, every time that a component is equal to zero in $V2$, this component is erased in both vectors. Analogously, $V2^S(u)$ and a codification for the son are obtained.

A desirable property of a crossover is that the children inherit the structures shared by the parents. The crossover introduced has this property. Activities with just one appearance in both activity lists appear also just once in the children. Also, activities with the same number of parts and the same part durations in both parents are also transferred in that way to the children.

The two-point crossover introduced in Ballestín et al. (2008) for the *1_PRCPSP* coincides with the two-point crossover introduced here if we consider $maxnint_i = \varepsilon_i = 1, \forall\, i = 1, \ldots, n$.

As an example, consider the project in Fig. 12.2. The project consists of five non-dummy activities ($n = 5$) and two dummy activities (0 and 6) with one renewable resource of availability 4 ($K = 1, R_1 = 4$). The activities are represented in the vertices. The edges represent precedence relationships. If there is a directed edge from a vertex i to vertex j, it is because $(i, j) \in E$. Above each vertex is shown the duration of the activity and the number of units of the resource needed to ensure it is carried out. We suppose that $maxnint_1 = maxnint_3 = 4$, $maxnint_2 = maxnint_5 = 3$, $maxnint_4 = 1$; $\varepsilon_1 = \varepsilon_2 = \varepsilon_3 = \varepsilon_5 = 1$ and $\varepsilon_4 = 3$. We consider as the mother

Fig. 12.2 Project



the pair $\ell^M = (1, 2, 1, 3, 2, 4, 1, 2, 5, 3, 5)$ and $\Omega^M = (3, 4, 2, 7, 1, 8, 2, 1, 4, 5, 2)$ and as the father the pair $\ell^F = (2, 3, 4, 1, 2, 1, 5, 3, 4, 3, 5, 3, 5)$ and $\Omega^F = (1, 4, 5, 3, 5, 4, 2, 2, 3, 2, 1, 4, 3)$.

To do the crossover we transform the two pairs in the following vectors of 20 components ($maxnsub = 5 + 4 + 5 + 2 + 4 = 20$).

$$V1^M = (1, 2, 1, 3, 2, 4, 4, 1, 1, 1, 2, 2, 5, 3, 3, 3, 3, 5, 5, 5)$$
$$V2^M = (3, 4, 2, 7, 1, 8, 0, 2, 0, 0, 1, 0, 4, 5, 0, 0, 0, 2, 0, 0)$$
$$V1^F = (2, 3, 4, 1, 2, 2, 2, 1, 1, 1, 1, 5, 3, 4, 3, 5, 3, 3, 5, 5)$$
$$V2^F = (1, 4, 5, 3, 5, 0, 0, 4, 0, 0, 0, 2, 2, 3, 2, 1, 4, 0, 3, 0)$$

If we apply the one-point crossover considering the first four positions from the mother and the remaining sixteen positions completed with the subactivities not previously considered and in the order of the father, we obtain:

$$V1^D = (1, 2, 1, 3, 4, 2, 2, 2, 2, 1, 1, 1, 5, 3, 4, 3, 5, 3, 3, 5, 5)$$
$$V2^D = (3, 4, 2, 7, 5, 2, 0, 0, 2, 0, 0, 2, 0, 3, 1, 1, 4, 0, 3, 0)$$

and the corresponding codification, by erasing the zero components, is:

$$\ell^D = (1, 2, 1, 3, 4, 2, 1, 5, 4, 3, 5, 3, 5)$$
$$\Omega^D = (3, 4, 2, 7, 5, 2, 2, 2, 3, 1, 1, 4, 3)$$

Figure 12.1 shows the corresponding schedules after applying the serial SGS on the mother, father, and daughter. The fourth (fifth) schedule is the result of applying right justification (left justification) to the third (fourth) schedule. In brief, the last schedule is $DJ(serialSGS(\ell^D, \Omega^D))$ and their representation is:

$$\ell = (1, 3, 2, 4, 1, 3, 2, 5, 4, 3, 5)$$
$$\Omega = (3, 7, 4, 5, 4, 1, 2, 3, 3, 4, 3)$$

12.4 Specific Procedures Developed for the *Maxnint_PRCPSP*

The procedures described in the above section can be adapted to allow more breaks and changes in the lengths of the parts. As a result, it is possible to obtain better schedules. Ballestín et al. (2009) have adapted the three procedures to the *Maxnint_PRCPSP* and have presented a new procedure, the unary operator, a mutation operation introduced for this particular problem. We shall look at each one of these procedures.

12.4.1 Decoding Procedure: Maxnint_Serial SGS

Given $\gamma = (\ell, \Omega)$, we define $nsub_i$ as the number of occurrences of the activity i in ℓ and $totalnsub = \sum_{i=1}^{n} nsub_i$; $totalnsub$ is therefore the dimension of vectors ℓ and Ω.

The *Maxnint_Serial SGS* works as the *serialSGS* explained in Sect. 12.3.2 but the procedure permits more interruptions. During the creation of the schedule the procedure changes the codification according to the current number of subactivities and their duration. The procedure schedules one by one the subactivities that appear in ℓ. When scheduling a subactivity $\ell(u) = i$, the procedure calculates t as the finish time of the last scheduled predecessor of subactivity i. Two cases are considered:

If t coincides with the finish time of a subactivity of activity i:

Let h be the maximum value such that in $[t, t+h[$ there are enough free resources to schedule i. If $h \geq \Omega(u)$ or $\Omega(u) = \varepsilon_i$, the subactivity $\ell(u)$ is scheduled as in the serial SGS. If the subactivity is started in t, the component in Ω corresponding to the processing time of the subactivity of i that finished in t is increased in $\Omega(u)$ units, $nsub_i = nsub_i - 1$, and vectors ℓ, Ω are updated by decreasing their dimension in one unit and erasing component u in both vectors. If $h < \Omega(u)$ and $\Omega(u) \neq \varepsilon_i$, the procedure schedules $h1$ units of i at t where $h1 := \min(h, \Omega(u) - \varepsilon_i)$; the component in Ω corresponding to the processing time of the subactivity of i that finished in t is increased in $h1$ units, $\Omega(u)$ is updated as $\Omega(u) - h1$, and the procedure newly considers the subactivity $\ell(u)$.

Otherwise:

If $nsub_i - 1 = maxnint_i$ or $\Omega(u) < 2 \cdot \varepsilon_i$, then the subactivity i is scheduled as in the serial SGS. Otherwise, let $t1 \geq t$ be the minimum time instant, such that in $[t1, t1 + \varepsilon_i[$ there are enough free resources to schedule i and let h be the maximum value such that between $[t1, t1 + h[$ there are enough free resources to schedule i. If $h \geq \Omega(u)$, i is scheduled as in the serial SGS. Otherwise, activity i

will be interrupted. In this case, *Maxnint_Serial* SGS schedules the first $h1 := \min(h, \Omega(u) - \varepsilon_i)$ units of i in $[t1, t1+h1[$, makes $nsub_i = nsub_i + 1$, and updates vectors ℓ, Ω by increasing their dimension in one unit and inserting a component after the component u: $\ell(u+1) = i$, $\Omega(u) = h1$ and $\Omega(u+1) = \Omega(u) - h1$.

12.4.2 Maxnint_DJ

Maxnint_DJ consists in applying *Maxnint_Serial* SGS twice, firstly in the reverse project and secondly in the original project. *Maxnint_DJ* is the same procedure as the double justification presented in Sect. 12.3.3, but using *Maxnint_Serial SGS* instead of serial SGS. *Maxnint_DJ* can modify the interruptions in a schedule, something that did not occur with the double justification. Note that although new preemptions can appear, the procedure never increases the makespan of a schedule. Also, the double justification adapted to the *1_PRCPSP* presented in Ballestín et al. (2008) matches *Maxnint_DJ* procedure when $\varepsilon_i = 1$ and $maxnint_i = 1, \forall i = 1, \ldots, n$.

12.4.3 BeginEndOnePointCrossover

The procedure obtains two children, daughter, and son from a mother $\gamma^M = (\ell^M, \Omega^M)$ and a father $\gamma^F = (\ell^F, \Omega^F)$. During the creation of the children the procedure fills vectors ℓ and Ω. When the children have been partially built, we can calculate $leftdur_i$ as the processing time of activity i not yet considered in vector Ω: $leftdur_i = p_i - \sum_{u=1}^{maxnsub} \Omega(u) | \lambda(u) = i$.

To generate a daughter, the procedure considers two empty vectors ℓ^D and Ω^D with initial length *maxnsub* and generates a random integer number q between 1 and $totalnsub^M$. The first q positions in vectors ℓ^D and Ω^D are copied from the mother. Next, the procedure iteratively considers the father components, starting from the last. Let u be the last component not considered yet in the father such that $leftdur^D_{\lambda^F(u)} \neq 0$ and let v be the last empty position in the daughter. Let i be the activity stored in $\ell^F(u)$. Then $\ell^D(v) = \ell^F(u)$. If $nsub^D_i < maxnint_i - 1$ and $leftdur^D_i \geq \varepsilon_i + \Omega^F(u)$, then $\Omega^D(v) = \Omega^D(u)$, otherwise, i is not further interrupted in the daughter, and $\Omega^D(v) = leftdur^D_i$. The procedure ends by erasing the blank cells from ℓ^D and Ω^D. Similarly, a son can be obtained by interchanging the roles of the father and the mother.

If we apply the *BeginEndOnePointCrossover* on the mother and father in the last example, with $q = 4$, we obtain the daughter:

$$\ell^D = (1, 2, 1, 3, 4, 1, 5, 3, 4, 3, 5, 3, 5)$$

$$\Omega^D = (3, 4, 2, 7, 5, 2, 2, 1, 3, 2, 1, 4, 3)$$

12.4.4 Unary Operator

This unary operator is a mutation operator that acts on a codification and changes the position of some subactivities in ℓ and/or reduces the number of interruptions of some activities. The authors also tested allowing more interruptions, but preliminary tests showed that this does not improve the quality of the final solutions. They also commented that one reason could be that other procedures such as *Maxnint_Serial* SGS and *Maxnint_DJ* already produce new interruptions in the solutions.

The *UnaryOperator* selects $\pi_{mut} \cdot totalnsub$ subactivities, where π_{mut} is the mutation probability. If a subactivity $i = \ell(u)$ is selected and $nsub_i > 1$ then, with probability equal to 0.5, the number of interruptions of activity i is reduced by one and, with probability equal to 0.5, the position in ℓ of this subactivity is changed, if it is possible. Otherwise ($nsub_i = 1$), the position in ℓ of this subactivity is changed, if it is possible.

To reduce the number of interruptions, the component in Ω corresponding to the first processing time of a subactivity of i is increased in $\Omega(u)$, $nsub_i = nsub_i - 1$, and vectors ℓ, Ω are updated by decreasing their dimension in one unit and erasing component u in both vectors.

To change the position of the subactivity in ℓ the procedure calculates the first and last possible positions to place subactivity $i = \ell(u)$ within ℓ so that ℓ remains a list of activities, taking into account the precedence relationships among the subactivities of activity i already defined. If more than one position is possible, a position is randomly selected and the unary operator relocates i in this position, maintaining the relative order of the rest of the subactivities in ℓ. Obviously Ω is consistently updated.

12.5 Algorithms for the *Maxnint_PRCPSP*

Ballestín et al. (2008) tested the influence of interruption in the case of one interruption being allowed per activity (*1_PRCPSP*), when activity lists are randomly generated and when one of the best algorithms for the *RCPSP* is considered (the genetic algorithm of Hartmann 1998). Also, the influence of the double justification was tested when one interruption is allowed. Eight algorithms were considered in total, based on one random and one genetic algorithm and adding, or not, interruption and double justification. Ballestín et al. (2009) developed an evolutionary algorithm, *EvoAlg*, for the *Maxnint_PRCPSP*. Twelve different versions of the algorithm were tested by the authors by varying some parameters and techniques. We have considered the best version. Table 12.1 summarizes the 11 algorithms considered in this chapter for solving problems *RCPSP*, *1_PRCPSP*, *PRCPSP*, and *Max_nintPRCSP*. The considered algorithms work until a certain number of schedules (*nschedules*) are generated.

Table 12.1 Algorithms for the *RCPSP*, *1_PRCPSP*, *PRCPSP*, and *Maxnint_PRCPSP*

Algorithm	Problem	Explanation
Random	*RCPSP*	Generates *nschedules* random activity lists and applies *serial SGS* to each one
GA	*RCPSP*	Genetic algorithm of Hartmann (1998) with a maximum of *nschedules* schedules
DJRandom	*RCPSP*	Generates *nschedules*/3 random activity lists and applies serial SGS to each one. Finally, applies double justification to each schedule
DJGA	*RCPSP*	*GA* modified by applying double justification to each evaluated schedule until a maximum of *nschedules* schedules
HGA	*RCPSP*	Hybrid Genetic algorithm of Valls et al. (2008) with a maximum of *nschedules* schedules
1_Random	*1_PRCPSP*	Randomly generates *nschedules* codifications $\gamma = (\ell, \Omega)$ introduced in 12.3.1. The activity list ℓ contains each activity twice. The processing times in Ω are randomly generated. *Maxnint_serial SGS* is applied to each codification
1_GA	*1_PRCPSP*	*GA* with the codification $\gamma = (\ell, \Omega)$ from 12.3.1, *Maxnint_serial SGS* instead of serial SGS and the two-point crossover introduced in 12.3.4 instead of the two-point crossover
1_DJRandom	*1_PRCPSP*	Randomly generates *nschedules*/3 codifications $\gamma = (\ell, \Omega)$ introduced in 12.3.1. The activity list ℓ contains each activity twice. The processing times in Ω are randomly generated. *Maxnint_serial SGS* and *Maxnint_DJ* are applied to each codification
1_DJGA	*1_PRCPSP*	*GA* with the codification $\gamma = (\ell, \Omega)$ from 12.3.1, *Maxnint_serial SGS* and *Maxnint_DJ* instead of serial SGS and the two-point crossover introduced in 12.3.4 instead of the two-point crossover
∞ *HGA*	*PRCPSP*	*HGA* applied to the transformed project in which each activity i is split into p_i subactivities of duration 1
EvoAlg	*Maxnint_PRCPSP*	Algorithm in Fig. 12.3 with the following parameters: $\sigma_{pop} = 50, \varphi = \lfloor \sigma_{pop}/4 \rfloor, \pi_{cros} = 0.75, perceninterrup = 1, \pi_{mut} = 0.05$, and $maxlife = 6$ except for the best individual for which $maxlife = \infty$

The *EvoAlg* algorithm shown in Fig. 12.3 works with a "population" of individuals (codifications of solutions). The size of the population is σ_{pop}, and φ is the number of children generated at each iteration. One child can be obtained by applying the *UnaryOperator* or by crossing two individuals to obtain a daughter by the operator *BeginEndOnePointCrossover*. With probability π_{cros} we apply the crossover. Each individual stays in the population *maxlife* iterations, except the current best individual that never dies. This concept was defined as life span

```
1. for [$i := 1, 4\sigma_{pop}$]
    1.1 $\gamma :=$ CalculateInitialCodification
    1.2 $S :=$ Maxnint_SerialSGS($\gamma$)
    1.3 $S' :=$ Maxnint_DJ ($S$)
    1.4 $\gamma' :=$ Representation of $S'$
2. Create a population POP with the best $\sigma_{pop}$ codifications $\gamma'$
3. until the stopping criteria are met:
    3.1 Children $:= \varnothing$
    3.2 for [$i := 1, \varphi$]
        3.2.1 Draw a uniformly distributed random number $u \in [0,1]$
        3.2.2 if($u \le \pi_{cros}$)
            3.2.2.1 Select two individuals $\gamma_1$ and $\gamma_2$ from the population
            3.2.2.2 $\gamma :=$ BeginEndOnePointCrossover ($\gamma_1, \gamma_2$)
        3.2.3 else
            3.2.3.1 Select an individual $\gamma_1$ from the population
            3.2.3.2 $\gamma :=$ UnaryOperator($\gamma_1$)
        3.2.4 $S :=$ Maxnint_SerialSGS($\gamma$)
        3.2.5 $S' :=$ Maxnint_DJ ($S$)
        3.2.6 $\gamma' :=$ Representation of $S'$
        3.2.7 Children $:=$ Children $\cup \{\gamma'\}$
    3.3 Increase the life span of every individual of the population by 1
    3.4 Eliminate from POP the individuals with lifespan $:=$ maxlife
    3.5 POP $:=$ Best $\sigma_{pop}$ individuals of POP $\cup$ Children
4. The outcome is the individual with the best fitness
```

Fig. 12.3 Outline of algorithm *EvoAlg*

(Michalewicz 1994). The procedure to generate the initial population, *CalculateInitialCodification*, is explained later.

CalculateInitialCodification is a procedure that iteratively generates a codification $\gamma = (\ell, \Omega)$. While a new codification is being generated, the procedure maintains the set *Elig* that contains the activities that are not totally introduced in ℓ and whose predecessors have already been totally introduced in ℓ (an activity i is totally introduced in ℓ if the sum of the durations of Ω corresponding to i coincides with p_i). At iteration u, the procedure selects an activity i ($\ell(u) = i$) from the set *Elig* and randomly generates $\Omega(u)$ in such a way that some conditions are fulfilled. To select the activity i at iteration u, the regret-based biased random sampling method together with the Latest Finish Time priority rule is employed. The latest finish times of the activities are computed on the original graph (V, E). To assign $\Omega(u)$, the procedure calculates $leftdur_i$. If activity i has already been interrupted $maxnint_i$ times, or if $leftdur_i$ is less than $2 \cdot \varepsilon_i$, then it cannot be further interrupted and $\Omega(u) = leftdur_i$. Otherwise, a random number between 0 and 1 is generated. If this number is less than a parameter *perceninterrup*, then activity i is not interrupted and $\Omega(u) = leftdur_i$. If the random number is greater than or equal to *perceninterrup*, then $\Omega(u)$ is calculated as a random integer number between ε_i and $leftdur_i - \varepsilon_i$. Each time an activity i is totally introduced in ℓ ($leftdur_i = 0$), *Elig* is updated. The procedure ends when *Elig* is empty. This procedure builds a codification γ in *totalnsub* iterations, where *totalnsub* is unknown until the end of the procedure. Note that the designed procedure is able to generate every possible codification for the problem.

12.6 Computational Results

In this section we present the results of computational studies concerning the algorithms introduced in the Sect. 12.5. As test instances, we have used the standard sets j30 and j120 for the RCPSP. They were generated using PROGEN (Kolisch et al. 1995) under a full factorial experimental design with the following three independent problem parameters: network complexity (NC), resource factor (RF), and resource strength (RS). The set j120 (j30) consists of 600 (480) projects with 120 (30) non-dummy activities. Both sets require four resource types. Further details of these problem instances are given in Kolisch et al. (1995). They are available in the Project Scheduling Library (PSPLIB: http://www.om-db.wi.tum.de/psplib/main.html) along with their optimum or the best of all the known makespans. Among the four RCPSP instance sets in PSPLIB, we have selected the j30 set because it is the only set for which the optimal solution is known for all the instances in the set. j120 is the set with the largest and most difficult projects in the library. To use these sets of instances for the *ρ_PRCPSP* it is only necessary to add the possibility of interrupting each activity a maximum of ρ times. Throughout the text we will not consider the 120 instances of j30 with RS $=$ 1, since they are trivial both for the *RCPSP* and the *ρ_PRCPSP* (the *ES* is feasible and then optimal).

In the *RCPSP*, it is a common practice (see, for example Hartmann and Kolisch 2000) to compare heuristic algorithms by limiting the maximum number of evaluated schedules. The most usual number is 5,000 (*nschedules* $=$ 5,000), although several other limits have been considered. Throughout this section we will use the following measures of the quality of an algorithm. We will denote by *CP_dev* the average percentage deviation of an algorithm from the critical path makespan, which is a lower bound for the *RCPSP* whether or not preemption is allowed.

First of all, we have made comparisons when solving three particular cases of the *Maxnint_PRCPSP* (*RCPSP*, *1_PRCPSP*, and *PRCPSP*). We have considered three algorithms to solve the two first problems and two algorithms to solve the *PRCPSP*, with a maximum of 5,000 generated schedules. Table 12.2 shows the different results for j30 in terms of deviation with respect to the optimal solution and for j120 in terms of deviation with respect to the CPM (*CP_dev*).

Table 12.2 Computational results on the sets j30 and j120

						EvoAlg	∞ *HGA*	*EvoAlg*
nschedules = 5,000	*Random*	*DJGA*	*HGA*	*1_Random*	*1_DJGA*	*(1_PRCPSP)*	*(PRCPSP)*	*(PRCPSP)*
j30 (*opt_dev*)	0.95	0.2	0.06	0.2	−1.04	−1.45		−1.55
j120 (*CPM_dev*)	47.51	33.24	32.54	43.74	30.35	29.91	29.51	29.03

Table 12.3 *CP_dev* for different algorithms with different limits on the number of schedules and j120

				EvoAlg	∞*HGA*	*EvoAlg*
nschedules	*DJGA*	*HGA*	*1_DJGA*	(*1_PRCPSP*)	(*PRCPSP*)	(*PRCPSP*)
5,000	33.24	32.54	30.35	29.91	29.51	29.03
10,000	32.75	32.04	29.78	29.53	29.21	28.68
25,000	31.98	31.52	29.34	29.02	28.86	28.29
100,000	31.10	30.95	28.71	28.41	28.57	27.74

It can be observed that there is an important difference between the algorithms with and without interruption, with better results when preemption is allowed, especially in j120. In this set, there is a difference of 3.77 % between the random solutions in the *1_PRCPSP* and in the *RCPSP*. So, the data indicates that the search space in the *1_RCPSP* is of better quality than that of the RCPSP. There is also an important difference between *DJGA* and *1_DJGA*. *DJGA* and *HGA* are two of the best heuristics for the *RCPSP*, and the difference between *HGA* and *1_DJGA* is 2.19 %. *EvoAlg* outperforms the rest of the algorithms in the *1_PRCPSP* and the *PRCPSP*.

Table 12.3 shows the performance of the two best algorithms in each of the three problems, varying the number of schedules generated and without taking the 152 instances where the solution of *HGA* coincides with the critical path length. In these cases no improvement is possible.

As we can see, *EvoAlg* is capable of calculating better solutions if more schedules are evaluated and *EvoAlg* outperforms the rest of the algorithms in the *1_PRCPSP* in all cases. It also outperforms *HGA* in the *PRCPSP* even more clearly. *EvoAlg* with 5,000, 10,000, and 25,000 obtains better results than *HGA* with 10,000, 25,000, and 100,000 schedules, respectively. It seems that specific algorithms for the *PRCPSP* are clearly better than algorithms for the *RCPSP* used for this purpose.

To study the behavior of *EvoAlg* when different number of interruptions are allowed, Ballestín et al. (2009) tested *EvoAlg* in j120 in the ρ*_PRCPSP* with $\rho = 1, 2, 3$, and unlimited (*PRCPSP*) with a limit of 10,000, 25,000, and 100,000 schedules. All versions have been set the same parameters, except for the number of individuals in the population. The results are shown in Table 12.4(a)–(c) without taking into account the 152 instances where the solution of *HGA* coincides with the critical path length. The first column indicates the limit on the number of schedules. Table 12.4(a) contains the average of *CP_dev*. The average and maximum

Table 12.4 (a) *CP_dev* depends on the number of schedules calculated (j120), (b) average improvement with respect to *HGA* (j120), and (c) maximum average improvement with respect to *HGA* (j120)

nschedules	0	1	2	3	Unlimited
(a)					
5,000	32.54	29.91	29.30	29.12	29.03
10,000	32.04	29.53	28.92	28.76	28.68
25,000	31.52	29.02	28.48	28.33	28.29
100,000	30.95	28.41	27.95	27.83	27.74
(b)					
5,000		1.85	2.26	2.38	2.45
10,000		1.78	2.18	2.29	2.35
25,000		1.77	2.12	2.21	2.25
100,000		1.81	2.11	2.19	2.25
(c)					
5,000		8.26	9.93	9.15	9.82
10,000		7.34	7.34	8.33	8.72
25,000		7.34	7.59	9.68	9.68
100,000		7.34	9.35	8.13	8.33

improvement with respect to *HGA* are shown in Table 12.4(b) and (c) respectively. The authors define the improvement obtained by the algorithm in the instance *ins* as $(HGA(ins) - Alg(ins))/Alg(ins)$, where $HGA(ins)$ and $Alg(ins)$ are the solutions obtained in the instance *ins* by *HGA* and *Alg* respectively.

The results show that the algorithm obtains better results in all versions when the number of generated schedules increases. As far as the difference between interrupting activities or not, we can see that the difference decreases, although it seems to stabilize and is still between 2.45 and 2.25 % in 1,00,000. The maximum improvement does not decrease and the average improvement does not decrease much either. Taking everything into account, preemption also seems to be useful when many schedules are calculated.

EvoAlg applies double justification to each schedule generated. The goodness of the double justification in the *RCPSP* was proven in Valls et al. (2005) where the authors affirmed that this technique greatly improves the quality of the solution obtained, without having to calculate more schedules. Ballestín et al. (2008) found that statement was also valid in the case of interruption being allowed, by comparing the solutions obtained by the algorithms Random, *DJRandom*, *GA*, and *DJGA* in set j30 and j120. All algorithms in both problems and instance sets perform better with *DJ* than without it. Concerning the problem *1_PRCPSP*, double justification improves both algorithms in both sets, and the improvement is greater than in the *RCPSP*.

12.7 Time and Work Generalized Precedence Relationship with Integer Preemption

When activity preemption is not allowed and tasks have fixed durations and fixed resource requirements per period, the lag of a GPR can equivalently be expressed in terms of elapsed time or in terms of work content carried out. However, both terms are not equivalent when preemption is allowed. Quintanilla et al. (2012) presented a complete study of work and time GPRs which includes proper definitions, a new notation and all possible conversions among them.

When lags are expressed in terms of elapsed periods, the authors speak of time GPRs and when lags are expressed in terms of percentage work content they speak of work GPRs. A generalized precedence relationship can be represented by means of four parameters: $GPR_{ij} = (T, \varphi_i, \varphi_j, wt)$. T indicates the relationship type $(SS_{ij}^{min}, SF_{ij}^{min}, FS_{ij}^{min}, FF_{ij}^{min}, SS_{ij}^{max}, SF_{ij}^{max}, FS_{ij}^{max}, FF_{ij}^{max})$. If GPR_{ij} is a work relationship, then $wt = w$ and φ_i and φ_j are integer numbers between 0 and 100 indicating percentages. If GPR_{ij} is a time relationship, then $wt = t$ and φ_i and φ_j are integer numbers indicating time units.

Although we could think of 16 GPRs, 8 time and 8 work GPRs, it can be shown that a maximal GPR (work or time) can be converted into an opposite direction minimal GPR and vice versa, then we can consider only 8 different GPRs in total.

Next, we present the definitions for the minimal work GPRs, the corresponding inequality and the equivalence with the maximal work GPRs, where $t_i^s(\varphi_i)$ is defined as the minimum time instant at which $\varphi_i\%$ of task i is completed and $t_i^f(\varphi_i)$ as the maximum time instant at which $\varphi_i\%$ of task i still has to be completed:

- $(SS_{ij}^{min}, \varphi_i, \varphi_j, w)$ means that the initial $\varphi_j\%$ of task j can be completed only if the initial $\varphi_i\%$ of task i has been completed $\leftrightarrow$ $t_j^s(\varphi_j) \geq t_i^s(\varphi_i)$ $\leftrightarrow$ $(SS_{ji}^{max}, \varphi_j, \varphi_i, w)$ $\leftrightarrow$ the initial $\varphi_i\%$ of task i must be finished at the completion of the initial $\varphi_j\%$ of task j.
- $(SF_{ij}^{min}, \varphi_i, \varphi_j, w)$ means that the process of the final $\varphi_j\%$ of task j can be started only if the initial $\varphi_i\%$ of task i has been completed $\leftrightarrow$ $t_j^f(\varphi_j) \geq t_i^s(\varphi_i)$ $\leftrightarrow$ $(FS_{ji}^{max}, \varphi_j, \varphi_i, w)$ $\leftrightarrow$ the initial $\varphi_i\%$ of task i must be finished at the start of the process of the final $\varphi_j\%$ of task j.
- $(FS_{ij}^{min}, \varphi_j, \varphi_i, w)$ means that the initial $\varphi_j\%$ of task j can be completed only if the process of the final $\varphi_i\%$ of task i has been started $\leftrightarrow$ $t_j^s(\varphi_j) \geq t_i^f(\varphi_i)$ $\leftrightarrow$ $(SF_{ji}^{max}, \varphi_j, \varphi_i, w)$ $\leftrightarrow$ the process of the final $\varphi_i\%$ of task i must have started at the completion of the initial $\varphi_j\%$ of task j.
- $(FF_{ij}^{min}, \varphi_i, \varphi_j, w)$ means that the process of the final $\varphi_j\%$ of task j can be started only if the process of the final $\varphi_i\%$ of task i has been started $\leftrightarrow$ $t_j^f(\varphi_j) \geq t_i^f(\varphi_i)$ $\leftrightarrow$ $(FF_{ji}^{max}, \varphi_j, \varphi_i, w)$ $\leftrightarrow$ the process of the final $\varphi_i\%$ of task i must have started at the start of the process of the final $\varphi_j\%$ of task j.

On the other hand, when preemption is allowed, $t_i^s(\varphi_i)$ may be different from $t_i^f(100 - \varphi_i)$. Therefore, it is not true that a minimal work GPR of a given type can be converted into a minimal work GPR of another type.

Time GPRs are equivalent to the classical GPR that has been extensively studied (Bartusch et al. 1988, Demeulemeester and Herroelen 2002, Neumann et al. 2003). For example and following the introduced notation, $(SF_{ij}^{min}, \varphi_i, \varphi_j, t) \leftrightarrow C_j - \varphi_j \geq S_i + \varphi_i \leftrightarrow C_j \geq S_i + \varphi_i + \varphi_j \leftrightarrow (SF_{ij}^{min}, \varphi_i + \varphi_j, 0, \mathrm{t})$ that in classical notation is expressed as $SF_{ij}^{min} = \varphi_i + \varphi_j$. Any time GPR can be equivalently converted into a time GPR in which φ_i or φ_j are equal to 0.

As the distance $C_i - S_i$ is not a constant (due to the preemption), it is not true that a given work GPR, or time GPR, of a given type can always be converted into any other type.

12.8 Conclusions

In this chapter we have studied how to deal with integer preemption in project scheduling. Some preemption project scheduling problems previously introduced in the literature have been described and some procedures introduced for the *RCPSP* have been adapted to work with preemption. As well, some specific procedures for the *Maxnint_PRCPSP* are presented. Computational results show that the specific procedures outperform the adapted procedures. The generalized relationships when preemption is allowed and the time lag depends on the work content need a special treatment. A complete study of the different types of work GPRs with minimal and maximal work content is presented. Computational results show that preemption can be beneficial when trying to minimize C_{max}.

Acknowledgements This research was partially supported by Ministerio de Ciencia e Innovación, MTM2011-23546.

References

Ballestín F, Valls V, Quintanilla S (2008) Preemption in resource-constrained project scheduling. Eur J Oper Res 189:1636–1152

Ballestín F, Valls V, Quintanilla S (2009) Scheduling projects with limited number of preemptions. Comput Oper Res 36:2913–2925

Ballestín F, Barrios A, Valls V (2013) Looking for the best modes helps solving the MRCPSP/max. Int J Prod Res 51:813–827

Bartusch M, Möhring RH, Radermacher FJ (1988) Scheduling project networks with resource constraints and time windows. Ann Oper Res 16:201–240

Błażewicz J, Lenstra JK, RinooyKan AHG (1983) Scheduling subject to resource constraints: classification and complexity. Discrete Appl Math 5:11–24

Demeulemeester E, Herroelen W (1996) An efficient optimal procedure for the preemptive resource-constrained project scheduling problem. Eur J Oper Res 90:334–48

Demeulemeester E, Herroelen W (2002) Project scheduling: a research handbook. Kluwer Academic, Norwell

Hartmann, S (1998) A competitive genetic algorithm for resource-constrained project scheduling. Nav Res Logist 45:733–750

Hartmann S, Kolisch R (2000) Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. Eur J Oper Res 127:394–407

Kaplan LA (1988) Resource-constrained project scheduling with pre-emption of jobs. Unpublished Ph.D. dissertation, University of Michigan, Ann Arbor

Kaplan LA (1991) Resource-constrained project scheduling with setup times. Unpublished paper, Department of Management, University of Tennessee, Knoxville

Kolisch R, Hartmann S (2006) Experimental investigation of heuristics for resource-constrained project scheduling: an update. Eur J Oper Res 174(1):23–37

Kolisch R, Sprecher A, Drexl A (1995) Characterization and generation of a general class of resource-constrained project scheduling problems. Manag Sci 41:1693–703

Li F, Lai Ch, Shou Y (2011) Particle swarm optimization for preemptive project scheduling with resource constraints. In: Proceedings of the IEEM 2011. IEEE, Singapore, pp 869–871

Michalewicz Z (1994) Genetic algorithms + data structures = evolution programs. Springer, New York

Neumann K, Schwindt C, Zimmermann J (2003) Project scheduling with time windows and scarce resources. Springer, Berlin

Patterson JH (1984) A comparison of exact procedures for solving the multiple constrained resource project scheduling problem. Manag Sci 30(7):854–867

Quintanilla S, Pérez A, Lino P, Valls V (2012) Time and work generalised precedence relationships in project scheduling with pre-emption: an application to the management of service centres. Eur J Oper Res 219:59–72

Valls V, Ballestín F, Quintanilla S (2005) Justification and RCPSP: a technique that pays. Eur J Oper Res 165(2):375–386

Valls V, Ballestín F, Quintanilla S (2008) A hybrid genetic algorithm for the resource-constrained project scheduling problem. Eur J Oper Res 85(2):495–508

Van Peteghem V, Vanhoucke M (2010) A genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem. Eur J Oper Res 201:409–418

Chapter 13
Continuous Preemption Problems

Christoph Schwindt and Tobias Paetz

Abstract In this chapter we are concerned with project scheduling problems involving preemption of activities at arbitrary points in time. We survey the literature on preemptive project scheduling and propose a classification scheme for these problems. We then consider a project scheduling problem under continuous preemption, flexible resource allocation, and generalized feeding precedence relations between the activities. After providing a formal problem statement we reduce the problem to a canonical form only containing nonpositive completion-to-start time lags and investigate structural issues like necessary feasibility conditions and preemption gains. Next, we develop an MILP formulation that encodes a schedule as a sequence of slices containing sets of activities that are simultaneously in progress. Moreover, feasibility tests, preprocessing methods, and a column-generated based lower bound on the minimum project duration are presented. Finally, we report on the results of an experimental performance analysis of the MILP model for the project duration problem.

Keywords Column generation • Continuous preemption • Makespan minimization • Mixed-integer linear programming formulation • Project scheduling • Resource constraints

13.1 Introduction

This chapter is devoted to resource-constrained project scheduling problems in which the activities can be interrupted during their execution at any point in time. While there exists a considerable body of papers dealing with the case of job preemptions in machine and processor scheduling, preemptive resource-constrained project scheduling has received much less attention. On the other hand, there exist

C. Schwindt (✉) • T. Paetz
Institute of Management and Economics, Clausthal University of Technology,
Clausthal-Zellerfeld, Germany
e-mail: christoph.schwindt@tu-clausthal.de; tobias.paetz@tu-clausthal.de

C. Schwindt, J. Zimmermann (eds.), *Handbook on Project Management and Scheduling Vol. 1*, International Handbooks on Information Systems,
DOI 10.1007/978-3-319-05443-8_13

many practical scheduling problems in which resource units have to be allocated to divisible activities over time.

Examples of such applications include aggregate project planning, finite capacity scheduling, the remote dispatch of controllable electric appliances, or scheduling multi-processor tasks on parallel microprocessors of a computer system. When dealing with long-term undertakings like development or construction projects, it is generally expedient to start with a rough-cut planning model and to iteratively refine the project plan according to a *hierarchical project planning* approach (see Neumann et al. 2003, Sect. 3.10). At the higher planning levels, the projects are represented in an aggregate form where the activities to be scheduled represent subprojects or working packages. In such a project scheduling setting, there is commonly no need to assume that these aggregate activities must not be interrupted.

In production planning and control systems, *finite capacity scheduling* FCS receives time-phased production orders from the materials requirements planning (MRP) module and schedules these orders under resource constraints, before releasing them to the shop floor. Since MRP creates the production orders using methods for uncapacitated lot sizing, it is often necessary to split the orders on the FCS level to be able to meet the planned completion times of the orders. FCS problems can be modeled as resource-constrained project scheduling problems, the activities corresponding to the production orders (see, e.g., Franck et al. 1997).

An important objective of *short-term scheduling in smart grids* consists in the leveling of power demands over time. To reduce the load during peak usage periods, electric utility companies may switch off certain devices like electric heating facilities, air-conditioning plants, or cold storage houses, which can be controlled remotely via metering and communication technology. The heating or cooling tasks of these devices can be interpreted as interruptible activities, whose executions must be scheduled under temperature interval conditions.

In a multi-microprocessor computer system, a computing job requires one or more processors at a time. The scheduling of such jobs on the processors is termed *multi-processor tasks scheduling* in machine scheduling. When considering a central system with parallel (i. e., non-distributed) processors, communication times between the processors can be integrated into the processing time of the job and no communication delays need be considered. In certain applications, the jobs can be interrupted and resumed later at (nearly) no cost. Moreover, there may exist precedence relations between the jobs, e.g., if the jobs correspond to modules of a computer program. In such a setting, some modules may be processed concurrently on different processors, whereas other modules cannot be carried out in parallel since the output of one module is required as input of other modules. Various models and methods for multi-processor task scheduling can be found in Chap. 6 of the book by Błażewicz et al. (2007).

The remainder of the chapter is organized as follows. In Sect. 13.2 we review the literature on preemptive project scheduling problems. Section 13.3 is dedicated to a generic scheduling problem including continuous activity preemption and generalized feeding precedence relations between the activities. In Sect. 13.4 we study alternative representations of the problem, feasibility conditions, and potential preemption gains. An MILP formulation of polynomial size for the problem is proposed in Sect. 13.5, which also addresses the question of the maximum number of slices that are needed for encoding optimal schedules. In Sect. 13.6 we devise feasibility tests and preprocessing techniques, which can be used to reduce the CPU time requirements of commercial solvers on instances of the MILP model. Moreover, we explain how effective lower bounds on the minimum project duration can be obtained by solving a large-scale linear program via column generation. Section 13.7 presents the results of an experimental performance analysis of the model and the lower-bounding procedure for the minimum project duration problem. The chapter concludes with a brief summary and some remarks on future research directions in Sect. 13.8.

13.2 Literature Survey on Preemptive Project Scheduling

In this section we first develop a classification scheme for various preemptive resource-constrained project scheduling problems that have been studied in the literature. Next, we survey the different types of solution approaches that have been proposed for preemptive project scheduling.

13.2.1 Classification of Preemptive Project Scheduling Problems

Several variants of preemptive project scheduling problems with renewable resources have been studied in the literature. These variants can be classified according to attributes such as the type, the potential causes, and the maximum allowable number of preemptions, the flexibility of the resource allocation, and the type of precedence relations between the activities.

With respect to the *type of interruptions*, we may distinguish between the cases of integer, discrete, and continuous preemptions. Integer preemption, which is also referred to as integral preemption, assumes that interruptions can only occur at integral points in time. Remarkably, this problem setting has been a common (and mostly tacit) assumption in preemptive project scheduling for a long time. Integer preemption project scheduling problems are studied in Chap. 12 of this handbook. Project scheduling under continuous preemption, which allows activities to be suspended at any wanted moments, was already investigated in the seminal paper by Słowiński (1980). Only recently, however, the case of continuous preemption in project scheduling has been taken up again. Continuous preemption is sometimes also called rational preemption in the literature (see, e.g., Damay 2008). In a discrete preemption setting, the interruption of an activity is limited to a finite set of points in time at which the activity has attained one of the predefined progress portions.

In practice, there may exist various *causes for planned activity interruptions*. For example, an activity may be interrupted because some renewable resource becomes unavailable during a rest time or a planned maintenance operation. It is then generally assumed that the activity must be resumed as soon as the resource is available again. In the context of a scheduling problem, this kind of preemptions must be considered explicitly if not all resources are unavailable at the same time or if some activities must not be interrupted. Project scheduling problems with activity calenders have been considered by Franck et al. (2001b). An activity calendar specifies individual intervals for each activity during which it cannot be processed. A related problem setting in which each activity is associated with one of a given number of cyclic work/rest patterns has been studied by Yang and Chen (2000) and Vanhoucke et al. (2002) under the name of time-switch constraints (see also Chap. 30 of this handbook). Another example of a reason for suspending an activity is the release of some activity with higher priority. This case is referred to as selective preemption in scheduling (see Damay 2008). In the most general and most common setting, which we will call discretionary preemption in what follows, an activity can be interrupted for any reason. In particular, it may be necessary to consider discretionary preemption when seeking for a compact schedule. This point becomes immediately clear by considering the preemptive parallel machine problem $P2 \mid pmtn, p_j = 1 \mid C_{max}$ with three jobs of equal priority. The minimum makespan of 1.5 time units can only be attained by suspending the execution of some job at time 0.5 and resuming the same job at time 1.0 on the other machine.

In a preemptive scheduling problem, the *number of interruptions* per activity is usually not explicitly limited to some prescribed upper bound. As we will see later on, the number of interruptions required for obtaining an optimal preemptive schedule is often polynomially bounded in the number of activities. Since any interruption may incur additional cost, some authors have also studied preemptive project scheduling problems with an a-priori limit on the number of interruptions (see, e.g., Ballestín et al. 2008 or Li et al. 2011 for the case with at most one interruption per activity and Ballestín et al. 2009 or Chap. 12 of this handbook for the more general case where a limited number of interruptions are allowed for each activity).

The *allocation flexibility* specifies the way in which the reallocation of resource units can be performed when an activity is resumed after having been interrupted. In the more restrictive setting of an inflexible allocation, each activity must be carried out by the same resource units during the entire processing time. Otherwise, we speak of a flexible resource allocation. An inflexible resource allocation is generally encountered when dealing with human resources. A resource unit then represents a staff member and switching the resource unit after an interruption would require extensive briefing or instruction of the person pursuing the activity. A preemptive project scheduling problem with inflexible resource allocation is considered in Chap. 28 of this handbook. In the more common flexible allocation case, the resource units may be reallocated at no cost every time an activity is resumed. The example of the preemptive parallel machine scheduling problem mentioned above shows that in general a resource reallocation is necessary to obtain an optimal schedule.

Like in non-preemptive project scheduling, we may further distinguish between different *types of precedence relations* among the activities. Ordinary precedence relations ensure that an activity can only be started when all of its predecessors have been completed. An immediate generalization of this setting arises when we consider generalized precedence relations in the form of minimum and maximum time lags between the start or completion times of the activities. In difference to non-preemptive scheduling, however, the time lag between the start and the completion time of an activity is not given in advance. Consequently, a completion-to-start time lag between two activities cannot be transformed into an equivalent start-to-start time lag. That is why the concept of (ordinary) feeding precedence relations has been proposed by Kis (2005). A feeding precedence relation requires that an activity can only be started when the preceding activity has been executed to a given percentage. We may think of the preceding activity as being decomposed into two parts, the initial part representing the execution of the activity up to the specified percentage and the final part corresponding to the rest of the activity. Alfieria et al. (2011) considered three further types of feeding precedence relations, which enforce an activity to be started or to be completed, respectively, no later than the initial part of the preceding activity was finished or guarantee that an activity is not completed before the initial part of the preceding activity was finished (these types of precedence relations are also discussed in Chap. 56 in the second volume of this handbook). Recently, Quintanilla et al. (2012) introduced a more general type of constraints called generalized work relationships, which define ordinary precedence relations between the initial part of the preceding and the final part of the succeeding activity (see also Chap. 12 of this handbook). In Sect. 13.3 we will present the concept of generalized feeding precedence relations, which includes all of the above types of precedence relations that are applicable in the case of continuous preemption problems. A generalized feeding precedence relation between two activities defines a minimum or a maximum time lag between the initial and final parts of two activities. As we will show in Sect. 13.4, generalized feeding precedence relations can always be reduced to (possibly negative) minimum completion-to-start time lags by representing the individual parts as independent activities.

Table 13.1 summarizes the classification of preemptive project scheduling problems discussed in this section, the characteristics of the problem considered in this chapter being set in italics.

13.2.2 Solution Approaches to Preemptive Project Scheduling Problems

The earliest reference to a preemptive project scheduling problem we are aware of is the paper by Słowiński (1980). In this work, besides further types of scheduling problems, the preemptive project scheduling problem $PS \mid prec, pmtn \mid C_{max}$

Table 13.1 Classification of preemptive project scheduling problems

Attribute	Characteristics
Type of preemptions	Integer (integral) Discrete *Continuous* (*rational*)
Causes of preemptions	Calendars Selective *Discretionary*
Number of preemptions	Limited *Unlimited*
Resource allocation	Inflexible *Flexible*
Type of precedence relations	Ordinary precedence relations Ordinary feeding relations Generalized work relationships *Generalized feeding precedence relations*

with continuous preemption, flexible resource allocation, and ordinary precedence relations is solved based on a large-scale linear program containing a nonnegative duration variable for each set of activities that can be executed in parallel. In what follows, we will refer to such a set of activities as a feasible antichain of the precedence order. The linear program can be solved with the revised simplex algorithm. However, the linear program must be set up with only a subset of all feasible antichains, otherwise the solution of the linear program may violate some transitive precedence relations. That is why Słowiński suggests to generate the subset of antichains using a heuristic method presented in Słowiński (1978). The linear program containing a decision variable for each feasible antichain constitutes a relaxation of the problem in which the (direct and transitive) precedence relations are replaced by disjunctive constraints preventing the simultaneous execution of precedence-related activities. We will refer to this problem as the disjunctive relaxation.

The general idea of Słowiński's problem formulation was readopted by Mingozzi et al. (1998), Brucker and Knust (2000), and Baptiste and Demassey (2004) for computing lower bounds on the minimum duration of non-preemptive projects by solving the disjunctive relaxation (see Chap. 3 of this handbook). The same formulation is also the starting point of an exact branch-and-bound algorithm for $PS \mid prec, pmtn \mid C_{max}$, which has been proposed by Damay et al. in 2007. The main idea of the method consists in solving the linear program with a column-generation approach, which will be discussed in Sect. 13.6.4. If the resulting solution is feasible with respect to the precedence relations, any sequence of the active antichains respecting the precedence relations gives rise to an optimal solution. Otherwise, the solution induces an oriented cycle in a directed graph containing all active antichains as nodes and the precedence relations among the active antichains as arcs. In this case, the algorithm selects one $\subseteq$-minimal directed cycle and branches over all alternatives to break the cycle by forbidding the parallel execution of two activities jointly contained in one of the antichains on the cycle. In addition to the branch-and-bound procedure, Damay et al. (2007) also develop an order-theoretical characterization of basic solutions to Słowiński's linear program that can be translated into a precedence-feasible sequence of antichains. Furthermore, they devise a heuristic descent and neighborhood search methods for a generalized version of the problem in which some but not necessarily all activities may be interrupted. These methods rely on Słowiński's LP formulation of the disjunctive relaxation.

The increase in problem complexity that is due to the possibility of splitting activities is considerably reduced when only integer preemptions are permitted. Integer preemption problems of type $PS \mid prec, pmtn/int \mid f$ are amenable to classical time-indexed MILP formulations, which, e.g., were proposed in the doctoral dissertation of Kaplan (1988) for the project duration $f = C_{max}$. A combinatorial branch-and-bound algorithm for the case of integer preemptions was presented by Demeulemeester and Herroelen (1996). The basic idea consists in decomposing each activity into a sequence of unit-duration subactivities. The resulting project scheduling problem is then solved with an adapted version of a branch-and-bound algorithm for the non-preemptive case. Vanhoucke (2008) followed a similar approach for an integer preemption problem with setup times, in which the duration of an activity may also be shortened by processing several parts of the activity in parallel (fast-tracking). Nadjafi and Shadrokh (2008) presented a time-indexed MILP for the integer preemption problem with activity due dates and preemption costs. The objective function is composed of earliness and tardiness costs for the completion of the activities as well as costs incurred for the interruption of the activities. Similarly to the approaches by Demeulemeester and Herroelen and Vanhoucke, the decision variables of the model refer to the subactivities that arise from splitting the activities into unit-duration tasks.

Several authors proposed adaptations of priority-rule based methods to the case of activity preemptions (see, e.g., Richter and Yano 1986 and Kaplan 1988). Due to the way in which the serial and the parallel schedule-generation schemes dispatch the activities, these methods are implicitly dedicated to the case of an inflexible resource allocation and may systematically exclude optimal schedules. As already noticed by Damay (2008), it seems difficult to obtain optimal solutions to continuous preemption problems with flexible resource allocation by applying iterative schedule-construction procedures since the interruption of an activity is not necessarily caused by the completion or the release of another activity.

Bianco et al. (1999) considered the variant $PSm, 1 \mid prec, pmtn/int \mid C_{max}$ of the integer preemption problem with unary resources. They show that the problem is equivalent to a precedence-constrained weighted vertex coloring problem of the disjunctive activity-on-node graph in which incompatible activities are connected by edges. Each color corresponds to a unit time period, and the problem consists in assigning each node a set of color indices such that the number of allotted colors coincides with its duration, the color sets of incompatible activities are disjoint, the minimum color index of an activity is greater than the maximum color index of any of its predecessors, and a minimum number of different colors is needed. For solving this coloring problem, the authors devise an adapted version of a branch-and-bound algorithm for the weighted vertex coloring problem.

Integer preemption problems with a limited number of interruptions per activity were studied by Ballestín et al. (see also Chap. 12 of this handbook). In a paper of 2008 they focus on the problem in which each activity can be interrupted at most once. The serial schedule-generation scheme and the so-called double-justification technique are adapted to accommodate this problem setting. Double justification is a schedule-improvement procedure which starting from some feasible schedule performs backward and then forward scheduling of the activities in nonincreasing order of completion times and nondecreasing order of start times, respectively. In Ballestín et al. (2009) the general case of a given maximum number i of activity interruptions is investigated. Each activity is decomposed into a series of $i + 1$ subactivities representing the non-preemptive execution of a part of the activity. In difference to the approach by Demeulemeester and Herroelen (1996), the durations of the subactivities are not given in advance but are integer decision variables, bounded from below by some given minimum execution time during which no interruption is allowed. The problem is solved with an evolutionary algorithm encoding the individuals by pairs of a subactivity list and an associated duration vector and decoding the individuals into schedules by means of an adapted version of the serial schedule-generation scheme.

The multi-mode project scheduling problem $MPS \mid prec, pmtn \mid C_{max}$ with continuous preemptions and flexible mode assignment was already investigated by Słowiński (1980). The linear programming formulation of the disjunctive relaxation can easily be generalized to the case of alternative execution modes and limited nonrenewable resources. Based on this formulation, good feasible schedules can be generated in a similar way to the single-mode problem.

Several authors dealt with multi-mode integer preemption problems with inflexible mode assignment. In difference to the problem studied by Słowiński, they assume that the mode assignment is inflexible, i. e., that an activity must be resumed in the execution mode in which it had been processed before. Nudtasomboon and Randhawa (1997) presented a time-indexed binary linear programming model for the minimum duration, the minimum execution cost, and a resource leveling objective function. Moreover, they devise a goal programming formulation for the multi-criteria variant of the problem. The developed branch-and-bound methods are based on the principles of an implicit enumeration algorithm for the non-preemptive multi-mode project scheduling problem $MPS \mid prec \mid C_{max}$ by Talbot and an implicit enumeration method for binary goal programming, respectively. Van Peteghem and Vanhoucke (2010) present a bi-population genetic algorithm for the project duration version of the problem, which combines the serial schedule-generation scheme with a mode improvement procedure. Like in the approach by Demeulemeester and Herroelen (1996), the option to interrupt activities at integral points in time is represented by splitting each activity into a sequence of unit-duration subactivities.

A variant of the problem with renewable resources of time-varying capacities is discussed in Buddhakulsomsiri and Kim (2006). In addition to a time-indexed MILP problem formulation, the authors describe a branch-and-bound algorithm adapted from the precedence tree algorithm by Hartmann and Drexl (1998) for the non-preemptive problem. An extensive experimental performance analysis of the algorithm shows that activity preemption may significantly reduce the minimum project duration when resources are not permanently available due to vacations or scheduled downtimes. Buddhakulsomsiri and Kim (2007) proposed a priority-rule based method for the same problem, which is based on the concept of a moving resource strength. In each iteration of the schedule-generation scheme, the moving resource strength is computed as the ratio of the mean and the maximum resource capacity that is available in the time interval between the minimum earliest start time and the minimum latest start time of some eligible activity. A small value of the ratio indicates that the splitting of the activity to be scheduled is encouraged.

13.3 The Preemptive Resource-Constrained Project Scheduling Problem With Generalized Feeding Precedence Relations

The balance of this chapter is devoted to a generic resource-constrained project scheduling problem, which includes the cases of preemptive and non-preemptive activities, continuous, discrete, and integer preemptions, as well as ordinary feeding relations and generalized work relationships. In Sect. 13.3.1 we define the problem under consideration and provide a descriptive model. In Sect. 13.3.2 we elaborate on the semantic power of the problem setting and explain how non-preemptive activities and the different types of preemptions and precedence relations can be represented within this framework.

13.3.1 Problem Definition and Descriptive Model

We consider a project comprising a set V of n activities i with deterministic durations $p_i \in \mathbb{Z}_{\geq 0}$. To execute the activities, a set $\mathscr{R}$ of renewable resources k with capacities $R_k \in \mathbb{N}$ is available. During its execution, each activity i requires $r_{ik} \in \mathbb{Z}_{\geq 0}$ units of resources $k \in \mathscr{R}$. According to the continuous preemption and flexible resource allocation settings, the processing of an activity may be stopped at any point in time and resumed later on using an arbitrary set of r_{ik} available units of each resource k. A solution to the scheduling problem can be specified as a trajectory $x : t \mapsto (x_i(t))_{i \in V}$, where $x_i(t) \in [0, 1]$ denotes the percentage of activity i that has been processed by time $t \geq 0$. In what follows, we will refer to this percentage as the relative progress of activity i at time t. For activities $i \in V$

with $p_i = 0$ representing project events we adopt the convention that $x_i(t) = 1$ at the occurrence time of i. The project is completed when all activities i have been executed with their respective durations p_i, i. e., when $x_i(t) = 1$ for all $i \in V$.

Obviously, it holds that $0 \leq x_i(t + \Delta t) - x_i(t) \leq \Delta t / p_i$ for any $t \geq 0$ and any $\Delta t \geq 0$ if $p_i > 0$. Consequently, the right-hand derivative

$$\frac{\mathrm{d}^+ x_i}{\mathrm{d}t}(t) = \lim_{\Delta t \downarrow 0} \frac{x_i(t + \Delta t) - x_i(t)}{\Delta t} \begin{cases} \geq 0 \text{ and} \\ \leq \lim_{\Delta t \downarrow 0} \frac{\Delta t / p_i}{\Delta t} = \frac{1}{p_i} \end{cases}$$

of function $x_i(t)$ exists for each real activity $i \in V$ and any $t \geq 0$. As it has been shown by Baptiste et al. for a more general scheduling problem, each feasible instance of the problem admits a solution with a finite number of activity interruptions (Lemma 2 in Baptiste et al. 2004 or Theorem 3.4 in Baptiste et al. 2011). That is why for each $i \in V$ with $p_i > 0$, function $x_i(t)$ is piecewise linear and the right-continuous function

$$y_i(t) := p_i \cdot \frac{\mathrm{d}^+ x_i}{\mathrm{d}t}(t)$$

equals one precisely if activity i is in progress at time t and zero, otherwise. Hence, the resource constraints of the scheduling problem can be stated as

$$\sum_{i \in V'} r_{ik} y_i(t) \leq R_k \quad (k \in \mathscr{R};\ t \geq 0) \tag{13.1}$$

where $V' := \{i \in V \mid p_i > 0\}$. For events $i \in V$ we define $y_i(t) := 1$ if $t = \min\{t' \mid x_i(t') = 1\}$ and $y_i(t) := 0$, otherwise.

We assume that for certain pairs (i, j) of activities $i, j \in V$, a generalized feeding precedence relation $\Delta_{ij} = (\xi_i, \xi_j, \delta_{ij})$ with $\xi_i \in]0, 1] \cap \mathbb{Q}$ and $\xi_j \in [0, 1[\cap \mathbb{Q}$ must be observed. Precedence relation Δ_{ij} requires that the final portion $1 - \xi_j$ of activity j can be started δ_{ij} time units after activity i attained relative progress ξ_i at the earliest. Let

$$t_i^-(\xi) := \min\{t \mid x_i(t) \geq \xi\} \quad (0 < \xi \leq 1) \text{ and}$$
$$t_i^+(\xi) := \sup\{t \mid x_i(t) \leq \xi\} \quad (0 \leq \xi < 1)$$

be the earliest and latest times $t \geq 0$, respectively, at which activity i has been executed to portion ξ. The reason why the supremum is needed in the definition of $t_i^+(\xi)$ is that by the above convention, function $x_i(t)$ is a right-continuous step function for events i. We note that the start and completion times S_i and C_i of activity i can be expressed as $S_i = t_i^+(0)$ and $C_i = t_i^-(1)$, where $S_i = C_i$ for events $i \in V$.

Let $E \subseteq V \times V$ be the set of activity pairs (i, j) for which a precedence relation Δ_{ij} has been specified. The generalized feeding precedence relations are stated as the inequalities

$$t_j^+(\xi_j) \geq t_i^-(\xi_i) + \delta_{ij} \quad ((i,j) \in E) \tag{13.2}$$

If time lag δ_{ij} is negative, $-\delta_{ij} > 0$ can be interpreted as a maximum time lag between time points $t_j^+(\xi_j)$ and $t_i^-(\xi_i)$. Since $\sup\{t \mid x_j(t) \leq \xi\} = \inf\{t \mid x_j(t) > \xi\}$ for all $\xi < 1$, the left-hand side $t_j^+(\xi_j)$ of constraint (13.2) equals the *earliest* point in time at which the relative progress of j exceeds ξ_j.

Furthermore, we point out that precedence relations of type $\Delta_{ij} = (0, \xi_j, \delta_{ij})$ and $\Delta_{ij} = (\xi_i, 1, \delta_{ij})$ cannot be established, given that $t_i^-(0)$ and $t_i^+(1)$ have not been defined. These precedence relations, however, would not be meaningful because they could be satisfied by interrupting activity i immediately after its start or interrupting activity j immediately before its completion, respectively. More generally, constraints that arise from replacing function t_j^+ by t_j^- or function t_i^- by t_i^+ in inequality (13.2) are not purposeful either in the context of continuous preemption. For example, the constraint $t_j^-(\xi_j) \geq t$ for some right-hand side t says that the initial portion ξ_j of activity j must not be completed before time t. In case of continuous preemption, portion $\xi_j - \varepsilon$ of activity i with arbitrarily small $\varepsilon > 0$ can still be completed before time t, rendering the feasible region of the scheduling problem non-closed and hence the scheduling problem possibly infeasible even when the feasible region is nonempty. A similar reasoning applies to constraints of type $t_i^+(\xi_i) \leq t$. These observations also become clear by taking into account that inequality (13.2) defines a minimum time lag δ_{ij} between the completion of the initial portion ξ_i of activity i and the start of the final portion $1 - \xi_j$ of activity j. By altering the superscripts $+$ or $-$, one would define start-to-start, start-to-completion, and completion-to-completion relations between the two parts of activities i and j. As we have seen before, these relations are not meaningful when (parts of) activities can be continuously interrupted.

Example 13.1. We consider a precedence relation $\Delta_{ij} = (0.25, 0.4, 3)$ between two preemptive activities i and j with durations $p_i = 4$ and $p_j = 5$. Suppose that activity i is executed in time intervals $[0, 1[$ and $[3, 6[$ and activity j is in progress in intervals $[0, 2[$ and $[4, 7[$. Figure 13.1 shows the corresponding Gantt chart and the graphs of functions x_i and x_j. Activity i attains relative progress $\xi_i = 0.25$ at time $t = t_i^-(\xi_i) = 1$, and the latest point in time at which the relative progress of j equals $\xi_j = 0.4$ is $t = t_j^+(\xi_j) = 4$. Since $t_j^+(\xi_j) = 4 \geq t_i^-(\xi) + \delta_{ij} = 1 + 3$, the schedule for activities i and j satisfies the precedence relation.

We show that inequality (13.2) models the requirement that earlier than δ_{ij} time units after the percentage ξ_i of activity i has been completed, the relative progress of activity j must not exceed ξ_j. This can be seen as follows, the third equivalence following from the nondecreasing property of function x_j.

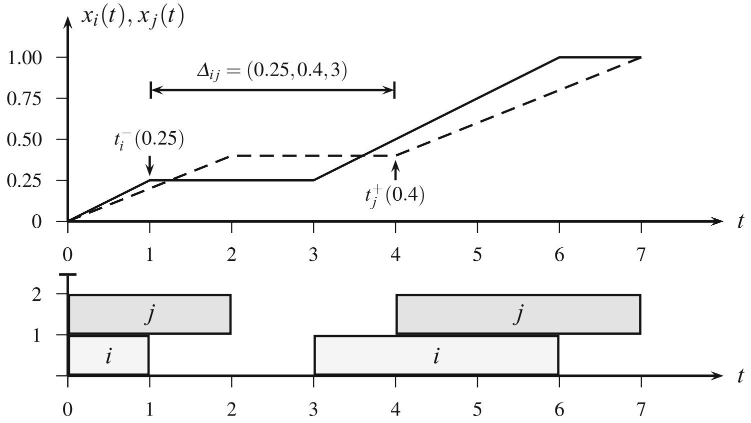


Fig. 13.1 Definition of generalized feeding precedence relations

$$
\begin{aligned}
& t_j^+(\xi_j) \geq t_i^-(\xi_i) + \delta_{ij} \\
\Leftrightarrow\ & \sup\{t \mid x_j(t) \leq \xi_j\} \geq \min\{t \mid x_i(t) \geq \xi_i\} + \delta_{ij} \\
\Leftrightarrow\ & \inf\{t \mid x_j(t) > \xi_j\} \geq \min\{t \mid x_i(t) \geq \xi_i\} + \delta_{ij} \\
\Leftrightarrow\ & (x_j(t') > \xi_j \Rightarrow t' \geq \min\{t \mid x_i(t) \geq \xi_i\} + \delta_{ij}) \\
\Leftrightarrow\ & (t' < \min\{t \mid x_i(t) \geq \xi_i\} + \delta_{ij} \Rightarrow x_j(t') \leq \xi_j)
\end{aligned}
$$

Let $f(x)$ denote the objective function value of solution x for the scheduling problem under consideration. For example, f may be chosen to be the project duration or makespan $C_{max} = \max_{i \in V} C_i$. The resulting resource-constrained project scheduling problem $PS \mid temp, feed, pmtn \mid f$ with continuous preemptions, flexible resource allocation, and generalized feeding precedence relations can be written as

$$
(P) \begin{cases} \text{Min.} & f(x) \\ \text{s. t.} & \text{Eqs. (13.1) and (13.2)} \end{cases}
$$

As we will see later on, problem (P) also includes the case of non-preemptive activities. Hence, (P) is a generalization of the non-preemptive project scheduling problem $PS \mid temp \mid f$, for which the feasibility variant is known to be strongly $\mathcal{NP}$-complete.

In project scheduling with generalized precedence relations, it is customary to represent the project as an activity-on-node network $N = (V, E, \Delta)$ with node set V, arc set E, and arc weights Δ. For notational convenience it is assumed that in addition to the n project activities, set V also contains a project beginning

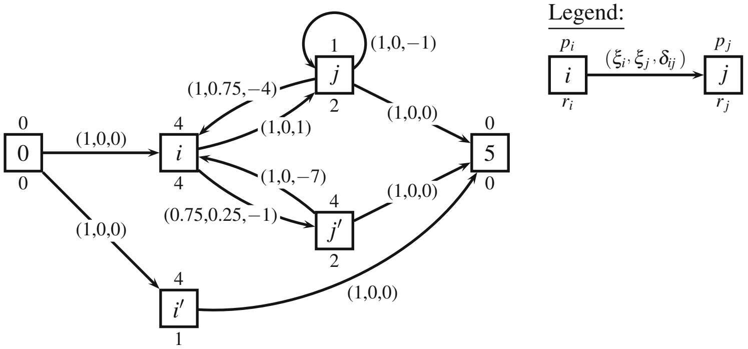


Fig. 13.2 Example project with four real activities and one resource with capacity $R = 4$

event $i = 0$ with $S_i = 0$ and a project termination event $i = n + 1$ with $C_{n+1} = C_{max}$. Network N contains an arc for each pair (i, j) of activities i and j for which a precedence relation Δ_{ij} has been defined. In difference to the case of non-preemptive scheduling, set E may also contain loops and parallel arcs. A loop $(i, i) \in E$ with $\Delta_{ii} = (\xi_i^1, \xi_i^2, \delta_{ii})$ defines a prescribed lag of δ_{ii} time units between the moments when activity i reaches relative progress ξ_i^1 and starts being processed beyond percentage ξ_i^2. For example, if $\xi_i^1 = 1$ and $\xi_i^2 = 0$, the precedence relation implies $C_i - S_i \leq -\delta_{ii}$, which means that activity i must not be interrupted longer than $-\delta_{ii} - p_i$ time units. In particular, if $\delta_{ii} = -p_i$, activity i is non-preemptive. Parallel arcs $(i, j)^1$ and $(i, j)^2$ belonging to different precedence relations Δ_{ij}^1 and Δ_{ij}^2 may also be meaningful since in general the corresponding inequalities cannot be reduced to a single constraint of type (13.2). In Sect. 13.4.1 we will develop a canonical representation of our problem for which without loss of generality we may assume that the project network is a simple digraph with loops but without parallel arcs.

Example 13.2. Consider the project with four real activities i, i', j, j' and a single renewable resource of capacity $R = 4$ represented by the activity-on-node network N depicted in Fig. 13.2. The arcs emanating from node 0 ensure that no activity is started before the project beginning, and the arcs leading to node 5 guarantee that all activities are terminated when the project is completed. The oriented cycle containing nodes i and j expresses the requirement that activity j can only be started one time unit after the completion of activity i and that activity j must be completed four time units after the last quarter of activity i was started. The loop (j, j) prevents activity j from being interrupted.

Figure 13.3 shows a schedule for the execution of the four activities with the minimum makespan $C_{max} = 8.5$. The example illustrates that even though all activity durations p_i and all durations $p_i \cdot \xi_i$ of parts of activities are integral, certain activities are started or completed at non-integral points in time. Moreover, one and

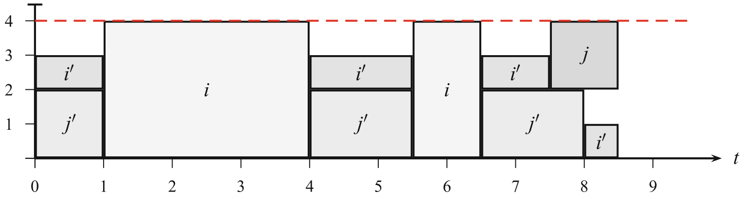


Fig. 13.3 Schedule with minimum makespan for example project

the same activity set may be in progress in different time intervals, and two activity sets may alternate several times.

13.3.2 Semantic Power of the Model

The preemptive project scheduling problem (P) defined in the previous section includes most of the preemptive project scheduling problems considered in the literature as special cases. As we already noticed, interruptions of an activity i can be prevented by adding the precedence relation $\Delta_{ii} = (1, 0, -p_i)$, which defines a maximum time lag of p_i time units between start time S_i and completion time C_i. Consequently, (P) is a generalization of respective non-preemptive scheduling problems. The case of discrete preemption can be modeled as follows. Assume that activity i can be interrupted precisely $t_1, t_2, \ldots, t_\nu$ time units after its start. We then split activity i into a sequence of subactivities $i_1, i_2, \ldots, i_\nu, i_{\nu+1}$ with durations $t_1, t_2 - t_1, \ldots, t_\nu - t_{\nu-1}, p_i - t_\nu$, introduce completion-to-start precedence relations $\Delta_{i_\mu i_{\mu+1}} = (1, 0, 0)$ between any two subsequent subactivities i_μ and $i_{\mu+1}$ and prevent the interruption of all subactivities i_μ by maximum time lags. Since integer preemption is a special case of discrete preemption, integer preemption problems are included as well.

With respect to the type of precedence relations, the model is very generic. Of course, an ordinary precedence relation between two activities i and j can be represented by relation $\Delta_{ij} = (1, 0, 0)$. A delayed precedence relation imposing a time lag $\delta_{ij}^{min} > 0$ between the completion of activity i and the start of activity j corresponds to the case of $\Delta_{ij} = (1, 0, \delta_{ij}^{min})$, whereas a maximum start-to-completion time lag δ_{ij}^{max} between i and j can be modeled as relation $\Delta_{ji} = (1, 0, -\delta_{ij}^{max})$. A relation $\Delta_{ij} = (1, 0, \delta_{ij})$ with arbitrary δ_{ij} is termed a time relationship and denoted by $(FS_{ij}^{min}, \varphi_i, 0, t)$ with $\varphi_i = \delta_{ij}$ by Quintanilla et al. (2012), see also Chap. 12 of this handbook. More generally, the relation $\Delta_{ji} = (\xi_j, \xi_i, -\delta_{ij}^{max})$ with $\delta_{ij}^{max} > 0$ gives rise to the maximum time lag constraint $t_j^-(\xi_j) \leq t_i^+(\xi_i) + \delta_{ij}^{max}$, which ensures that activity j attains relative progress ξ_j not later than δ_{ij}^{max} time units after the final portion $1 - \xi_i$ of activity i was started. An ordinary feeding precedence relation stating that activity j can only be started when $\xi_i \cdot 100\,\%$ of activity i have been completed (i. e., $S_j \geq t_i^-(\xi_i)$) is represented by relation $\Delta_{ij} = (\xi_i, 0, 0)$. As mentioned in Sect. 13.2.1, three further types of feeding precedence relations have been introduced by Alfieria et al. (2011), corresponding to the inequalities $S_j \le t_i^-(\xi_i)$, $C_j \le t_i^-(\xi_i)$, and $C_j \ge t_i^-(\xi_i)$. Neither of these constraints is purposeful in the context of continuous preemption. The second inequality, however, could be slightly modified to $C_j \le t_i^+(\xi_i)$, being equivalent to relation $\Delta_{ji} = (1, \xi_i, 0)$. For the generalized work relationships proposed by Quintanilla et al. (2012), an analogous observation can be made. The work relationship $(SF_{ij}^{min}, \varphi_i, \varphi_j, w)$ states that the final φ_j % of activity j can only be started when the initial φ_i % of activity i have been completed, i. e., $t_j^+(1 - \varphi_j/100) \ge t_i^-(\varphi_i/100)$. Such a relationship can be expressed as relation $\Delta_{ij} = (\varphi_i/100, 1 - \varphi_j/100, 0)$. The three remaining types of work relationships giving rise to the inequalities $t_j^-(\varphi_j/100) \ge t_i^-(\varphi_i/100)$, $t_j^-(\varphi_j/100) \ge t_i^+(1 - \varphi_i/100)$, and $t_j^+(1 - \varphi_j/100) \ge t_i^+(1 - \varphi_i/100)$ cannot be formulated in our framework given that they are not meaningful when coping with continuous preemption (recall our discussion in Sect. 13.3.1).

Finally, we show that generalized feeding precedence relations may also serve to define lower and upper bounds on the relative progress of an activity with positive duration. We consider some relation $\Delta_{ij} = (\xi_i, \xi_j, \delta_{ij})$. Due to the nondecreasing property and the continuity of function x_j, the respective inequality $t_j^+(\xi_j) = \sup\{t \mid x_j(t) \le \xi_j\} \ge t_i^-(\xi_i) + \delta_{ij} =: t'$ is equivalent to the constraint $\xi_j \ge x_j(t')$, which imposes upper bound ξ_j on the relative progress of activity j at time t'. Symmetrically, the inequality can also be written as $t'' := t_j^+(\xi_j) - \delta_{ij} \ge t_i^-(\xi_i) = \min\{t \mid x_i(t) \ge \xi_i\}$, which is equivalent to lower bounding condition $x_i(t'') \ge \xi_i$.

13.4 Structural Issues

In this section we first present a reduction of our problem to a canonical representation, which significantly facilitates the analysis. Furthermore, following the general approach of Słowiński for continuous preemptive project scheduling problems, we give an alternative interpretation of the problem as a sequencing problem of resource-feasible activity sets of variable duration. Next, we develop necessary feasibility conditions, which may be checked in a preprocessing step of a solution procedure to detect infeasible instances. Finally, we are concerned with preemption gains for the project duration problem, i. e., with the question of how large the relative improvement in the makespan can become when we move from a non-preemptive to a preemptive problem setting.

13.4.1 Alternative Representations of the Problem

We begin by proving the following proposition, which allows us to restrict ourselves to precedence relations $\Delta_{ij} = (1, 0, \delta_{ij})$ with $\delta_{ij} \le 0$. We say that a problem instance is in canonical form if all precedence relations are specified in this manner. In what follows, we will designate completion-to-start minimum time lags between activities i and j by δ_{ij}^{cs}.

Proposition 13.1 (Canonical Representation). *Any instance of PS | temp,feed, pmtn | f can be transformed into an equivalent instance of problem PS | temp,pmtn | f containing only nonpositive and integral completion-to-start minimum time lags and no parallel arcs in the project network.*

Proof. Consider a generalized feeding precedence relation $\Delta_{ij} = (\xi_i, \xi_j, \delta_{ij})$. As already mentioned, the relation can be interpreted as a completion-to-start time lag between the initial portion ξ_i of activity i and the final portion $1 - \xi_j$ of activity j. That is why we split both activities i and j into two parts i_1, i_2 and j_1, j_2 with durations $p_{i_1} = \xi_i \cdot p_i$, $p_{i_2} = (1 - \xi_i) \cdot p_i$, $p_{j_1} = \xi_j \cdot p_j$, and $p_{j_2} = (1 - \xi_j) \cdot p_j$. The precedence relation is then translated into the completion-to-start relation $\Delta_{i_1 j_2} = (1, 0, \delta_{ij}^{cs})$ between activities i_1 and j_2. Furthermore, we link the two parts belonging to the original activities by completion-to-start relations $\Delta_{i_1 i_2} = \Delta_{j_1 j_2} = (1, 0, 0)$ and replace arcs that lead to the original nodes i or j by arcs leading to the respective initial parts i_1 or j_1 and arcs that emanated from nodes i or j by arcs emanating from the respective final parts i_2 or j_2. In case that there exist more than one generalized feeding precedence relation involving one and the same activity i or j of duration p and these relations refer to $\nu > 1$ different portions ξ, the portions are sorted in increasing order $\xi^1, \xi^2, \ldots, \xi^\nu$ and the activities are partitioned into $\nu + 1$ parts with durations $p_\mu = (\xi^\mu - \xi^{\mu-1}) \cdot p > 0$ for $\mu = 1, \ldots, \nu + 1$, where $\xi^0 := 0$ and $\xi^{\nu+1} := 1$. Next, we eliminate any delayed precedence relation $\Delta_{ij} = (1, 0, \delta_{ij})$ with positive time lag $\delta_{ij} > 0$ by introducing a dummy activity h of duration $h = \delta_{ij}$ and replacing the original relation Δ_{ij} by two ordinary precedence relations $\Delta_{ih} = \Delta_{hj} = (1, 0, 0)$. Now assume that arc set E contains parallel arcs, say $(i, j)^1$ and $(i, j)^2$. We can substitute the parallel arcs into a single arc (i, j) with weight $\delta_{ij}^{cs} = \max\{\delta_{(ij)^1}^{cs}, \delta_{(ij)^2}^{cs}\}$. Finally, we multiply all time lags and activity durations by the least common denominator of all δ_{ij}^{cs} to obtain integral time lags. □

According to Proposition 13.1, we may replace inequality (13.2) in the definition of problem (P) by the simpler constraint

$$S_j \geq C_i + \delta_{ij}^{cs} \quad ((i, j) \in E) \tag{13.3}$$

Example 13.3. We return to the problem instance with four real activities introduced in Example 13.2. According to the transformation rules that we applied in the proof of Proposition 13.1, we decompose activities i and j' into two parts each. Note that since both relations $\Delta_{ij'} = (0.75, 0.25, -1)$ and $\Delta_{ji} = (1, 0.75, -4)$ refer to the same portion $\xi_i = 0.75$ of activity i, it is not necessary to consider a third part of activity i. Moreover, the delayed precedence relation $\Delta_{ij} = (1, 0, 1)$ is eliminated by introducing dummy activity h. The resulting activity-on-node network of the canonical representation of the instance is displayed in Fig. 13.4.

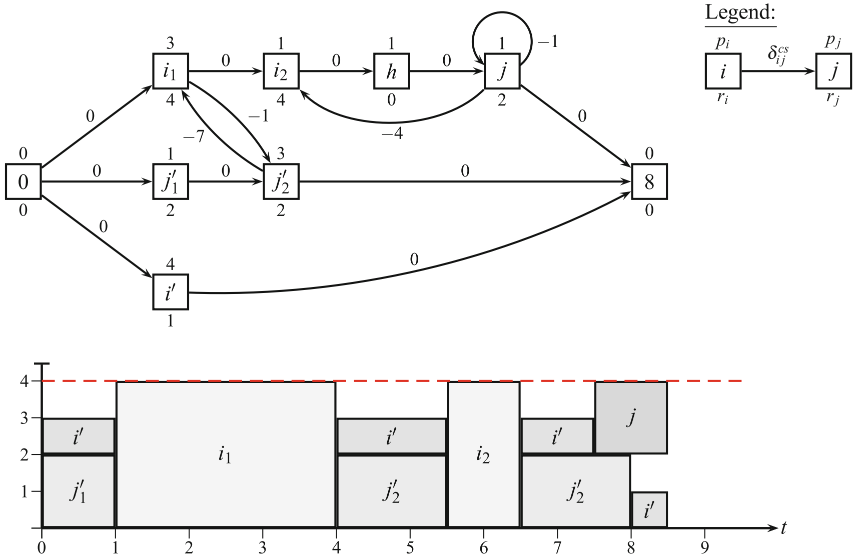


Fig. 13.4 Canonical form of example project and corresponding optimal schedule

In the remainder of this chapter we suppose that the problem is posed in the canonical form. We proceed by characterizing our problem as a sequencing problem of activity sets with variable durations. First we give a formal definition of a schedule and its decomposition into slices.

Definition 13.1 (Schedule and Slices). A *schedule* is a pair $\sigma = (\pi, z)$ composed of a finite sequence $\pi = (A_1, A_2, \ldots, A_\nu)$ of activity sets $A_\mu \subseteq V$ and an associated duration vector $z = (z_1, \ldots, z_\nu)$ with $z_\mu \geq 0$ for all $\mu = 1, \ldots, \nu$. For the sake of uniqueness we establish the convention that if $z_\mu = 0$, then $\mu = \nu$ or $z_{\mu+1} > 0$ and $A_\mu \supset A_{\mu+1}$. We say that schedule σ is *complete* if each activity is fully executed, i. e., if

$$\bigcup\nolimits_\mu A_\mu = V \text{ and } \sum_{\mu : i \in A_\mu} z_\mu = p_i \quad (i \in V) \tag{13.4}$$

Schedule σ is *resource-feasible* if and only if all sets A_μ with $z_\mu > 0$ are resource-feasible, i. e., if

$$\sum_{i \in A_\mu} r_{ik} \leq R_k \quad (\mu = 1, \ldots, \nu : z_\mu > 0;\ k \in \mathscr{R})$$

The schedule is *time-feasible* if the sequence π of activity sets with durations z is compatible with the completion-to-start time lags δ_{ij}^{cs}. These precedence relations can be expressed by exploiting that $C_i = \max\{\sum_{\lambda=1}^{\mu} z_\lambda \mid i \in A_\mu\}$ and $S_j =$

$\min\{\sum_{\lambda=1}^{\mu-1} z_\lambda \mid j \in A_\mu\}$, where $\sum_\emptyset z_\lambda := 0$. Hence, the temporal constraints (13.3) for the activities included in schedule σ can be formulated as

$$\min\{\sum_{\lambda=1}^{\mu-1} z_\lambda \mid j \in A_\mu\} \geq \max\{\sum_{\lambda=1}^{\mu} z_\lambda \mid i \in A_\mu\} + \delta_{ij}^{cs} \quad ((i,j) \in E : i,j \in \bigcup_\mu A_\mu)$$

Finally, we define a *feasible schedule* to be a complete, resource-feasible, and time-feasible schedule. An *optimal schedule* is a feasible schedule σ with minimum objective function value $f(z)$. For $f = C_{max}$ it holds that $f(z) = C_{max}(\sigma) = \sum_{\mu=1}^{\nu} z_\mu$. Schedule σ partitions interval $[0, C_{max}(\sigma)]$ into intervals τ_μ with left boundaries $t_\mu = \sum_{\lambda=1}^{\mu-1} z_\lambda$ and right boundaries $t'_\mu = \sum_{\lambda=1}^{\mu} z_\lambda$ ($\mu = 1, \ldots, \nu$). The pair $s_\mu = (\tau_\mu, A_\mu)$ is called the μ-th *slice* of schedule σ, and t_μ and t'_μ are referred to as the left and right ends of slice s_μ. We speak of a positive slice s_μ if its size $t'_\mu - t_\mu = z_\mu > 0$; the activity set A_μ of a positive slice s_μ is said to be active.

Remarks 13.1.

1. In difference to the case of ordinary precedence relations, it may be necessary that one and the same activity set A occurs more than once in sequence π to be able to encode an optimal solution (see the schedule depicted in Fig. 13.4 for an example).
2. Since the activity set V contains events, slice intervals τ may be singletons, half-open intervals, or open intervals. For example, the schedule displayed in Fig. 13.4 consists of the nine slices $s_1 = (\{0\}, \{0, i', j'_1\})$, $s_2 = (]0,1[, \{i', j'_1\})$, $s_3 = ([1,4[, \{i_1\})$, $s_4 = ([4,5.5[, \{i', j'_2\})$, $s_5 = ([5.5,6.5[, \{i_2\})$, $s_6 = ([6.5,7.5[, \{i', j'_2\})$, $s_7 = ([7.5,8[, \{j, j'_2\})$, $s_8 = ([8,8.5[, \{i', j\})$, and $s_9 = (\{8.5\}, \{8\})$.
3. The start time of activity j coincides with the minimum left end t_μ of a slice s_μ with $j \in A_\mu$, whereas the completion time of activity i equals the maximum right end t'_λ of a slice s_λ with $i \in A_\lambda$. That is why the precedence relations can be expressed as
$$\min_{\mu : j \in A_\mu} t_\mu \geq \max_{\lambda : i \in A_\lambda} t'_\lambda + \delta_{ij}^{cs} \quad ((i,j) \in E)$$
4. The set $\mathscr{A}(\sigma, t)$ of activities being in progress or occurring at time t, which is called the active set of schedule σ at time t, coincides with the set A_μ for which $t \in \tau_\mu$. This is due to the convention that was established in Definition 13.1 to ensure the uniqueness of the schedule representation of a solution to problem (P).
5. Any feasible schedule σ with minimum makespan $C_{max}(\sigma)$ processes at least one activity i until the project has been terminated, i. e., $\mathscr{A}(\sigma, t) \neq \emptyset$ for all $t \in [0, C_{max}(\sigma)]$.
6. Complete schedules σ and trajectories x are alternative ways of encoding solutions to project scheduling problem (P). Given a schedule σ, the trajectorial

representation x of the solution can be obtained via

$$x_i(t) = \frac{1}{p_i}\int_{t'=0:i\in\mathscr{A}(\sigma,t')}^{t} \mathrm{d}t'$$

where for events i, de l'Hôpital's rule provides $x_i(t) = 1$ if $i \in \mathscr{A}(\sigma,t')$ for some $t' \leq t$ and $x_i(t) = 0$, otherwise.

In what follows we reformulate problem (P) as the combination of the disjunctive relaxation , which consists in sizing the slices of the schedule, and a sequencing problem of the slices subject to completion-to-start time lags.

Theorem 13.1 (Sizing and Sequencing). *Given an instance I of problem (P) and a set $\mathscr{A}'$ of (pairwise different) resource-feasible activity sets A with associated vector z of durations (or slice sizes) $z_A \geq 0$ satisfying the completeness requirements $\bigcup_{A\in\mathscr{A}'} A = V$ and $\sum_{A\in\mathscr{A}':i\in A} z_A = p_i$ for all $i \in V$, there exists a feasible schedule $\sigma = (\pi, z')$ for I with sequence $\pi = (A_\mu)_\mu$ on set $\mathscr{A}'$ and duration vector $z' = (z'_\mu)_\mu$ satisfying $\sum_{\mu:A_\mu=A} z'_\mu = z_A$ for all $A \in \mathscr{A}'$ if and only if the instance $I(z)$ of the single-machine problem $1 \mid temp, pmtn \mid \cdot$ with jobs $A \in \mathscr{A}'$ of durations z_A and completion-to-start time lags $\delta^{cs}_{AA'} = \max_{(i,j)\in(A\times A')\cap E} \delta^{cs}_{ij}$ for all (A, A') with $(A \times A') \cap E \neq \emptyset$ is feasible.*

Proof. Consider a feasible solution to single-machine problem $I(z)$. This solution gives rise to a schedule $\sigma = (\pi, z')$ composed of slices $s_\mu = (\tau_\mu, A_\mu)$. By construction of $I(z)$, it holds that $\sum_{\mu:A_\mu=A} z'_\mu = z_A$ for all $A \in \mathscr{A}'$ and hence $\sum_{\mu:i\in A_\mu} z_\mu = p_i$ for all $i \in V$, which means that σ is complete. Since all sets A_μ are resource-feasible, σ is resource-feasible as well. Moreover, the time lags of $I(z)$ ensure that $\min_{\mu:A_\mu=A'} t_\mu \geq \max_{\lambda:A_\lambda=A} t'_\lambda + \delta^{cs}_{AA'}$ for all (A, A') with $(A \times A') \cap E \neq \emptyset$. For each arc $(i, j) \in E$ it holds that $\delta^{cs}_{AA'} \geq \delta^{cs}_{ij}$ for all $(A \times A') : i \in A, j \in A'$. Consequently, it holds that $S_j - C_i = \min_{A':j\in A'} \min_{\mu:A_\mu=A} t_\mu - \max_{A:i\in A} \max_{\lambda:A_\lambda=A'} t'_\lambda \geq \delta^{cs}_{ij}$ for all $(i, j) \in E$, which provides the time-feasibility and hence the feasibility of schedule σ for instance I.

Now assume that we are given a feasible schedule $\sigma = (\pi, z')$ for instance I with sequence $\pi = (A_\mu)_\mu$ and $\sum_{\mu:A_\mu=A} z'_\mu = z_A$ for all $A \in \mathscr{A}'$. Since no two slices of σ overlap in time, σ represents a feasible solution to $I(z)$ if the completion-to-start precedence relations are satisfied for all (A, A') with $(A \times A') \cap E \neq \emptyset$. Since σ is feasible for I, it holds that $S_j \geq C_i + \delta^{cs}_{ij}$ for all $(i, j) \in E$. Accordingly, for a pair (A, A') with $(A \times A') \cap E \neq \emptyset$ we have

$$\min_{i\in A, j\in A'}(S_j - C_i - \delta^{cs}_{ij}) \geq 0 \Leftrightarrow \min_{j\in A'} S_j \geq \max_{i\in A} C_i + \max_{i\in A, j\in A'} \delta^{cs}_{ij}$$

$$\Leftrightarrow \min_{j\in A'} \min_{\mu:j\in A_\mu} t_\mu \geq \max_{i\in A} \max_{\lambda:i\in A_\lambda} t'_\lambda + \delta^{cs}_{AA'} \Rightarrow \min_{\mu:A_\mu=A'} t_\mu \geq \max_{\lambda:A_\lambda=A} t'_\lambda + \delta^{cs}_{AA'}$$

which proves the precedence relation between jobs A and A' to be satisfied. □

Remark 13.2. Without loss of generality, the set $\mathscr{A}'$ of activity sets A considered in Theorem 13.1 can be assumed to contain only activities that can pairwise overlap in some time-feasible schedule. Let $d_{ij}^{cs} \in \mathbb{Z} \cup \{-\infty\}$ be the completion-to-start time lag between two activities i and j implied by the precedence relations from set E. Time lag d_{ij}^{cs} equals the minimum difference $S_j - C_i$ arising in any complete and time-feasible schedule. In Sect. 13.6 we present a modified version of Floyd and Warshall's algorithm for computing longest path lengths in a network, which yields the matrix $D^{cs} = (d_{ij}^{cs})_{i,j \in V}$ of all time lags. From the Helly property of intervals it follows that there exists a time-feasible schedule for which all activities $i \in A$ overlap in time precisely if A is an antichain of preorder $\Theta(D^{cs}) := \{(i,j) \in V \times V \mid d_{ij}^{cs} \geq 0\}$ (see Lemma 2.7 in Schwindt 2005). In what follows, the set of all (resource-)feasible antichains of $\Theta(D^{cs})$ and all singletons containing an event $i \in V$ is denoted by $\overline{\mathscr{A}}$. In slight abuse of terminology, we will designate all sets $A \in \overline{\mathscr{A}}$ as feasible antichains.

For the special case of ordinary precedence relations, the existence of a feasible single-machine schedule can be verified efficiently. The problem is feasible if and only if the directed graph $G = (\mathscr{A}', E(\mathscr{A}'))$ with node set $\mathscr{A}'$ and arcs set $E(\mathscr{A}') = \{(A, A') \in \mathscr{A}' \times \mathscr{A}' \mid (A \times A') \cap E \neq \emptyset\}$ is acyclic. This assertion also follows from Theorem 2.7 given in Damay et al. (2007), for the special case where all activities are preemptive. In case of generalized precedence relations considered in this chapter, checking the feasibility of the single-machine problem is strongly $\mathscr{NP}$-complete. As it has been shown by Wikum et al. (1994), the feasibility variant of the non-preemptive single-machine problem with nonpositive completion-to-start time lags $\delta_{AA'}^{cs}$ between jobs A and A' is strongly $\mathscr{NP}$-complete even if E is symmetric, $\delta_{AA'}^{cs} \cdot \delta_{A'A}^{cs} = 0$ for all $(A, A') \in E(\mathscr{A}')$, and the precedence graph containing the arcs (A, A') with $\delta_{AA'} = 0$ is a (special) intree (cf. problem "max delays, k $n_1, 1, \ldots, 1$-chains" in Table 1 of the above paper). Since the case of non-preemptive activities is contained in our problem setting, the single machine problem arising from the sizing of the activity sets is a generalization of the problem investigated by Wikum et al. (1994).

Now we are ready to reformulate problem (P) as a sizing-and-sequencing problem (P') of the set of all feasible antichains $A \in \overline{\mathscr{A}}$.

$$(P') \left\{ \begin{array}{lll} \text{Min. } f(z) & & \\ \text{s.t.} \displaystyle\sum_{A \in \overline{\mathscr{A}} : i \in A} z_A = p_i & & (i \in V) \\ \qquad \text{Instance } I(z) \text{ is feasible} & & \\ \qquad z_A \geq 0 & & (A \in \overline{\mathscr{A}}) \end{array} \right.$$

When considering the makespan criterion C_{max}, the objective function to be minimized is $f(z) = \sum_{A \in \overline{\mathscr{A}}} z_A$.

13.4.2 Feasibility Conditions

As we have seen before, the feasibility variant of problem (P) is strongly $\mathcal{NP}$-complete. In this section we present different necessary feasibility conditions, which may be evaluated for detecting the infeasibility of a problem instance I. These conditions form the basis of feasibility tests presented in Sect. 13.6.2.

Theorem 13.2 (General Feasibility Conditions). *For a given set of activities $U \neq \emptyset$ let $LB(U)$ be some lower bound on the minimum makespan $C^*_{max}(U) = \max_{i \in U} C_i - \min_{j \in U} S_j$ for executing all activities $i \in U$ in a feasible schedule. Instance I is only feasible if*

$$\max_{\emptyset \neq U \subseteq V} (LB(U) + \min_{i,j \in U} d^{cs}_{ij}) \leq 0$$

Proof. The negated completion-to-start time lag $-d^{cs}_{ij}$ between activities i and j equals the maximum time lag between the start of activity j and the completion of activity i, i. e., $C_i - S_j \leq -d^{cs}_{ij}$ for any complete and time-feasible schedule. Consequently, for each nonempty set U and each feasible schedule we have

$$\begin{aligned} -\min_{i,j \in U} d^{cs}_{ij} &= \max_{i,j \in U} (-d^{cs}_{ij}) \\ &\geq \max_{i,j \in U} (C_i - S_j) = \max_{i \in U} C_i - \min_{j \in U} S_j \\ &\geq C^*_{max}(U) \geq LB(U) \end{aligned}$$

which provides $LB(U) + \min_{i,j \in U} d^{cs}_{ij} \leq 0$. Hence, if there is some set $U \neq \emptyset$ such that $LB(U) + \min_{i,j \in U} d^{cs}_{ij} > 0$, then there does not exist any feasible schedule, i. e., instance I is infeasible. □

Let $V' = \{i \in V \mid p_i > 0\}$ denote the set of activities with positive durations. A set of activities $U \subseteq V'$, which is not resource-feasible, is called a forbidden set. The two activities i, j of a forbidden set $\{i, j\}$ are said to be incompatible. The following example shows that in difference to the non-preemptive case there exist feasible instances with incompatible activities i, j such that $p_i + p_j > -d^{cs}_{ij}$ and $p_i + p_j > -d^{cs}_{ji}$.

Example 13.4. Consider a forbidden set $U = \{i, j\}$ with activities of durations $p_i = p_j = 2$. We suppose that activity i can be interrupted for two time units, whereas activity j cannot be split. Furthermore, we define the minimum time lags $\delta^{cs}_{ij} = \delta^{cs}_{ji} = -3$. The corresponding time lag matrix is $D^{cs} = \begin{pmatrix} -4 & -3 \\ -3 & -2 \end{pmatrix}$ and thus $LB(U) + \min_{i,j \in U} d_{ij} = p_i + p_j + d_{ii} = 2 + 2 - 4 \leq 0$. Figure 13.5 shows the unique feasible schedule for activities i, j.

To apply feasibility conditions derived from Theorem 13.2, we need lower bounds $LB(U)$ on the minimum makespan $C^*_{max}(U)$ for processing activities $i \in U$.

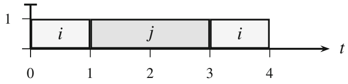


Fig. 13.5 Unique feasible schedule for set U of Example 13.4

The next proposition shows how such lower bounds can be obtained based on forbidden sets.

Proposition 13.2 (Lower Bound on Minimum Subset Makespan). *Given some integer $m \geq 2$, let $U \subseteq V'$ be a set of activities such that each subset $U' \subseteq U$ containing m elements is forbidden. Then $LB(U) = \frac{1}{m-1}\sum_{i \in U} p_i$ is a lower bound on makespan $C^*_{max}(U)$.*

Proof. From the definition of set U it follows that for each complete and resource-feasible schedule and any point in time, at most $m-1$ activities from set U are executed in parallel. Now consider some feasible schedule realizing the minimum makespan $C^*_{max}(U)$ for set U. Since the schedule is resource-feasible and because all activities $i \in U$ are entirely processed within $C^*_{max}(U)$ time units, it holds that $C^*_{max}(U) \cdot (m-1) \geq \sum_{i \in U} p_i$ and hence $LB(U) = \frac{1}{m-1}\sum_{i \in U} p_i \leq C^*_{max}(U)$. □

Corollary 13.1 (Feasibility Conditions Based on Forbidden Sets).

1. *Let $\mathscr{F}'$ be the set of all activity sets $U \subseteq V'$ with $|U| \geq 2$ containing only pairwise incompatible activities. Instance I is only feasible if*

$$\max_{U \in \mathscr{F}'} \Big(\sum_{i \in U} p_i + \min_{i,j \in U} d^{cs}_{ij} \Big) \leq 0 \tag{13.5}$$

2. *Let $\overline{\mathscr{F}}$ be the set of all forbidden activity sets $U \subseteq V'$. Instance I is only feasible if*

$$\max_{U \in \overline{\mathscr{F}}} \Big(\sum_{i \in U} p_i + (|U|-1) \cdot \min_{i,j \in U} d^{cs}_{ij} \Big) \leq 0 \tag{13.6}$$

Proof. The proofs of assertions 1. and 2. immediately follow from Theorem 13.2 and Proposition 13.2 by choosing $m = 2$ and $m = |U|$, respectively. □

The two feasibility conditions of Corollary 13.1 can be evaluated by solving mixed-integer linear programs of moderate sizes. If one of the two programs yields a solution with a positive objective function value, the tested instance is infeasible. Let us first consider program (FT_1) implementing the first test. The binary variables x_i serve to encode activity sets U, where $x_i = 1$ if $i \in U$ and $x_i = 0$, otherwise. The value of the continuous variable d^{min} coincides with $\min_{i,j \in U} d^{cs}_{ij}$ in any optimal solution to (FT_1).

$$
(FT_1)\left\{\begin{array}{llll}
\text{Max. } f(x) = \sum\limits_{i \in V'} p_i x_i + d^{min} & & & \\
\text{s. t. } \quad \sum\limits_{i \in V'} x_i \geq 2 & & & (13.7) \\
\quad x_i + x_j \leq 1 & (i, j \in V' : \{i, j\} \notin \overline{\mathscr{F}}) & & (13.8) \\
\quad d^{min} \leq d_{ij}^{cs} \cdot (x_i + x_j - 1) & ((i, j) \in V' \times V' : d_{ij}^{cs} \leq 0) & & (13.9) \\
\quad x_i \in \{0, 1\} & (i \in V') & &
\end{array}\right.
$$

The objective function expresses the argument $\sum_{i \in U} p_i + \min_{i,j \in U} d_{ij}^{cs}$ of the left-hand side of condition (13.5). Constraint (13.7) ensures that only sets U containing at least two activities are taken into account, and inequality (13.8) excludes all sets U containing two compatible activities i, j. Condition (13.9) guarantees that $d^{min} \leq \min_{i,j \in U} d_{ij}^{cs}$.

The second program (FT_2) evaluates condition (13.6). It involves two types of binary variables, x_i and y_k, as well as continuous variable D^{min}. Variables x_i have the same interpretation as for program (FT_1), whereas variables y_k are used to model the forbiddenness of sets U, where $y_k = 0$ if the joint requirements of the activities $i \in U$ for resource k do not exceed the capacity R_k. In an optimal solution to (FT_2), variable D^{min} equals $(|U| - 1) \cdot \min_{i,j \in U} d_{ij}^{cs}$.

$$
(FT_2)\left\{\begin{array}{lll}
\text{Max. } f(x) = \sum\limits_{i \in V'} p_i x_i + D^{min} & & \\
\text{s. t. } \quad R_k y_k \leq \sum\limits_{i \in V'} r_{ik} x_i - 1 & (k \in \mathscr{R}) & (13.10) \\
\quad \sum\limits_{k \in \mathscr{R}} y_k \geq 1 & & (13.11) \\
\quad D^{min} \leq d_{ij}^{cs} \cdot \Big[(\sum\limits_{h \in V'} x_h - 1) + & & \\
\qquad (n-2)(x_i + x_j - 2) \Big] & ((i, j) \in V' \times V' : d_{ij}^{cs} \leq 0) & (13.12) \\
\quad x_i, y_k \in \{0, 1\} & (i \in V';\ k \in \mathscr{R}) &
\end{array}\right.
$$

The objective function of (FT_2) models the term $\sum_{i \in U} p_i + (|U| - 1) \cdot \min_{i,j \in U} d_{ij}^{cs}$ that is maximized on the left-hand side of condition (13.6). Inequality (13.10) sets y_k to zero if U does not violate capacity R_k. Constraint (13.11) ensures that set U exceeds the capacity of at least one resource, i. e., that U is a forbidden set. By inequality (13.12) the value of D^{min} is equal to $(|U| - 1) \cdot \min_{i,j \in U} d_{ij}^{cs}$ in any optimal solution to (FT_2).

13.4.3 Preemption Gains for Makespan Minimization

The preemptive version of a scheduling problem always represents a relaxation of the respective non-preemptive problem. As we have seen for our scheduling problem, this relaxation is not necessarily more tractable to solve. An optimal preemptive schedule, however, offers nonnegative preemption gains γ, which are defined as the relative improvement in the objective function value with respect to an optimal schedule for the non-preemptive problem.

For instances of machine scheduling problems with m uniform machines, Braun and Schmidt (2003) examined the ratio ρ_i of the minimum makespan C^i_{max} for the version of the problem in which at most i interruptions are allowed and the minimum makespan $C^{pmtn}_{max} = C^{\infty}_{max}$ for the fully preemptive version $Pm \mid pmtn \mid C_{max}$ of the problem. They showed that ratio ρ_i is bounded from above by

$$\overline{\rho}_i = 2 - \frac{2}{\frac{m}{i+1} + 1}$$

In particular, the ratio cannot exceed $\overline{\rho}_0 = 2 - \frac{2}{m+1} < 2$ when considering the non-preemptive problem $Pm \mid\mid C_{max}$ where $i = 0$. The ratio provides an upper bound on the preemption gains γ of $\overline{\gamma} = 1 - \frac{1}{\overline{\rho}_0} = \frac{1}{2} - \frac{1}{2m}$. Since for problem (P) the feasibility of an instance may depend on the preemptability of activities, the preemption gains γ may be unbounded. The following proposition shows that even when limiting the analysis to feasible instances of (P), the tightest upper bound $\overline{\gamma}$ on gains γ cannot be smaller than $2/3$.

Proposition 13.3. *There exist feasible instances I of problem $Pm \mid temp, pmtn \mid C_{max}$ with $m \geq 2$ with preemption gains $\gamma = \frac{2}{3} - \frac{2}{9m+3} - \varepsilon$ for arbitrarily small $\varepsilon > 0$.*

Proof. We consider an instance I of a parallel machine problem with m parallel processors and nonpositive completion-to-start time lags. Aside from the dummy activities 0 and $n+1$ set V comprises two sets of activities V_1 and V_2. Set V_1 contains $m+1$ activities $i = 1, \dots, m+1$ of equal duration $p_i = p$, where $p = a \cdot m$ for some $a \in \mathbb{N}$. The second set V_2 consists of m unit-duration activities $i = m+2, \dots, 2m+1$ and a dummy activity $2m+2$ of duration $\frac{m+1}{m}p$, which does not require a processor. Between any two different activities $i, j \in V_1$ we introduce time lags $\delta^{cs}_{ij} = -2p$. The execution times of the unit-duration activities $i \in V_2$ are all fixed to interval $[\frac{m+1}{m}p, \frac{m+1}{m}p + 1[$ by time lags $\delta^{cs}_{(2m+2)i} = 0$ and $\delta^{cs}_{i0} = -(\frac{m+1}{m}p + 1)$.

Optimal schedules for the preemptive and the non-preemptive version of the problem are shown in Fig. 13.6. The dark shaded activities $i \in V_2$ serve to erect a barricade. If activity splitting is allowed, the light shaded activities $i \in V_1$ can all be positioned left to the barricade, and the resulting makespan is $C^{pmtn}_{max} = \frac{m+1}{m}p + 1$. Otherwise, the makespan $C^0_{max}(V_1)$ for processing the activities from set V_1 equals

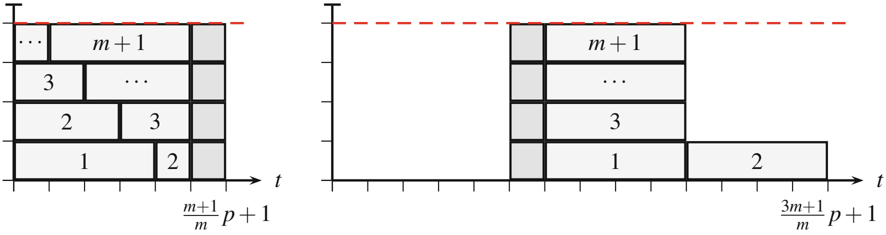


Fig. 13.6 Optimal schedules for problems $Pm \mid temp, pmtn \mid C_{max}$ and $Pm \mid temp \mid C_{max}$

$2p$ because one activity must be shifted behind the completion of the others. Due to the maximum time lags of $2p$ time units between all activities $i, j \in V_1$, the complete set V_1 can only be started behind the barricade, leading to a makespan of $C^0_{max} = \frac{3m+1}{m}p + 1$. Hence, the resulting preemption gains are $\gamma = 1 - \frac{(m+1)p+m}{(3m+1)p+m}$. By interpreting gains γ as a function $\gamma(p)$ of the duration of the activities $i \in V_1$ and taking the limit as p tends to infinity, we obtain $\lim_{p\to\infty} \gamma(p) = 1 - \frac{m+1}{3m+1} = \frac{2}{3} - \frac{2}{9m+3}$, which concludes the proof. □

13.5 A Novel MILP Problem Formulation

In this section we propose a new type of MILP model for preemptive (and non-preemptive) project scheduling problems. In Sect. 13.5.1 we develop the MILP formulation, and in Sect. 13.5.2 we analyze the size of the model in terms of the number of slices required to represent a feasible schedule with minimum makespan. As a by-product, we obtain an upper bound on the number of activity interruptions.

13.5.1 Formulation of the Model

The model formulation is based on the schedule representation of solutions to problem (P). According to this representation, a solution is defined as a sequence $(s_1, \ldots, s_\nu)$ of slices s_μ with associated feasible antichains A_μ that are in progress in the respective slices s_μ. In contrast to the approaches by Słowiński (1980) and Damay et al. (2007), the feasible antichains are not input of the model but are encoded by binary variables $y_{i\mu}$ that indicate whether or not activity i is executed in slice s_μ. Thus, it is no longer necessary to consider a decision variable z_A for each feasible antichain A, and we obtain a much more compact problem formulation. By introducing for each slice s_μ the continuous decision variable z_μ representing its size, the increase in the progress proportion x_i of activity i during slice s_μ is

$$\Delta x_{i\mu} = \frac{z_\mu}{p_i} \cdot y_{i\mu} \tag{13.13}$$

The completeness conditions (13.4) for the schedule can be stated as $\sum_{\mu=1}^{\nu} \Delta x_{i\mu}=1$ $(i \in V)$. The requirement of the activities $i \in A_\mu$ for resource k equals $\sum_{i\in V'} r_{ik} \cdot y_{i\mu}$ and must not exceed capacity R_k. If activity i is in progress in slice s_μ (i. e., $y_{i\mu} = 1$), then $S_i \leq \sum_{\lambda=1}^{\mu-1} z_\lambda$ and $C_i \geq \sum_{\lambda=1}^{\mu} z_\lambda$. Both inequalities are binding if i is started or completed, respectively, in slice s_μ. By combining the above considerations, we obtain the MILP formulation (P'') of problem (P) for the makespan criterion C_{max}, where $p^{max} = \max_{i\in V} p_i$ and UB denotes some upper bound on the minimum project duration. The model can easily be adapted to other objective functions f, provided that they can be expressed as piecewise linear and convex functions in the decision variables Δx, y, and z. Examples of such objective functions considered in project scheduling will be indicated in the sequel.

$$(P'')\left\{\begin{array}{lll}
\text{Min. } C_{max} = \sum_{\mu=1}^{\nu} z_\mu & & \\
\text{s.t.} & 0 \leq z_\mu - p_i \Delta x_{i\mu} \leq p^{max} \cdot (1 - y_{i\mu}) & (i \in V;\ \mu = 1,\ldots,\nu) \\
& 0 \leq \Delta x_{i\mu} \leq y_{i\mu} & (i \in V;\ \mu = 1,\ldots,\nu) \\
& \sum_{\mu=1}^{\nu} \Delta x_{i\mu} = 1 & (i \in V) \\
& \sum_{i\in V'} r_{ik} \cdot y_{i\mu} \leq R_k & (\mu = 1,\ldots,\nu;\ k \in \mathscr{R}) \\
& S_i \leq \sum_{\lambda=1}^{\mu-1} z_\lambda + UB \cdot (1 - y_{i\mu}) & (i \in V;\ \mu = 1,\ldots,\nu) \\
& C_i \geq \sum_{\lambda=1}^{\mu} z_\lambda - UB \cdot (1 - y_{i\mu}) & (i \in V;\ \mu = 1,\ldots,\nu) \\
& S_j \geq C_i + \delta_{ij}^{cs} & ((i,j) \in E) \\
& y_{i\mu} \in \{0,1\} & (i \in V;\ \mu = 1,\ldots,\nu)
\end{array}\right.$$

The first two pairs of inequalities in (P'') serve us to eliminate the bilinear products $z_\mu \cdot y_{i\mu}$ in the definition (13.13) of variables $\Delta x_{i\mu}$. As we will show in Sect. 13.5.2, the number ν of slices s_μ is linear in the number of activities; hence the model is polynomial in the problem size. A variant of the model for the non-preemptive version of our problem is obtained by adding the equations $C_i = S_i + p_i$ for all $i \in V$.

Remarks 13.3.

1. For the case of ordinary precedence relations, the precedence constraints can be expressed via the inequalities

$$(\nu-\mu+1)\cdot y_{j\mu} \le \nu-\mu+1-\sum_{\lambda=\mu}^{\nu} y_{i\lambda} \quad ((i,j)\in E;\ \mu=1,\dots,\nu) \tag{13.14}$$

which yields a version of model (P'') with a stronger LP relaxation. In our computational experiments, which are described in Sect. 13.7, we added conditions (13.14) for all $(i,j) \in E : \delta_{ij}^{cs} = 0$. Moreover, we tried a large number of redundant constraints, intended to strengthen the LP relaxation and to speed up the solution of the model. From these investigations we identified the two following inequalities, which had a positive impact on computation times.

$$S_i + p_i \le C_i \le C_{max} - d_{i(n+1)}^{cs} \quad (i \in V)$$

2. As mentioned before, analogous models can be formulated for any objective function, which can be expressed as a linear or piecewise linear and convex objective function in the decision variables. For example, the objective function of the total availability cost problem $PS \mid temp \mid \Sigma c_k \max r_{kt}$ (see, e.g., Chaps. 15 and 16 of this handbook)

$$f(\sigma) = \sum_{k\in\mathscr{R}} c_k \max\{r_k(\sigma,t) \mid t \ge 0\} = \sum_{k\in\mathscr{R}} c_k \max_{\mu=1,\dots,\nu} \sum_{i\in V'} r_{ik} y_{i\mu}$$

with $r_k(\sigma,t) = \sum_{i\in\mathscr{A}(\sigma,t)} r_{ik}$ is piecewise linear and convex in variables $y_{i\mu}$. The same holds true for the total adjustment cost problem $PS \mid temp \mid \Sigma c_k \Sigma \Delta_{kt}$ (see Chap. 17 of this handbook) with objective function

$$f(\sigma) = \sum_{k\in\mathscr{R}} c_k \sum_{t\in DT} [\Delta r_k(\sigma,t)]^+ = \sum_{k\in\mathscr{R}} c_k \sum_{\mu=1}^{\nu} \left[\sum_{i\in V'} r_{ik}(y_{i\mu} - y_{i(\mu-1)})\right]^+$$

where $y_{i0} := 0$ and $[a]^+ := \max\{a,0\}$, DT denotes the set containing $t = 0$ and all jump discontinuities t in resource profiles $r_k(\sigma,\cdot)$ and $\Delta r_k(\sigma,t)$ is the positive or negative step height of $r_k(\sigma,\cdot)$ at time t.

13.5.2 Number of Slices and Activity Interruptions

In order to instantiate model (P'') for given instances I, we need know the number ν of required slices. In this section we prove that this number is bounded from above by $2n - 1$ and that this bound is tight. Furthermore, we derive an upper bound on the maximum number of activity preemptions.

Theorem 13.3 (Number of Slices). *Let $n_1 = |V'|$ be the number of activities $i \in V$ with positive durations p_i. For each feasible instance I of $PS \mid temp, pmtn \mid C_{max}$ in canonical form there exists an optimal schedule with at most $n + n_1 - 1$ positive slices s_μ.*

Proof. Given an optimal solution $\zeta = (S, C, \Delta x, y, z)$ to model (P'') for instance I, let $\mathscr{S} \subseteq \{1, \dots, \nu\}$ be the index set of slices s_μ occupied by at least one activity, i. e., $\mathscr{S} = \{\mu = 1, \dots, \nu \mid \sum_{i \in V} y_{i\mu} \geq 1\}$. Since at any point in time $t < C_{max}$ at least one activity $i \in V$ is in progress (see item 5 of Remarks 13.1), we have $z_\mu = 0$ if $\mu \notin \mathscr{S}$. Suppose that all binary variables $y_{i\mu}$ of program (P'') are fixed to zero or one according to their values in ζ. Problem (P'') then turns into a linear program with decision variables S_i, C_i, z_μ, and $\Delta x_{i\mu}$ for $i \in V$ and $\mu \in \mathscr{S}$. The variables $\Delta x_{i\mu}$ can be eliminated from the linear program by replacing equations $\sum_{\mu=1}^{\nu} \Delta x_{i\mu} = 1$ $(i \in V)$ with $\sum_{\mu=1}^{\nu} z_{i\mu} = p_i$ $(i \in V')$. For activity i we denote by $\mu_i^{min} := \min\{\mu \in \mathscr{S} \mid y_{i\mu} = 1\}$ and $\mu_i^{max} := \max\{\mu \in \mathscr{S} \mid y_{i\mu} = 1\}$ the first and the last slice to which i has been assigned. The linear program can then be formulated as follows:

$$
(LP) \left\{
\begin{array}{llll}
\text{Min. } C_{max} = \displaystyle\sum_{\mu \in \mathscr{S}} z_\mu & & & \\
\text{s. t.} \quad \displaystyle\sum_{\mu \in \mathscr{S}: y_{i\mu} = 1} z_\mu = p_i & & (i \in V') & (13.15) \\
\quad S_i = \displaystyle\sum_{\mu \in \mathscr{S}: \mu < \mu_i^{min}} z_\mu & & (i \in V) & (13.16) \\
\quad C_i = \displaystyle\sum_{\mu \in \mathscr{S}: \mu \leq \mu_i^{max}} z_\mu & & (i \in V) & (13.17) \\
\quad S_j \geq C_i + \delta_{ij}^{cs} & & ((i, j) \in E) & (13.18) \\
\quad z_\mu \geq 0 & & (\mu \in \mathscr{S}) &
\end{array}
\right.
$$

Since we set the values of the binary variables according to solution ζ, the part $\zeta' = (S, C, z)$ of ζ solves linear program (LP). We construct an equivalent linear program (LP') whose optimal basic solutions also solve (LP) and contain no more than $n + n_1 - 1$ variables z_μ with positive values.

For each activity i starting at time $S_i = 0$ we replace Eq. (13.16) by the nonnegativity condition $S_i \geq 0$; analogously, if $C_i = 0$, we remove Eq. (13.17) for i and add the corresponding condition $C_i \geq 0$. In this way, for each of the n' remaining Eqs. (13.16) and (13.17) we have a variable S_i or C_i with positive value in ζ'.

Let $E' \subseteq E$ be the set of arcs in which the loops (i,i) have been deleted. As it is well-known (see, e.g., Neumann et al. 2003, Sect. 1.3) the set of generalized precedence relations (13.18) referring to E' can be replaced by a set of equations $S_j = C_i + \delta_{ij}^{cs}$ for which the respective arcs $(i,j) \in E'$ belong to a spanning outtree of the precedence graph $G = (V, E)$ rooted at node 0. In fact, the arcs of the equations form a spanning forest G' of G, and without loss of generality we may assume that G' contains at least one arc emanating from node 0 and exactly one arc leading to node $n+1$. Each equation corresponds to a generalized precedence relation that is binding in ζ'. Since spanning forest G' contains no more than $|V| - 1 = n + 1$ arcs, we obtain at most $n+1$ equations, which replace the constraints (13.18) for $(i,j) \in E'$. The set of all binding precedence relations $(i,j) \in E$ may also contain loops. For each such loop (i,i), however, the start time S_i is explained by the equation $S_i = C_i + \delta_{ii}^{cs}$, which implies that G' can be chosen in such a way that node i is a source of a component of G'. This means that for the generalized precedence relations (13.18) we still need at most one equation per activity $i \neq 0$, such that the number of these equations remains bounded by $n+1$.

We can save two more equations by taking advantage of the convention that instance I is specified in canonical form. First, we consider the project termination node $n+1$ of the spanning forest. Since $p_{n+1} = 0$ and the objective function value $\sum_{\mu \in \mathscr{S}} z_\mu$ coincides with C_{n+1}, it holds that in ζ' variable C_{n+1} takes its minimum value S_{n+1}. From $\delta_{i(n+1)}^{cs} = 0$ for all $(i, n+1) \in E$ it follows that $S_{n+1} = C_i$ for the predecessor i of $n+1$ in G'. Thus, we may remove equation $S_{n+1} = C_i$ and replace equations $S_j = C_{n+1} + \delta_{(n+1)j}^{cs}$ by equations $S_j = C_i + \delta_{(n+1)j}^{cs}$ if there exist successors j of $n+1$ in G'. Second, the canonical form also implies $\delta_{0i}^{cs} = 0$ for all $(0,i) \in E$. At least one of these arcs $(0,i)$ belongs to G', and the respective equations can be removed since their inequalities (13.18) are already expressed as the nonnegativity conditions $S_i \geq 0$.

In sum, the transformed linear program (LP') comprises n_1 equations of type (13.15), $n' \leq 2(n+2)$ equations of type (13.16) and (13.17), and at most $n-1$ equations of type $S_j = C_i + \delta_{ij}^{cs}$. Hence, any basic solution ζ'' to (LP') contains no more than $n_1 + n' + n - 1$ basic variables. Since each variable with a positive value must be a basic variable, n' out of these basic variables correspond to positive start and completion times S_i and C_i. There remain at most $n_1 + n - 1$ basic variables that may correspond to positive variables z_μ. By construction of (LP'), an optimal basic solution ζ'' to (LP') also solves (LP), which shows that the number of positive slices s_μ in a solution to instance I can be bounded from above by $n + n_1 - 1$. □

Remarks 13.4.

1. From Theorem 13.3 it follows that the number of slices required in model (P'') is $\nu = n + n_1 - 1 + (n + 2 - n_1) = 2n + 1$.
2. Equations (13.16)–(13.18) as well as decision variables S_i and C_i can be eliminated from (LP) by expressing the generalized precedence relations by means of the inequality

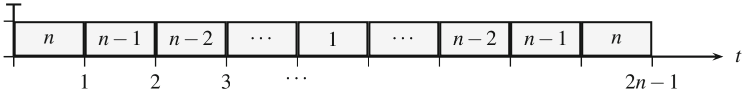


Fig. 13.7 Unique feasible schedule, being composed of $2n - 1$ positive slices

$$\sum_{\mu_j^{min} \leq \mu \leq \mu_i^{max}} z_\mu \leq -\delta_{ij}^{cs} \quad ((i,j) \in E) \tag{13.19}$$

The following example shows that the bound of $2n - 1$ positive slices is tight even if $|\mathscr{R}| = 1$, $R = 1$, and $r_i = 1$ for all $i \in V$.

Example 13.5. Consider the single-machine problem with n activities of durations $p_1 = 1$ and $p_i = 2$ for $i = 2, \ldots, n$. Any two consecutive activities i and $i + 1$ are mutually linked by the two time lags $\delta_{i(i+1)}^{cs} = \delta_{(i+1)i}^{cs} = -2i$. Figure 13.7 shows the unique feasible schedule for this instance, which is composed of $2n - 1$ positive slices.

Remark 13.5. The first item of Remarks 13.3 implies that Eqs. (13.16)–(13.18) of the linear program (*LP*) in the proof of Theorem 13.3 can be omitted for all arcs corresponding to ordinary precedence relations. Hence, for any instance I of problem $PS \mid prec, pmtn \mid C_{max}$ there exists an optimal schedule with at most n positive slices. This property also follows from the Rational Structure Theorem given in Baptiste et al. (2004).

A common assumption in preemptive scheduling is that activities can be interrupted at no cost. In practice, however, splitting activities may incur additional coordination or setup efforts, and schedules according to which activities are frequently stopped and resumed would be unworkable. That is why we are interested in the maximum number of interruptions that must be accepted to obtain a feasible schedule with minimum objective function value. For a generalization of problem $PS \mid temp, pmtn \mid f$ considered in this chapter, Baptiste et al. (2004) have shown that the number of activity interruptions remains polynomially bounded in the size of the problem instance. As a direct consequence of Theorem 13.3 we obtain the following upper bound, which, however, may be far from being tight.

Corollary 13.2. *For each feasible instance I of $PS \mid temp, pmtn \mid C_{max}$ in canonical form there exists an optimal schedule with at most $n(n-1)$ activity interruptions. If only ordinary precedence relations are present, the number of activity interruptions is limited by $\frac{n(n-1)}{2}$.*

The next example provides an instance for which $n^2/4$ interruptions are needed to obtain a feasible schedule.

Example 13.6. We consider a project with an even number n of real activities and one renewable resource of capacity $R = \frac{n}{2}$. The set of real activities consists of two

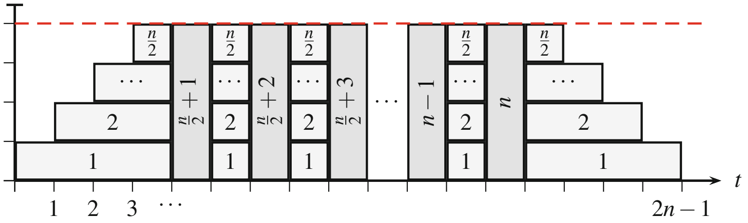


Fig. 13.8 Unique feasible schedule, incurring $\frac{n^2}{4}$ interruptions

subsets V_1 and V_2. Set V_1 contains activities $i = 1, \ldots \frac{n}{2}$ of durations $\frac{3}{2}n - 2i + 1$ and resource requirements $r_i = 1$ (light shaded activities in Fig. 13.8). Two consecutive activities i and $i+1$ from V_1 are linked by a time lag $\delta^{cs}_{i(i+1)} = -2(n-i)$. Moreover, we introduce the time lags $\delta^{cs}_{i0} = -(2n-i)$ for all $i \in V_1$, which define the deadlines $\overline{d}_i = 2n - i$ for the completion of activities $i \in V_1$.

Set V_2 contains unit-duration activities $i = \frac{n}{2} + 1, \ldots, n$ with resource requirements of $r_i = \frac{n}{2}$ and deadlines $\overline{d}_i = -\delta^{cs}_{i0} = 2i - \frac{n}{2} - 1$ (dark shaded activities in Fig. 13.8). Since $p_1 = \frac{3}{2}n - 1 \geq \overline{d}_i$ for all $i \in V_2$, activity 1 is completed after all activities $i \in V_2$, which implies $C_1 \geq p_1 + \frac{n}{2} \cdot 1 = \frac{3}{2}n - 1 + \frac{n}{2} = 2n - 1$. Due to deadline $\overline{d}_1 = 2n - 1$, activity 1 is pulled tight and must be completed precisely at time $2n - 1$. By adding time lags $\delta^{cs}_{1i} = -(\frac{5}{2}n - 2i + 1)$, the execution times of all activities i from set V_2 are fixed to intervals $[2(i-1) - \frac{n}{2}, 2(i-1) - \frac{n}{2} + 1[$.

Time lag $\delta^{cs}_{1,2} = -2n + 2$ constrains activity 2 to start no earlier than $t = 1$ and thus $C_2 \geq 1 + p_2 = \frac{3}{2}n - 2$. Since the completion time of activity 2 is right-shifted by one time unit for each activity $i \in V_2$ that must be completed earlier, it also finishes after all activities $i \in V_2$, and its earliest completion time $2n - 2$ coincides with its deadline $\overline{d}_2$. Applying iteratively the same arguments for activities $i = 3, \ldots, \frac{n}{2}$ shows that all activities $i \in V_1$ are completed at their respective deadlines $\overline{d}_i$ after all activities from set V_2. In sum, the schedule displayed in Fig. 13.8 is the only feasible solution and causes $\frac{n}{2} \cdot \frac{n}{2} = \frac{n^2}{4}$ interruptions of activities. It is composed of $2n - 1$ positive slices.

13.6 Preprocessing Methods and Lower Bounds

The performance of MILP solvers on the mixed-binary linear program (P'') can be significantly improved by applying preprocessing techniques and providing effective lower bounds. In this section we present a method for computing the matrix D^{cs} of all transitive time lags d^{cs}_{ij}, algorithms for enhancing matrix D^{cs} by time lags that are implied by resource constraints, an efficient heuristic for testing the feasibility of problem instances, a simple method for fixing the values of certain decision variables, and a column-generation based lower bound on the minimum project duration.

13.6.1 Computing and Tightening the Time Lag Matrix

The necessary feasibility conditions proposed in Sect. 13.4.2 are based on the matrix D^{cs} of transitive completion-to-start time lags, which determine the maximum feasible time span $-d^{cs}_{ji}$ between the start of activity i and the completion of activity j between any two activities i, j. For non-preemptive problems with generalized precedence relations, the temporal constraints are usually specified as start-to-start time lags δ^{ss}_{ij} because the transitive hull of these time lags coincides with the distance matrix of the project network, i. e., the transitive time lags d^{ss}_{ij} coincide with the longest path lengths (or distances, for short) ℓ_{ij} from nodes i to nodes j. That is why matrix D^{ss} can be computed efficiently using some algorithm for longest path calculations in simple networks like the Floyd-Warshall algorithm (see Floyd 1962). This algorithm can be slightly modified to provide matrix D^{cs}. To explain how the algorithm works, we introduce an event-on-node network N', which contains a start node i^s and a completion node i^c for each activity i. The arcs of this network represent minimum time lags between the occurrence times of the events. For each activity $i \in V$, nodes i^s and i^c are linked by arc (i^s, i^c) weighted with the duration p_i of activity i. This arc ensures that the completion of i can occur p_i time units after its start at the earliest. The arcs $(i, j) \in E$ representing completion-to-start time lags between activities i and j are translated into arcs (i^c, j^s) in N'. In particular, this implies that non-preemptive activities i, which were represented as loops in project network N, create cycles (i^s, i^c, i^s) in N' and hence N' is a simple network without loops. By applying Floyd and Warshall's algorithm to N' we obtain the transitive time lags between the occurrence times of all events. The completion-to-start time lags d^{cs}_{ij} correspond to distances $\ell_{i^c j^s}$ from completion to start events.

In our implementation of the algorithm we perform all computations directly on project network N. This is possible because in N' it holds that $\ell_{i^s j^s} = \ell_{i^c j^s} + p_i$, $\ell_{i^s j^c} = \ell_{i^c j^s} + p_i + p_j$, and $\ell_{i^c j^c} = \ell_{i^c j^s} + p_j$ and consequently all "triple operations" $\ell_{hj} := \max\{\ell_{hj}, \ell_{hi} + \ell_{ij}\}$ updating the path lengths can be limited to path lengths from completion to start nodes. The resulting method is displayed in Algorithm 13.1.

There exist many situations in which the resource constraints and the precedence relations imply new precedence relations or, to put it differently, tighten the time lags d^{cs}_{ij}. In what follows we present two preprocessing procedures for detecting such precedence relations. The first method was devised by Heilmann and Schwindt (1997) for the non-preemptive problem $PS \mid temp \mid C_{max}$ and can, with minor modifications, be applied to problem (P). The idea consists in identifying for each pair of activities i and j the minimum durations of activities $h \in V$ that must be processed between the completion of i and the start of j. For each resource we

Algorithm 13.1: Modified Floyd-Warshall algorithm

```
Require: Instance $I$ of problem $PS \mid temp, pmtn \mid f$
Ensure: Compute time lag matrix $D^{cs}$
for all $i, j \in V$ do
    set $d_{ij}^{cs} := -\infty$;
end for
for all $(i, j) \in E$ do
    set $d_{ij}^{cs} := \delta_{ij}^{cs}$;
end for
for all $i \in V : p_i = 0$ do
    set $d_{ii}^{cs} := 0$;
end for
for all $i \in V$ do
    for all $h, j \in V$ do
        set $d_{hj}^{cs} := \max\{d_{hj}^{cs}, d_{hi}^{cs} + p_i + d_{ij}^{cs}\}$;
    end for
end for
```

then construct an instance of the single-machine problem $1 \mid r_j, q_j, pmtn \mid C_{max}$ with release dates r_j and quarantine times q_j whose optimum makespan yields a lower bound on the time lag $S_j - C_i$ between the completion of i and the start of j in any feasible schedule. Consequently, if the makespan is larger than the current value of d_{ij}^{cs}, we can set d_{ij}^{cs} to the makespan. The single-machine problem can be solved efficiently in $\mathcal{O}(n \log n)$ time by Carlier's preemptive version of Schrage's algorithm (see Carlier 1982).

As it is easily verified, the minimum processing time of activity h between the completion of i and the start of j is

$$p_h(i, j) = \max\{0, \min\{p_h, d_{ij}^{cs}, d_{ih}^{cs} + p_h, d_{hj}^{cs} + p_h\}\} \tag{13.20}$$

The time lag d_{ih}^{cs} defines a release date $r_h(i)$ for activity h after the completion of i, and the time lag d_{hj}^{cs} can be interpreted as a quarantine time $q_h(j)$ after the completion of activity h before the start of j.

Since we want to solve a single-machine problem, we have to scale the durations $p_h(i, j)$ with the ratio $\frac{r_{hk}}{R_k} \leq 1$ of resource requirement and resource capacity, which gives rise to resource-specific durations $p_h(i, j, k) = \frac{r_{hk} \cdot p_h(i,j)}{R_k}$ for resources $k \in \mathscr{R}$. In this way we obtain an instance $I(i, j, k)$ of the preemptive single-machine problem for each pair of activities i, j and each resource $k \in \mathscr{R}$. If the resulting minimum makespan $C_{max}^*(i, j, k)$ is larger than d_{ij}^{cs}, we can put d_{ij}^{cs} to $C_{max}^*(i, j, k)$. The complete method is shown in Algorithm 13.2.

The procedure only considers pairs (i, j) and activities h for which Eq. (13.20) gives a positive processing time $p_h(i, j)$. The time lags are updated until the

Algorithm 13.2: Single-machine preprocessing

```
Require: Instance $I$ of problem $PS \mid temp, pmtn \mid f$, time lag matrix $D^{cs}$
Ensure: Tighten time lag matrix $D^{cs}$

repeat
  set $stop :=$ true;
  for all $i, j \in V : d^{cs}_{ij} > 0$ do
    set $V(i,j) := \{h \in V \mid \min\{d^{cs}_{ih}, d^{cs}_{hj}\} + p_h > 0\}$;
    for all $h \in V(i,j)$ do
      set release date $r_h(i,j) := d^{cs}_{ih}$ and quarantine time $q_h(i,j) := d^{cs}_{hj}$;
    end for
    for all $k \in \mathscr{R}$ do
      for all $h \in V(i,j)$ do
        set processing time
        $p_h(i,j,k) := \frac{r_{hk}}{R_k} \cdot \min\{p_h, d^{cs}_{ij}, d^{cs}_{ih} + p_h, d^{cs}_{hj} + p_h\}$;
      end for
      determine makespan $C^*_{max}(i,j,k)$ of optimum Schrage schedule;
      if $C^*_{max}(i,j,k) > d^{cs}_{ij}$ then
        set $d^{cs}_{ij} := C^*_{max}(i,j,k)$ and $stop :=$ false;
      end if
    end for
  end for
  if not $stop$ then
    restore transitivity of $D^{cs}$ by executing lines 10–14 of Algorithm 13.1;
  end if
until $stop$;
```

sequence of matrices D^{cs} reached a fixed point. The time complexity of one iteration of the repeat-until loop is $\mathscr{O}(Kn^3 \log n)$, where $K = |\mathscr{R}|$. After each iteration, we restore the transitivity of matrix D^{cs} with the modified Floyd-Warshall algorithm. In doing so, further time lags may be strengthened.

Applied to the project of Example 13.3, Algorithm 13.2 increases the time lags d^{cs}_{0h} from 4 to 4.5, d^{cs}_{0j} from 5 to 5.5, $d^{cs}_{0j_2'}$ from 2 to 2.5, and d^{cs}_{08} from 6 to 7.5. Time lag d^{cs}_{08} also provides the new lower bound of 7.5 for the minimum project duration.

The second method for tightening matrix D^{cs} considers situations in which two incompatible activities g and h have to be partly or entirely carried out between the completion of an activity i and the start of an activity j. Since g and h are incompatible, the time lag between the start of the first and the completion of the second will always be greater than or equal to $p_g + p_h$, independently of the actual timing of the activities. Hence, the completion-to-start time lag between i and j is bounded from below by the time lag $\min\{d^{cs}_{ig}, d^{cs}_{ih}\}$ between the completion of i and the start of one of the two activities, plus the durations $p_g + p_h$ of both activities, plus the time lag $\min\{d^{cs}_{gj}, d^{cs}_{hj}\}$ between the completion of one of the two activities

Algorithm 13.3: Disjunctive preprocessing

```
Require: Instance $I$ of problem $PS \mid temp, pmtn \mid f$, time lag matrix $D^{cs}$
Ensure: Tighten time lag matrix $D^{cs}$

repeat
    put $stop$ := true;
    for all $i, j \in V$ do
        for all incompatible $g, h \in V \setminus \{i, j\} : g < h$ do
            if $d^{cs}_{ij} < \min\{d^{cs}_{ig}, d^{cs}_{ih}\} + p_g + p_h + \min\{d^{cs}_{gj} + d^{cs}_{hj}\}$ then
                set $d^{cs}_{ij} := \min\{d^{cs}_{ig}, d^{cs}_{ih}\} + p_g + p_h + \min\{d^{cs}_{gj}, d^{cs}_{hj}\}$ and
                $stop$ := false;
            end if
        end for
    end for
    if not $stop$ then
        restore transitivity of $D^{cs}$ by executing lines 10–14 of Algorithm 13.1;
    end if
until $stop$;
```

and the start of j. Algorithm 13.3 iterates pairs (i, j) and incompatible activities g and h until no further time lag d^{cs}_{ij} can be updated. If the incompatibilities between activities $g, h \in V$ are evaluated before starting the iterations, the time complexity of one iteration of the repeat-until loop is $\mathscr{O}(n^4)$.

For the project of Example 13.3, Algorithm 13.3 allows an additional increase in time lag $d^{cs}_{0j'_2}$ from 2.5 to $\min\{d^{cs}_{0i_1}, d^{cs}_{0j'_1}\} + p_{i_1} + p_{j'_1} + \min\{d^{cs}_{i_1j'_2}, d^{cs}_{j'_1j'_2}\} = 3$.

13.6.2 Efficient Feasibility Tests

In Sect. 13.4.2 we have proposed necessary feasibility conditions and respective mixed-integer linear programs for evaluating these conditions. In what follows, we discuss a procedure implementing three simple feasibility tests, which may also be applied to large test instances. The first two tests evaluate a weakened version of condition (13.5) for which set $\mathscr{F}'$ is restricted to all two-element or three-element sets U of pairwise incompatible activities, respectively. The third test is based on condition (13.6), but instead of taking the maximum over all forbidden sets U, only the three-element nondominated forbidden sets are considered (we say that a forbidden set is nondominated if it does not include any proper subset that is forbidden). The three feasibility tests are summarized in Algorithm 13.4; the symbols $\mathscr{F}_2$ and $\mathscr{F}_3$ stand for the sets of all two-element and nondominated three-element forbidden sets. The time complexity of the algorithm is $\mathscr{O}(Kn^3)$.

The two procedures for tightening the time lags and the feasibility tests can be combined into the preprocessing method displayed in Algorithm 13.5. The

Algorithm 13.4: Feasibility tests

```
Require: Instance $I$ of problem $PS \mid temp, pmtn \mid f$, time lag matrix $D^{cs}$
Ensure: If false is returned, $I$ is infeasible
for all $i \in V$ do
    if $d_{ii}^{cs} > -p_i$ then
        return false;
    end if
end for
generate sets $\mathscr{F}_2$ and $\mathscr{F}_3$;
for all $i, j \in V : i < j$ do
    set $d_{ij}^{min} := \min\{d_{ii}^{cs}, d_{ij}^{cs}, d_{ji}^{cs}, d_{jj}^{cs}\}$;
    if $\{i, j\} \in \mathscr{F}_2$ and $p_i + p_j + d_{ij}^{min} > 0$ then
        return false;
    end if
end for
for all $h, i, j \in V : h < i < j$ do
    set $d_{hij}^{min} := \min\{d_{hi}^{min}, d_{hj}^{min}, d_{ij}^{min}\}$;
    if $\{h, i\}, \{h, j\}, \{i, j\} \in \mathscr{F}_2$ and $p_h + p_i + p_j + d_{hij}^{min} > 0$ then
        return false;
    else if $\{h, i, j\} \in \mathscr{F}_3$ and $p_h + p_i + p_j + 2 \cdot d_{hij}^{min} > 0$ then
        return false;
    end if
end for
return true;
```

algorithm iterates the individual procedures until either matrix D^{cs} constitutes a fixed point of the method or the tested problem instance I has been shown to be infeasible. Our computational experiments with small and medium-sized instances of the project duration problem $PS \mid temp, pmtn \mid C_{max}$ indicate that the method is able to reliably identify infeasible problem instances (see Sect. 13.7).

13.6.3 Fixing Variables

Consider a feasible instance I of the resource-relaxation of problem (P), in which the resource constraints (13.1) have been deleted, and assume that the objective function f to be minimized is regular, i. e., nondecreasing in the completion times C_i of the activities. Obviously, the unique schedule $\sigma(EC)$ completing all activities at their earliest completion times $EC_i = d_{0i}^{cs} + p_i$ represents an optimal solution to I because this schedule is feasible and no activity can be finished earlier.

Algorithm 13.5: Preprocessing

```
Require: Instance $I$ of problem $PS \mid temp, pmtn \mid f$
Ensure: Compute and tighten time lag matrix $D^{cs}$; if false is returned, $I$ is
    infeasible
compute initial time lag matrix $D^{cs}$ (Algorithm 13.1);
if feasibility tests (Algorithm 13.4) return false then
    return false;
end if
while single-machine or disjunctive preprocessing steps
(Algorithms 13.2, 13.3) tighten matrix $D^{cs}$ do
    if feasibility tests (Algorithm 13.4) return false then
        return false;
    end if
end while
return true;
```

The idea of the following procedure consists in fixing all decision variables of MILP model (P'') referring to a time interval during which there is no need for delaying the completion of an activity to resolve a resource conflict. Given vector $EC = (EC_i)_{i \in V}$, we determine the earliest point in time t at which schedule $\sigma(EC)$ violates the resource constraints (13.1) for some resource $k \in \mathscr{R}$. The subproblem defined on the workload that is processed by time t is solved to optimality when executing the activities according to schedule $\sigma(EC)$. To obtain a feasible schedule after time t, it will be necessary to delay the completion of some activities i with respect to their earliest completion times EC_i. Such a displacement may, however, cause the start of other activities j with $EC_j < t$ to be delayed if $-\infty < d_{ij}^{cs} < 0$. That is why we can only lock the schedule up to time $t' = \min\{t, \min\{EC_j - p_j \mid d_{ij}^{cs} > -\infty$ for some $i \in V$ with $EC_i > t\}\}$. This part of schedule $\sigma(EC)$ is frozen by fixing the corresponding decision variables $\Delta x_{i\mu}$, $y_{i\mu}$, and z_μ for all activities i with $EC_i - p_i < t'$ and all slice indices $\mu = 1, \ldots, \nu$ with $\sum_{\lambda=1}^{\mu} z_\lambda \leq t'$ to their respective values in $\sigma(EC)$. We note that this procedure allows to fix all decision variables in cases where schedule $\sigma(EC)$ is resource-feasible and that for instances containing only ordinary precedence relations, the right end t' of the frozen zone always coincides with the earliest conflict time t.

13.6.4 Column-Generation Based Lower Bound

The computation time of an implicit enumeration algorithm may be significantly reduced if effective lower bounds on the minimum objective function values are available. In this section we sketch a column-generation procedure used by Damay et al. (2007) to compute the values of a lower bound on the minimum project duration that was initially proposed by Mingozzi et al. (1998, see Chap. 3 of this handbook for details).

We start from the representation of problem (P) as the sizing-and-sequencing problem (P') introduced in Sect. 13.4.1. By removing the sequencing part of the problem, we obtain the disjunctive relaxation of the problem as the following linear program, containing a nonnegative decision variable z_μ for each feasible antichain $A \in \overline{\mathscr{A}}$.

$$(LP) \left\{ \begin{array}{lll} \text{Min. } C_{max} = \displaystyle\sum_{A \in \overline{\mathscr{A}}} z_A & & \\ \text{s.t.} \quad \displaystyle\sum_{A \in \overline{\mathscr{A}}: i \in A} z_A = p_i & (i \in V) & \\ \qquad z_A \geq 0 & (A \in \overline{\mathscr{A}}) & \end{array} \right. \tag{13.21}$$

Mingozzi et al. (1998) considered the relaxed variant of (LP) in which Eq. (13.21) is replaced by the respective $\geq$-inequality and only nondominated (i. e., $\subseteq$-minimal) antichains are taken into account. In this way, the number of decision variables is markedly reduced but still remains exponential in the number n of activities. In what follows, we will explain how (LP) can be solved within a reasonable amount of time using a (delayed) column-generation procedure.

The basic idea of the method consists in deferring the generation of the columns of the coefficient matrix to the point in time when the respective column enters the basis. The algorithm first computes some initial basic solution to the linear program. In each iteration it then determines a nonbasic variable with negative reduced cost by solving an appropriate pricing problem and performs a simplex pivot entering this nonbasic variable into the basis. The procedure is terminated when all reduced costs are nonnegative and hence the current basic solution is optimal.

Let B be some basic matrix to (LP) and $z^B = B^{-1} \cdot p$ with $p = (p_i)_{i \in V}$ be the corresponding basic solution. The simplex multipliers u_i associated to B are obtained as $u = (B^\top)^{-1} \cdot \mathbb{1}$ with $\mathbb{1} = (1, 1, \ldots, 1)^\top$. From the constraint in the dual

$$(D) \left\{ \begin{array}{ll} \text{Max. } \displaystyle\sum_{i \in V} p_i u_i & \\ \text{s.t.} \quad \displaystyle\sum_{i \in A} u_i \leq 1 & (i \in V) \end{array} \right.$$

of (LP) it follows that the reduced cost of variable z_A with respect to multipliers u is $\rho_A = 1 - \sum_{i \in A} u_i$. Accordingly, a sufficient optimality condition for z^B is $\min_{A \in \overline{\mathscr{A}}} \rho_A = 0$. For given vector u of simplex multipliers for z^B, the pricing problem $(PP(u))$ consists in computing a (nonbasic) variable z_{A^*} with minimum reduced cost ρ_{A^*}. Let w_i denote a binary indicator variable with $w_i = 1$ if and only

Algorithm 13.6: Column-generation method

```
Require: Disjunctive relaxation ($LP$) for instance $I$ of problem
    $PS|temp, pmtn|C_{max}$
Ensure: Optimal solution $z$ proving lower bound $LB = \sum_{A \in \overline{\mathscr{A}}} z_A$

  determine earliest completion vector $EC$ subject to ordinary precedence
  relations $(i, j) \in \Theta(D^{cs})$ and construct corresponding basic matrix $B$;
  repeat
     compute basic solution $z = B^{-1} \cdot p$ and simplex multipliers $u = (B^{\top})^{-1} \cdot \mathbb{1}$;
     determine solution $w^*$ to pricing problem ($PP(u)$) and set
     $A^* := \{i \in V | w_i^* = 1\}$;
     if $\rho_{A^*} < 0$ then
        compute $\gamma = B^{-1} \cdot w^*$ and choose $A' \in \text{argmin}\{\frac{z_A}{\gamma_A} \mid \gamma_A > 0\}$;
        replace column $A^*$ in $B$ by column $A'$;
     end if
  until $\rho_{A^*} = 0$;
```

if $i \in A^*$. The pricing problem now can be stated as the following binary linear program.

$$(PP(u)) \begin{cases} \text{Min. } \rho_{A^*} = 1 - \sum_{i \in V} u_i w_i & \\ \text{s.t. } \; w_i + w_j \leq 1 & ((i,j) \in \Theta(D^{cs})) \quad (13.22) \\ \quad \sum_{i \in V'} r_{ik} w_i \leq R_k & (k \in \mathscr{R}) \quad (13.23) \\ \quad w_i \in \{0, 1\} & (i \in V) \end{cases}$$

Constraints (13.22) and (13.23) ensure that a solution w to ($PP(u)$) encodes a feasible antichain A of preorder $\Theta(D^{cs})$, see Remark 13.2. As was already noticed by Damay et al. (2007), program ($PP(u)$) can be interpreted as an ($\mathscr{NP}$-hard) multidimensional knapsack problem. However, our computational experience shows that the pricing problem is solved within a split second with commercial MILP solvers.

Algorithm 13.6 summarizes the column-generation procedure for solving linear program (LP). For the project of Example 13.3, the method provides the tight lower bound on the minimum duration of 8.5 time units within six iterations.

Finally, we note that the disjunctive relaxation (LP) of (P) can be strengthened by taking into account that the transitive completion-to-start time lags d_{ij}^{cs} define upper bounds on the maximum execution times of antichains in time-feasible schedules. For example, if an antichain A contains two activities i and j, its duration z_A is bounded from above by $\min\{-d_{ij}^{cs}, -d_{ji}^{cs}\}$. This observation can be exploited to add the following relaxation of inequality (13.19) to linear program

(LP):

$$\sum_{A \in \overline{\mathscr{A}}:i,j \in A} z_A \leq -d_{ij}^{cs} \quad (i,j \in V : 0 < -d_{ij}^{cs} < \min\{p_i, p_j\})$$

13.7 Computational Results

We tested the efficiency of the MILP formulation (P'') of problem (P) using two standard test sets from literature. The model was coded under GAMS 24.0.1 invoking CPLEX 12.5 as MILP solver. The computational experiments were performed on a dual-core PC with 3.16 GHz clock pulse and 3 GB RAM operating under Windows 7. The solver was stopped after a computation time limit of 300 s. All problem instances were preprocessed by Algorithm 13.5 and lower bounds were computed by the column-generation method of Algorithm 13.6. The preprocessing methods were implemented in C# under MS Visual Studio 2010, whereas the column-generation procedure was coded under GAMS.

Table 13.2 summarizes the results that we obtained for the 480 KSD-30 instances of problem $PS \mid prec, pmtn \mid C_{max}$ containing 30 activities and four resources each (see Kolisch and Sprecher 1996). The hardness of the instances is largely influenced by the resource strength parameter RS, which respectively equals 0.2, 0.5, 0.7, or 1.0 for 120 of the 480 instances; the smaller RS, the scarcer are the resources. The columns p_{opt} and p_{opt}^{fnd} display the percentages of instances for which the enumeration was completed within the imposed time limit (i. e., the schedule found was proved to be optimal) and for which an optimal schedule was found, respectively. $\Delta_{opt}^{\varnothing}$ and Δ_{nonp} designate the mean relative deviations of the makespans from the optimum project durations of the preemptive problem (communicated by Damay 2011) and the non-preemptive problem (taken from the PSPLIB website). The next two columns show the maximum reduction Δ_{nonp}^{min} in project duration obtained by our model as compared to the non-preemptive problem and the corresponding percentage p_{imp} of instances with positive preemption gains. The last column provides the mean number n_{int} of activity interruptions in the computed schedules. In sum, the model is able to solve more than two thirds of the instances to optimality within 5 min of computation time. The remaining optimality

Table 13.2 Performance of MILP model for the KSD-30 instances

	p_{opt}	p_{opt}^{fnd}	$\Delta_{opt}^{\varnothing}$	Δ_{nonp}	Δ_{nonp}^{min}	p_{imp}	n_{int}
$RS = 0.2$	10.0 %	31.7 %	2.5 %	−0.6 %	−8.8 %	46.7 %	14.2
$RS = 0.5$	40.0 %	60.0 %	0.7 %	−1.7 %	−7.3 %	57.5 %	11.4
$RS = 0.7$	80.0 %	87.5 %	0.2 %	−0.9 %	−6.7 %	27.5 %	8.2
$RS = 1.0$	100.0 %	100.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0
Total	57.5 %	69.8 %	0.8 %	−0.8 %	−8.8 %	32.9 %	8.5

Table 13.3 Performance of the MILP model for the UBO-10 instances

	p_{opt}	p_{inf}	p_{feas}	p_{unk}	p_{opt}^{fnd}	$\Delta_{LB}^{\varnothing}$	Δ_{nonp}	p_{imp}	n_{int}
$RS = 0.0$	20.0 %	20.0 %	60.0 %	0.0 %	43.3 %	3.9 %	−1.9 %	36.7 %	1.4
$RS = 0.25$	23.3 %	10.0 %	66.7 %	0.0 %	50.0 %	3.4 %	−2.7 %	43.3 %	2.8
$RS = 0.5$	56.7 %	13.3 %	30.0 %	0.0 %	60.0 %	1.6 %	−1.9 %	33.3 %	1.8
Total	33.3 %	14.4 %	52.2 %	0.0 %	51.1 %	2.9 %	−2.2 %	37.8 %	2.0

gap of 0.8 % is relatively small. Even if on average the decrease in the project duration is only minor, there exist projects for which almost 9 % of the scheduled lead time can be saved by considering activity preemptions. All instances with $RS = 1.0$ were already settled by the variable-fixing procedure given that schedule $\sigma(EC)$ is feasible.

The computed schedules contained a mean number of 21.3 positive slices, from which 8.1 were frozen by the variable-fixing algorithm. The column-generation procedure solved the instances of linear program (LP) within 11.9 s and 73.0 iterations on average. The mean gap between the resulting lower bounds and the optimum objective function values equals 2.0 % (it is reduced to 1.6 % if the lower bound $d_{0(n+1)}^{cs}$ provided by the preprocessing is also taken into account), and the bound is tight for 57.5 % of all instances.

To test the performance of the model for the case of generalized precedence relations we used the UBO-10 and UBO-20 sets of instances with 10 and 20 activities and five resources each introduced by Franck et al. (2001a). These instances are much harder to solve, which is due to the tighter resource constraints and the presence of minimum and maximum time lags. Since not all instances are feasible, we also provide the percentages p_{inf} and p_{unk} of instances for which the model could prove infeasibility and for which the feasibility status remained unknown. p_{feas} denotes the percentage of instances for which a feasible solution was found, but the enumeration could not be completed within the time limit. The mean relative deviation $\Delta_{LB}^{\varnothing}$ of the computed project durations from the lower bounds refer to the maximum of the two lower bounds obtained with the preprocessing and with the column-generation procedure.

The results for the UBO-10 instances are shown in Table 13.3. On the one hand, the proportion of instances for which the enumeration could be completed within the time limit (47.8 %) is smaller than for the larger KSD-30 instances, on the other hand the improvement with respect to the non-preemptive optimal solutions is superior compared to the case of ordinary precedence relations. For each instance, either a feasible schedule could be found or the instance was shown to be infeasible by the infeasibility tests. The maximum savings in project duration enabled by activity interruptions amount to $-\Delta_{nonp}^{min} = 16.7\,\%$.

On average, the decision variables of 1.21 slices were fixed in advance, the computed schedules comprise 10.2 positive slices, and column generation took 4.4 s and 17.7 iterations to solve the disjunctive relaxation (LP).

Table 13.4 Performance of the MILP model for the UBO-20 instances

	p_{opt}	p_{inf}	p_{feas}	p_{unk}	p_{opt}^{fnd}	$\Delta_{LB}^{\varnothing}$	Δ_{nonp}	p_{imp}	n_{int}
$RS = 0.0$	3.3 %	16.7 %	80.0 %	0.0 %	13.3 %	7.5 %	−3.0 %	63.3 %	3.6
$RS = 0.25$	16.7 %	6.7 %	76.7 %	0.0 %	23.3 %	7.8 %	−1.3 %	60.0 %	5.7
$RS = 0.5$	50.0 %	6.7 %	43.3 %	0.0 %	50.0 %	3.7 %	−0.5 %	46.7 %	4.6
Total	23.3 %	10.0 %	66.7 %	0.0 %	28.9 %	6.3 %	−1.4 %	56.7 %	4.6

The results for the instances of the UBO-20 set are listed in Table 13.4. As compared to the smaller instances of the UBO-10 set, the number of instances for which the solver stopped before the runtime limit was reached decreases, from 47.8 to 33.3 %, and the mean percentage gap between the lower bounds and the computed makespans increases, from 2.9 to 6.3 %. Nevertheless, we were still able to identify all infeasible instances by applying the feasibility test, and for each feasible instance the solver found a feasible schedule. Interestingly, the percentage of instances for which preemption pays is increased, from 37.8 to 56.7 %. The maximum realized preemption gains are $-\Delta_{nonp}^{min} = 12.5\,\%$.

Only a small fraction of the slices could be frozen by the variable-fixing method. On average, the computed schedules consist of 21.5 positive slices, 1.22 of which were settled before the enumeration started. The column-generation method solved the linear program within a mean computation time of 19.7 s and 72.7 iterations.

13.8 Conclusions

Preemptive resource-constrained project scheduling problems arise in the context of hierarchical project planning and in different project scheduling applications like production planning or computer-system operation. In this chapter we provided a survey on different types of preemptive project scheduling problems and respective solution approaches that were discussed in the literature. Furthermore, we proposed a generic preemptive scheduling model with continuous preemption and generalized feeding precedence relations, for which we addressed several structural issues relating to the representation of solutions, feasibility conditions, preemption gains, number of schedule slices, and required number of activity interruptions. Based on the reduction to a canonical form, in which all precedence relations are expressed in terms of completion-to-start time lags, we developed a novel MILP formulation for preemptive and non-preemptive scheduling problems. To enhance the performance of MILP solvers on instances of the model, we devised several preprocessing methods and explained how effective lower bounds can be computed via column-generation techniques. Computational experience with the model on different benchmark data sets from literature shows that it is possible to solve small and medium-sized problem instances with up to 30 activities within a reasonable amount of time and a moderate optimality gap.

There are various avenues for future research on complex preemptive scheduling problems. For example, open issues include tight upper bounds on the maximum number of interruptions and on the maximum preemption gains for feasible instances. A question that we were not able to settle is related to the structure of efficient solutions. We conjecture that for any feasible instance of the scheduling problem formulated in canonical form there exists an optimal schedule for which at each slice end, at least one activity is started or completed. The design of an efficient enumeration scheme following the general approach by Damay et al. (2007) also deserves further scrutiny. Such a scheme would rely on the decomposition of the problem into a disjunctive relaxation for slice-sizing and a slice-sequencing problem. The disjunctive relaxation can be formulated in such a way that all prescribed precedence relations are represented via dummy activities and the temporal constraints can be expressed as linear inequalities in binary variables. Given an optimal solution to the disjunctive relaxation, the existence of a feasible schedule containing precisely the slices corresponding to the optimal basis could then be checked by solving the sequencing problem for fixed slice sizes. In case of infeasibility of the sequencing problem, branching over forbidden activity sets would provide a complete enumeration scheme. Finally, further research could be oriented towards an appropriate concept of active schedules in the presence of continuous preemptions. To the best of our knowledge, the concepts that were proposed for complex scheduling problems give rise to an uncountable set of active schedules. New concepts leading to a finite set of efficient solutions could be a stepping stone for future advancements in exact and heuristic solution procedures for complex preemptive scheduling problems.

Acknowledgements The authors are indebted to Dr. Jean Damay for providing the benchmark results of the KSD-30 instances.

References

Alfieria A, Toliob T, Urgo M (2011) A project scheduling approach to production planning with feeding precedence relations. Int J Prod Res 49:995–1020

Ballestín F, Valls V, Quintanilla S (2008) Pre-emption in resource-constrained project scheduling. Eur J Oper Res 189:1136–1152

Ballestín F, Valls V, Quintanilla S (2009) Scheduling projects with limited number of preemptions. Comput Oper Res 36:2913–2925

Baptiste P, Demassey S (2004) Tight LP bounds for resource constrained project scheduling. OR Spectr 26:251–262

Baptiste P, Carlier J, Kononov A, Queyranned M, Sevastyanov S, Sviridenko M (2004) Structural properties of preemptive schedules. IBM Research Report, IBM T.J. Watson Research Center, Yorktown Heigths. Available at http://www.research.ibm.com/people/s/sviri/papers/structure27.pdf. Cited 8 Feb 2014

Baptiste P, Carlier J, Kononov A, Queyranned M, Sevastyanov S, Sviridenko M (2011) Properties of optimal schedules in preemptive shop scheduling. Discrete Appl Math 159:272–280

Bianco L, Caramia M, Dell'Olmo P (1999) Solving a preemptive project scheduling problem with coloring techniques. In: Węglarz J (ed) Project scheduling: recent models, algorithms, and applications. Kluwer Academic Publishers, Boston, pp 135–145

Błażewicz J, Ecker KH, Pesch E, Schmidt G, Węglarz J (2007) Handbook on scheduling: from theory to applications. Springer, Berlin

Braun O, Schmidt G (2003) Parallel processor scheduling with limited number of preemptions. SIAM J Comput 32:671–680

Brucker P, Knust S (2000) A linear programming and constraint propagation-based lower bound for the RCPSP. Eur J Oper Res 127:355–362

Buddhakulsomsiri J, Kim DS (2006) Properties of multi-mode resource-constrained project scheduling problems with resource vacations and activity splitting. Eur J Oper Res 175:279–295

Buddhakulsomsiri J, Kim DS (2007) Priority rule-based heuristic for multi-mode resource-constrained project scheduling problems with resource vacations and activity splitting. Eur J Oper Res 178:374–390

Carlier J (1982) The one-machine sequencing problem. Eur J Oper Res 11:42–47

Damay J (2008) Preemptive activities. In: Artigues C, Demassey S, Néron E (eds) Resource-constrained project scheduling: models, algorithms, extensions and applications. Wiley, Hoboken, pp 139–147

Damay J (2011) Personal communication

Damay J, Quilliot A, Sanlaville E (2007) Linear programming based algorithms for preemptive and non-preemptive RCPSP. Eur J Oper Res 182:1012–1022

Demeulemeester EL, Herroelen WS (1996) An efficient optimal solution procedure for the preemptive resource-constrained project scheduling problem. Eur J Oper Res 90:334–348

Floyd RW (1962) Algorithm 97: shortest path. Commun ACM 5:345

Franck B, Neumann K, Schwindt C (1997) A capacity-oriented hierarchical approach to single-item and small-batch production planning using project-scheduling methods. OR Spektrum 19:77–85

Franck B, Neumann K, Schwindt C (2001a) Truncated branch-and-bound, schedule-construction, and schedule-improvement procedures for resource-constrained project scheduling. OR Spektrum 23:297–324

Franck B, Neumann K, Schwindt C (2001b) Project scheduling with calendars. OR Spektrum 23:325–334

Hartmann S, Drexl A (1998) Project scheduling with multiple modes: a comparison of exact algorithms. Networks 32:283–298

Heilmann R, Schwindt C (1997) Lower bounds for RCPSP/max. Technical Report WIOR-511, University of Karlsruhe, Germany

Kaplan L (1988) Resource-constrained project scheduling with preemption of jobs. Ph.D. dissertation, University of Michigan, Ann Arbor

Kis T (2005) A branch-and-cut algorithm for scheduling of projects with variable-intensity activities. Math Program 103:515–539

Kolisch R, Sprecher A (1996) PSPLIB: a project scheduling library. Eur J Oper Res 96:205–216

Li F, Lai C, Shou Y (2011) Particle swarm optimization for preemptive project scheduling with resource constraints. In: Proceedings of the 2011 IEEE international conference on industrial engineering and engineering management (IEEM), Singapore, pp 869–873

Mingozzi A, Maniezzo V, Ricciardelly S, Bianco L (1998) An exact algorithm for the resource-constrained project scheduling based on a new mathematical formulation. Manage Sci 44:714–729

Nadjafi BA, Shadrokh S (2008) The preemptive resource-constrained project scheduling problem subject to due dates and preemption penalties: an integer programming approach. J Ind Eng 1:35–39

Neumann K, Schwindt C, Zimmermann J (2003) Project scheduling with time windows and scarce resources. Springer, Berlin

Nudtasomboon N, Randhawa SU (1997) Resource-constrained project scheduling with renewable and non-renewable resources and time-resource tradeoffs. Comput Ind Eng 32:227–242

Quintanilla S, Pérez A, Lino P, Valls V (2012) Time and work generalised precedence relatioships in project scheduling with pre-emption: an application to the management of Service Centres. Eur J Oper Res 219:59–72

Richter LK, Yano CA (1986) A comparison of heuristics for preemptive resource-constrained project scheduling. Technical Report 86–39, Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor

Schwindt C (2005) Resource allocation in project management. Springer, Berlin

Słowiński R (1978) A node ordering heuristic for network scheduling under multiple resource constraints. Found Control Eng 3:19–27

Słowiński R (1980) Two approaches to problems of resource allocation among project activities: a comparative study. J Oper Res Soc 31:711–723

Vanhoucke M (2008) Setup times and fast tracking in resource-constrained project scheduling. Comput Ind Eng 54:1062–1070

Vanhoucke M, Demeulemeester E, Herroelen W (2002) Discrete time/cost tradeoffs in project scheduling with time-switch constraints. J Oper Res Soc 53:741–751

Van Peteghem V, Vanhoucke M (2010) A genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem. Eur J Oper Res 201:409–418

Wikum ED, Llewellyn DC, Nemhauser GL (1994) One-machine generalized precedence constrained scheduling problems. Oper Res Lett 16:87–99

Yang HH, Chen YL (2000) Finding the critical path in an activity network with time-switch constraints. Eur J Oper Res 120:603–613

Part V
Non-Regular Objectives in Project Scheduling

Chapter 14
Exact and Heuristic Methods for the Resource-Constrained Net Present Value Problem

Hanyu Gu, Andreas Schutt, Peter J. Stuckey, Mark G. Wallace, and Geoffrey Chu

Abstract An important variant of the resource-constrained project scheduling problem is to maximise the net present value. Significant progress has been made recently on this problem for both exact and inexact methods. The lazy clause generation based constraint programming approach is the state of the art among the exact methods and is briefly discussed. The performance of the Lagrangian relaxation based decomposition method is greatly improved when the forward-backward improvement heuristic is employed. A novel decomposition approach is designed for very large industrial problems which can make full use of the parallel computing capability of modern personal computers. Computational results are also presented to compare different approaches on both difficult benchmark problems and large industrial applications.

Keywords Constraint programming • Lagrangian relaxation • Net present value • Project scheduling • Resource constraints

H. Gu (✉)
School of Mathematical Sciences, University of Technology, Sydney, Australia
e-mail: Hanyu.Gu@uts.edu.au

A. Schutt • P.J. Stuckey • G. Chu
National ICT Australia & Computing and Information Systems, University of Melbourne, Melbourne, Australia
e-mail: andreas.schutt@nicta.com.au; peter.stuckey@nicta.com.au; Geoffrey.Chu@unimelb.edu.au

M.G. Wallace
Faculty of Information Technology, Monash University, Caulfield, Australia
e-mail: mark.wallace@monash.edu

C. Schwindt, J. Zimmermann (eds.), *Handbook on Project Management and Scheduling Vol. 1*, International Handbooks on Information Systems,
DOI 10.1007/978-3-319-05443-8_14

14.1 Introduction

The resource-constrained project scheduling problem (RCPSP) is one of the most studied scheduling problems. It consists of scarce resources, activities, and precedence constraints between pairs of activities where a precedence constraint expresses that an activity can be run after the execution of its preceding activity is finished. Each activity requires some units of resources during their execution. The aim is to build a schedule that satisfies the resource and precedence constraints. Here, we assume renewable resources (i.e., their supply is constant during the planning period) and non-preemptive activities (i.e., once started their execution cannot be interrupted). Usually the objective in solving resource-constrained project scheduling problems is to minimise the makespan, i.e., to complete the entire project in the minimum total time. But another important objective is to maximise the net present value (*npv*), because it better captures the financial aspects of the project. In this formulation each activity has an associated cash flow which may be a payment (negative cash flow) or a receipt (positive cash flow). These cash flows are discounted with respect to a discount rate, which makes it, in general, beneficial for the *npv* to execute activities with a positive (negative) cash flow as early (late) as possible. The problem is to maximise the *npv* for a given RCPSP problem. We denote the problem RCPSPDC, i.e., RCPSP with discounted cash flows. It is classified as $PS|prec, \overline{d}| \sum c_i^F \beta^{C_i}$ in Brucker et al. (1999).

Optimisation of the net present value for project scheduling problems was first introduced in Russell (1970). Different exact and inexact methods for RCPSPDC with or without generalised precedence constraints have been proposed, the reader is referred to Hartmann and Briskorn (2010) for a more extensive literature overview of solution approaches for RCPSPDC and other variants or extensions of RCPSP.

Most exact methods for RCPSPDC use a branch-and-bound algorithm to maximise the *npv*. The approaches in Icmeli and Erengüç (1996), Vanhoucke et al. (2001) and Neumann and Zimmermann (2002) are based on the branch-and-bound algorithm in Demeulemeester and Herroelen (1992, 1997) for RCPSP. These algorithms use a schedule-generation scheme which resolves resource conflicts by adding new precedence constraints between activities in conflict. The method in Vanhoucke et al. (2001) improves upon the one in Icmeli and Erengüç (1996) whereas the work of Neumann and Zimmermann (2002) considers RCPSPDC with generalised precedence constraints. Recently Schutt et al. (2012) developed the lazy clause generation based approaches which provide the state of the art exact method for RCPSPDC.

But exact methods can only solve problems up to one hundred activities. In contrast various meta-heuristics can produce good solutions with very little CPU time. The evolutionary population based scatter search algorithm (Vanhoucke 2010) achieved the best results in comparison with other meta-heuristics such as genetic algorithms (Kolisch and Hartmann 2006) and tabu search (Zhu and Padman 1999) on a comprehensive set of test cases. To cope with large industrial applications with thousands of activities, various rules based heuristics

(Baroum and Patterson 1996; Padman et al. 1997; Selle and Zimmermann 2003) are used in practice. However these approaches are not robust especially for problems with tight constraints.

Decomposition methods are widely used for large-scale combinatorial optimisation problems. Lagrangian relaxation was successfully applied on RCPSP for up to 1,000 jobs (Möhring et al. 2003). It has also been applied to RCPSPDC with good results (Kimms 2001) and is more robust than rule based heuristics. Gu et al. (2013) produced very competitive results for problems with tight deadlines and resource constraints by combining the forward-backward improvement heuristic with Lagrangian relaxation based on the α-point heuristic. For very large industrial problems Gu et al. (2012) proposed a novel decomposition approach that can make full use of the parallel computing capability of modern personal computers.

This chapter is organized as follows. The lazy clause generation based exact method is discussed in Sect. 14.3, after models for the RCPSPDC are introduced in Sect. 14.2. The Lagrangian relaxation method is described in Sect. 14.4 which also includes the forward-backward improvement heuristic using the idea of α-point. The novel decomposition approach for large problems is presented in Sect. 14.5. Computational results are shown in Sect. 14.6, and a summary is given in Sect. 14.7.

14.2 Models for RCPSPDC

In this section we give two models for RCPSPDC, a constraint programming model, and a mixed integer programming model.

The RCPSPDC is defined as follows: A set of activities $V = \{1, \ldots, n\}$ is subject to precedence relations in $E \subset V^2$ between two activities, and scarce resources in $\mathscr{R}$. The goal is to find a schedule $S = (S_i)_{i \in V}$ that respects the precedence and resource constraints, and maximises the *npv* which is equal to $\sum_{i=1}^{n} e^{-\alpha S_i} c_i^F$ where α is the continuous discount rate, S_i is the *start time* of the activity i, and c_i^F is the cash flow that occurs when activity i starts.[1]

Each activity i has a finite *duration* p_i and requires (non-negative) r_{ik} units of resource k, $k \in \mathscr{R}$ for its execution where r_{ik} is the *resource requirement* or *usage* of activity i for resource k. A resource $k \in \mathscr{R}$ has a constant *resource capacity* R_k over the planning period which cannot be exceeded at any point in time. The *planning period* is given by $[0, \overline{d})$ where $\overline{d}$ is the maximal project duration. Each resource k is modeled by a single cumulative constraint (Aggoun and Beldiceanu 1993): $\texttt{cumulative}(S, p, [r_{ik} | i \in V], R_k)$.

The constraint programming model of RCPSPDC problem can be stated as follows:

[1] Note that much of the work on PSPDC and RCPSPDC considers discounted cash flows at the end of activities, and find solutions to end time variables, here we use start times. If the cash flows occur at the completion time instead of the start time, c_i^F has to be replaced by $c_i^F e^{\alpha p_i}$.

$$NPV \;=\; \text{Max.} \sum\nolimits_{i=1}^{n} e^{-\alpha S_i} c_i^F \tag{14.1}$$

$$\text{s. t.} \qquad S_i + p_i \leq S_j \quad ((i,j) \in E) \tag{14.2}$$

$$\texttt{cumulative}(S, p, [r_{ik} | i \in V], R_k) \quad (k \in \mathscr{R}) \tag{14.3}$$

$$0 \leq S_i \leq \overline{d} - p_i \quad (i \in V) \tag{14.4}$$

The objective is to maximise the net present value (Eq. 14.1) which is subject to the precedence constraints (Eq. 14.2), and the resource constraints (Eq. 14.3). All start times must be non-negative and all activities must be scheduled in the planning period (Eq. 14.4).

We now show a MIP model for RCPSPDC. Let w_{jt} be the discounted cash flow of activity j when starting at time t, i.e., $w_{jt} = e^{-\alpha t} c_j^F$. The time-indexed formulation (Doersch and Patterson 1977) is:

$$NPV \;=\; \text{Max.} \sum_j \sum_t w_{jt} x_{jt} \tag{14.5}$$

$$\text{s. t.} \qquad \sum_t x_{jt} = 1 \quad (j \in V) \tag{14.6}$$

$$\sum_{\tau=t}^{\overline{d}} x_{i\tau} + \sum_{\tau=0}^{t+p_i-1} x_{j\tau} \leq 1 \quad ((i,j) \in E;\; t = 0, \ldots, \overline{d}) \tag{14.7}$$

$$\sum_j r_{jk} \Big(\sum_{\tau=t-p_j+1}^{t} x_{j\tau} \Big) \leq R_k \quad (k \in \mathscr{R};\; t = 0, \ldots, \overline{d}) \tag{14.8}$$

$$\text{all variables binary} \tag{14.9}$$

where the binary variable $x_{jt} = 1$ if activity j starts at t ($S_j = t$), and $x_{jt} = 0$ otherwise. The assignment constraints (14.6) ensure that each activity has exactly one start time. The precedence constraints (14.7) imply that activity j cannot start before $t + p_i$ if activity i starts at or after time t for each $(i, j) \in E$. The resource constraints (14.8) enforce that the resource consumption of all activities processed simultaneously must be within the resource capacity.

14.3 Lazy Clause Generation Based Exact Method

Lazy clause generation (LCG) is a sophisticated learning technology in constraint programming (CP). In a CP solver each variable S_i, $1 \leq i \leq n$ has an initial domain of possible values $D^0(S_i)$ which is initially $[0, \overline{d} - p_i]$. The solver maintains a current domain D for all variables. The CP search interleaves propagation with search. The constraints are represented by propagators that, given the current domain D, creates a new smaller domain D' by eliminating infeasible values. For more details on CP see, e.g., Schulte and Stuckey (2008).

For a learning solver we also represent the domain of each variable S_i using Boolean variables $[\![S_i \leq v]\!], 0 \leq v < \overline{d} - p_i$. These are used to track the reasons for propagation and generate nogoods. For more details see Ohrimenko et al. (2009). We use the notation $[\![S_i \geq v]\!], 1 \leq v \leq \overline{d} - p_i$ as shorthand for $\neg[\![S_i \leq v - 1]\!]$, and treat $[\![S_i \geq 0]\!]$ and $[\![S_i \leq \overline{d} - p_i]\!]$ as synonyms for *true*. Propagators in a learning solver must explain each reduction in domain by building a clausal explanation using these Boolean variables.

Efficient propagators for `cumulative` constraints (Schutt et al. 2011) and precedence constraints (Harvey and Stuckey 2003) which explain themselves are well understood.

Optimisation problems are typically solved in CP via branch and bound. Given an objective function f which is to be maximised, when a solution S' is found with objective value $f(S')$, a new constraint $f(S) > f(S')$ is posted to enforce that we only look for better solutions in the subsequent search.

Chapter 7 of this handbook demonstrates how to solve RCPSP with generalised precedence relations very efficiently using LCG. To solve RCPSPDC using LCG we need to build a propagator for the nonlinear objective constraint. Three different implementations were proposed in Schutt et al. (2012) with different strength and computational complexity. Here we only consider the simplest version which solves the max-NPV problem without precedence relations:

$$\begin{aligned} &\text{Max.} \quad \sum\nolimits_{i=1}^{n} e^{-\alpha S_i} c_i^F \\ &\text{s. t.} \quad \min D(S_i) \leq S_i \leq \max D(S_i) \quad (1 \leq i \leq n) \end{aligned}$$

This is easy to solve by simply setting $S_i = \min D(S_i)$ when $c_i^F > 0$ and $S_i = \max D(S_i)$ when $c_i^F < 0$. Define $T^+ = \{i \mid i \in \{1, \ldots, n\}, c_i^F > 0\}$ and $T^- = \{i \mid i \in \{1, \ldots, n\}, c_i^F < 0\}$.

In a branch and bound solution process, we will be trying to improve on a previous solution, and the constraint on the objective can be represented as: $\sum_{i=1}^{n} e^{-\alpha S_i} c_i^F > f^{best}$ where f^{best} is the current best solution found. We can build a propagator directly for this as follows:

Consistency: Calculate $DC = \sum_{i \in T^+} e^{-\alpha \min D(S_i)} c_i^F + \sum_{i \in T^-} e^{-\alpha \max D(S_i)} c_i^F$. If $DC \leq f^{best}$ then fail.

Example 14.1. Consider five activities with cash flows −100, 125, −150, 200, 20 and $\alpha = 0.01$. Suppose the current domain is $D(S_1) = [0, 7]$, $D(S_2) = [3, 8]$, $D(S_3) = [0, 4]$, $D(S_4) = [1, 5]$ and $D(S_5) = [4, 8]$. Then $DC = -100e^{-0.01(7)} + 125e^{-0.01(3)} - 150e^{-0.01(4)} + 200e^{-0.01(1)} + 20e^{-0.01(4)} = 101.17$. Suppose the current best solution is $f^{best} = 105$ then the propagator would trigger backtracking. □

Bounds propagation: If $DC > f^{best}$ then propagate bounds as follows. For each $i \in T^+$, let $DC_i = DC - e^{-\alpha \min D(S_i)} c_i^F$. Let $u_i = \lfloor -\alpha^{-1}(\ln(f^{best} - DC_i) - \ln(c_i^F)) \rfloor$ and set $D(S_i) = D(S_i) \cap [0, u_i]$. Similarly for each $i \in T^-$, let $DC_i = DC -$

$e^{-\alpha \max D(S_i)} c_i^F$ and $l_i = \lceil -\alpha^{-1}(\ln(DC_i - f^{best}) - \ln(-c_i^F)) \rceil$ and set $D(S_i) = D(S_i) \cap [l_i, \overline{d}]$.

Example 14.2. Consider the problem of Example 14.1 but assume the current best solution is $f^{best} = 100$. Then we can propagate bounds. $DC_2 = -20.13$ and hence a new upper bound on S_2 is $u_2 = \lfloor -100(\ln(100 - -20.13) - \ln(125)) \rfloor = \lfloor 3.97 \rfloor = 3$. Similarly $DC_4 = -96.84$ and a new upper bound for S_4 is $u_4 = \lfloor -100(\ln(100 - -96.84) - \ln(200)) \rfloor = \lfloor 1.59 \rfloor = 1$. Now $DC_5 = 81.95$ and a potential new upper bound for S_5 is $u_5 = \lfloor -100(\ln(100 - 81.95) - \ln(20)) \rfloor = \lfloor 10.26 \rfloor = 10$, but this is not lower than its current upper bound so there is no propagation. □

Explanation: The explanation of failure simply uses the appropriate bounds of the variables. If $DC \leq f^{best}$ then we generate explanation $\bigwedge_{j \in T^+} [\![S_j \geq \min D(S_j)]\!] \wedge \bigwedge_{j \in T^-} [\![S_j \leq \max D(S_j)]\!] \rightarrow \mathit{false}$. Similarly for bounds propagation if $DC > f^{best}$ we generate an explanation $\bigwedge_{j \in T^+ - \{i\}} [\![S_j \geq \min D(S_j)]\!] \wedge \bigwedge_{j \in T^-} [\![S_j \leq \max D(S_j)]\!] \rightarrow [\![S_i \leq u_i]\!]$ for changing the upper bound of S_i where $c_i^F > 0$, and $\bigwedge_{j \in T^+} [\![S_j \geq \min D(S_j)]\!] \wedge \bigwedge_{j \in T^- - \{i\}} [\![S_j \leq \max D(S_j)]\!] \rightarrow [\![S_i \geq l_i]\!]$ for changing the lower bound of S_i where $c_i^F < 0$.

Example 14.3. An explanation for the failure of Example 14.1 is $[\![S_1 \leq 7]\!] \wedge [\![S_2 \geq 3]\!] \wedge [\![S_3 \leq 4]\!] \wedge [\![S_4 \geq 1]\!] \wedge [\![S_5 \geq 4]\!] \rightarrow \mathit{false}$. An explanation for the new upper bound of S_4 in Example 14.2 is $[\![S_1 \leq 7]\!] \wedge [\![S_2 \geq 3]\!] \wedge [\![S_3 \leq 4]\!] \wedge [\![S_5 \geq 4]\!] \rightarrow [\![S_4 \leq 1]\!]$. □

Strengthening Explanations: The explanations described above are not necessarily *the strongest possible*, there may be weaker left hand sides of the explanation which still explain the right hand side consequence.

Assume $c_i^F > 0$ and $S_i \leq u_i$ is propagated. Then $DC_i + e^{-\alpha(u_i+1)} c_i^F \leq f^{best}$. Let $r = f^{best} - (DC_i + e^{-\alpha(u_i+1)} c_i^F)$ be the remainder before the bounds propagation will be weakened. We can relax the bounds on the other variables S_i, changing DC and DC_i as long as r remains non-negative. This generates a stronger (more reusable) explanation.

Example 14.4. Consider the explanation for the propagation of $S_4 \leq 1$ in Example 14.2. The remainder before the propagation is $r = 100 - (-96.84 + e^{-0.01(2)} 200) = 0.80$. If we relax the bound on S_5 to 0 the new DC is 101.96. The difference is $101.96 - 101.17 = 0.79$, hence the same upper bound for S_4 holds. Since $S_5 \geq 0$ is universally true we can remove it from the explanation obtaining a stronger explanation $[\![S_1 \leq 7]\!] \wedge [\![S_2 \geq 3]\!] \wedge [\![S_3 \leq 4]\!] \rightarrow [\![S_4 \leq 1]\!]$. Weakening any bound further drops the remainder to negative, so this explanation is as strong as possible. □

The more complex propagators defined in Schutt et al. (2012) take into account precedence relations in propagation. The results in Schutt et al. (2012) comprehensively improve on the previous state of the art complete method for RCPSPDC in Vanhoucke et al. (2001). More explanations about the lazy clause generation technique can also be found in Chap. 7 of this handbook.

14.4 Lagrangian Relaxation Based Forward-Backward Improvement Heuristic

The Lagrangian relaxation method works on the MIP model for RCPSPDC described in Sect. 14.2. Lagrangian relaxation identifies "hard" constraints in the optimisation problem, and removes these "hard" constraints by putting them in the objective function as penalties for the violation of these relaxed constraints (Fisher 1981). The Lagrangian Relaxation Problem (LRP) obtained by relaxing the resource constraints (14.8) with Lagrangian multipliers λ_{kt}, $k \in \mathscr{R}$, $t = 0, \ldots, \overline{d}$ is

$$Z_{LR}(\lambda) \; = \; \text{Max.}\, LRP(x) \tag{14.10}$$

$$\text{s. t.} \qquad (14.6), (14.7), (14.9) \tag{14.11}$$

where

$$LRP(x) = \sum_j \sum_t w_{jt} x_{jt} + \sum_{k \in \mathscr{R}} \sum_t \lambda_{kt} \left(R_k - \sum_j r_{jk} \left(\sum_{\tau = t - p_j + 1}^{t} x_{j\tau} \right) \right) \tag{14.12}$$

By rearranging the terms in Eq. (14.12) we have

$$LRP(x) = \sum_j \sum_t z_{jt} x_{jt} + \sum_{k \in \mathscr{R}} \sum_t \lambda_{kt} R_k \tag{14.13}$$

where

$$z_{jt} = w_{jt} - \sum_{\tau = t}^{t + p_j - 1} \sum_{k \in \mathscr{R}} \lambda_{k\tau} r_{jk} \tag{14.14}$$

The Lagrangian multiplier $\lambda_{k\tau}$ can be interpreted as the unit price for using resource k at time period τ. The discounted cash flow of activity j starting at time t is then further reduced in Constraints (14.14) by the amount paid for all the resources used from the start to the completion of this activity. It is well-known (Fisher 1981) that $Z_{LR}(\lambda)$ is a valid upper bound of RCPSPDC for $\lambda \geq 0$.

The polytope described by (14.6), (14.7), and (14.9) is integral (Chaudhuri et al. 1994). However, it is inefficient to solve LRP using a general LP solver. Instead, it can be transformed into a minimum cut problem (Möhring et al. 2003) and solved efficiently by a general max-flow algorithm.

The upper bound obtained by solving LRP can be tightened by optimising the Lagrangian Dual Problem(LDP) as

$$\min_{\lambda \geq 0} \quad Z_{LR}(\lambda) \tag{14.15}$$

We use the *standard subgradient algorithm* (SSA) (Fisher 1981) which updates the Lagrangian multipliers at the ς-th iteration λ^{ς} according to

$$\lambda^{\varsigma+1} = \left[\lambda^{\varsigma} - \delta^{\varsigma} \frac{Z_{LR}(\lambda^{\varsigma}) - LB^{*}}{||g_{\lambda}^{\varsigma}||^2} g_{\lambda}^{\varsigma} \right]^{+} \tag{14.16}$$

where $[\cdot]^{+}$ denotes the non-negative part of a vector, δ^{ς} is a scalar step size, LB^{*} is the best known lower bound, and g_{λ}^{ς} is a subgradient calculated as

$$g_{\lambda}^{\varsigma}(k,t) = R_k - \sum_j r_{jk} \left(\sum_{s=t-p_j+1}^{t} x_{js}^{\varsigma} \right) \tag{14.17}$$

where x^{ς} is the optimal solution of LRP at the ς-th iteration.

In practice δ^{ς} is reduced by a factor ρ if Z_{LR} is not improved by at least Δ^{ς} after n_{iter}^{max} iterations. The algorithm can terminate when δ^{ς} is small enough to avoid excessive iterations.

The schedule S derived from the solution to $Z_{LR}(\lambda)$ is normally not feasible with respect to the resource constraints. Kimms (2001) proposed a Lagrangian Relaxation based Heuristic (LRH) which works very well on some randomly generated test cases. However our experiments with the set of test instances in Vanhoucke (2010) clearly shows that LRH has difficulty in finding feasible solutions on a significant percentage of instances. Careful analysis suggests that the test instances used in Kimms (2001) have much looser deadline and smaller duality gap compared with those of Vanhoucke (2010). Since the Lagrangian relaxation solution may not be close to the optimal solution for the hardest cases, it is not surprising that the simple forward list scheduling based LRH failed.

We present a Lagrangian Relaxation based Forward-Backward Improvement heuristic (LR-FBI). For LR-FBI we try to find a feasible schedule similar to S using FBI(S) detailed in Algorithm 11. Firstly a set of keys $K(S)$ is created for S. A key is a vector $x \in \mathbb{R}^{|V|}$ which is decoded into a schedule by a schedule-generation scheme (SGS) (Hartmann and Kolisch 2000). The iterative forward/backward scheduling technique is used to reduce the makespan of a deadline infeasible schedule. It was introduced by Li and Willis (1992) for RCPSP and was adapted for many meta-heuristics (Vanhoucke 2010). Finally, the *NPV* of the schedule is further improved by shifting activities (shift) as in Kimms (2001).

To calculate keys $K(S)$, rather than use a Linear Programming relaxation of the original problem (Gu 2008; Gu et al. 2007; Savelsbergh et al. 2005), we use the computationally more efficient α-point idea of Möhring et al. (2003) which is based on a single LRP solution. The jth key element of the μth key is defined as $x_j^{\mu} = S_j + \alpha_j^{\mu} \times p_j, \alpha_j^{\mu} \in [0,1]$. We have two different strategies to create $K(S)$. Best-$\alpha(\nu)$ generates ν uniformly distributed keys with $\alpha_j^{\mu} = \mu/\nu, \mu = 0, \ldots, \nu-1$. Random-$\alpha(\nu)$ generates ν random keys where each α_j^{μ} is randomly chosen with uniform distribution.

Algorithm 14.1: FBI(S)

```
$best\_NPV := -\infty$; generate keys $K(S)$;
for $x \in K(S)$ do
    $right := true$; $S' :=$ SGS_left$(x)$ % decode $x$ to schedule $S'$;
    while $makespan(S') > \overline{d}$ do
        if $right$ then
            $S'' :=$ SGS_right$(S' + p)$ % rightmost schedule using activity end times;
        else $S'' :=$ SGS_left$(S')$ % leftmost schedule using activity start times;
        if $makespan(S'') \geq makespan(S')$ then return $best\_NPV$;
        $right := \neg right$; $S' := S''$;
    $S' :=$ shift$(S')$; if $NPV(S') > best\_NPV$ then $best\_NPV := NPV(S')$ ;
return $best\_NPV$ ;
```

Algorithm 14.2: SGS_left(x)

```
current clock time $t := 0$; unscheduled activities $U := V$ ;
while $U \neq \emptyset$ do
    calculate $A := \{i \in U \mid$ activity i has no predecessor in $U\}$;
    while $A \neq \emptyset$ do
        $i := \text{argmin}_{j \in A} x_j$;
        find the earliest time $t'$ to schedule activity $i$ with respect to the resource and
        precedence constraints for the partial schedule of $V \setminus U$ ;
        if $t' \leq t$ then
            $U := U \setminus i$; $A := A \setminus i$; $s_i := t'$ ;
        else $A := A \setminus i$ ;
    $t := t + 1$;
return $S$;
```

SGS_left(x) (SGS_right(x)) (Debels and Vanhoucke 2007) greedily schedules activities one by one as early (late) as possible respecting the (reverse) precedence constraints and resource constraints, in the order where i is scheduled before (after) j if $x_i < x_j$ ($x_i \geq x_j$). The resulting schedules are left(right)-justified (Sprecher et al. 1995). SGS can be implemented in both serial and parallel modes (Hartmann and Kolisch 2000). A simple implementation of the parallel mode of SGS_left(x) is given in Algorithm 12.

14.5 Lagrangian Relaxation Method for Large Problems

To overcome the scalability problem of the max-flow algorithm in solving LRP we further relax some precedence constraints so that activities can form clusters that are independent from each other. We partition the set of activities V into $V = V_1 \cup V_2 \cup \ldots \cup V_U$ where U is the given number of clusters. The multi-cut of this partition is

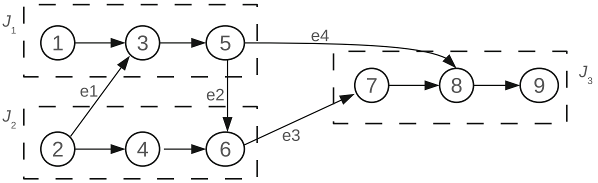


Fig. 14.1 Example of multi-cut for nine activities and three clusters

defined as the set of precedence relations $\hat{E} = \{(i, j) \in E | i \in V_g, j \in V_{g'}, g \neq g'\}$. Denote the set of precedence relations that hold on cluster V_g by $\hat{E}_g = \{(i, j) \in E | i \in V_g, j \in V_g\}$. Obviously we have $E = \hat{E} \cup \hat{E}_1 \cup \ldots \cup \hat{E}_U$. As an example given if Fig. 14.1, the set of nine activities is partitioned into $V_1 = \{1, 3, 5\}$, $V_2 = \{2, 4, 6\}$, and $V_3 = \{7, 8, 9\}$. The multi-cut of this partition is $\hat{E} = \{e1, e2, e3, e4\}$.

To reduce the number of Lagrangian multipliers introduced for the relaxed precedence relations, we use the weak form of the precedence constraints (Möhring et al. 2003)

$$\sum_t t(x_{jt} - x_{it}) \geq p_i \qquad ((i, j) \in \hat{E}) \tag{14.18}$$

By relaxing the precedence constraints (14.18) with Lagrangian multipliers μ we can obtain a Decomposable Lagrangian Relaxing Problem (DLRP)

$$Z_{LR}(\lambda, \mu) \;=\; \text{Max.}\, LRP(x) + \sum_{(i,j)\in\hat{E}} \mu_{ij} \left(\sum_t t(x_{jt} - x_{it}) - p_i \right) \tag{14.19}$$

$$\text{s. t.} \qquad (14.6), (14.9) \tag{14.20}$$

$$\sum_{\tau=t}^{\overline{d}} x_{i\tau} + \sum_{\tau=0}^{t+p_i-1} x_{j\tau} \leq 1 \quad ((i, j) \in E \backslash \hat{E};\ t = 0, \ldots, \overline{d}) \tag{14.21}$$

Let the set of precedence constraints in the multi-cut that have activity i as predecessor be $\hat{E}_i^+ = \{(i, j) \in \hat{E}\}$, the set of precedence constraints in the multi-cut that have activity i as successor be $\hat{E}_i^- = \{(j, i) \in \hat{E}\}$. For the example in Fig. 14.1, we have empty $\hat{E}_i^+$ and $\hat{E}_i^-$ except that $\hat{E}_2^+ = \{e1\}$, $\hat{E}_3^- = \{e1\}$, $\hat{E}_5^+ = \{e2, e4\}$, $\hat{E}_6^- = \{e2\}$, $\hat{E}_6^+ = \{e3\}$, $\hat{E}_7^- = \{e3\}$, and $\hat{E}_8^- = \{e4\}$.

By rearranging the items in the objective function of DLRP in Constraint (14.19), we get

$$LRP(x) + \sum_j \sum_t \left(\sum_{(i,j)\in\hat{E}_j^-} t\mu_{ij} - \sum_{(j,i)\in\hat{E}_j^+} t\mu_{ji} \right) x_{jt} - \sum_{(i,j)\in\hat{E}} \mu_{ij} p_i \tag{14.22}$$

By ignoring the constant terms the DLRP can be decomposed into U independent subproblems on each of the clusters V_g, $g = 1, \ldots, U$

$$\text{Max.} \sum_{j \in V_g} \sum_{t} \left(z_{jt} + \sum_{(i,j) \in \hat{E}_j^-} t\mu_{ij} - \sum_{(j,i) \in \hat{E}_j^+} t\mu_{ji} \right) x_{jt} \tag{14.23}$$

$$\text{s.t.} \quad \sum_{t} x_{jt} = 1 \quad (j \in V_g) \tag{14.24}$$

$$\sum_{\tau=t}^{\overline{d}} x_{i\tau} + \sum_{\tau=0}^{t+p_i-1} x_{j\tau} \leq 1 \quad ((i,j) \in \hat{E}_g;\ t = 0, \ldots, \overline{d}) \tag{14.25}$$

$$\text{all variables binary} \tag{14.26}$$

Since each subproblem has smaller size, the max-flow solver can solve DLRP much faster than LRP. Also these subproblems can be solved in parallel utilising the multi-core computers that are now ubiquitous. If main memory of the computer is a bottleneck we can construct the network flow model of each cluster on the fly. In this way we can solve the DLRP with over a hundred million variables within 500 MB memory.

The upper bound will become worse, i.e., $Z_{LR}(\lambda) \leq Z_{LR}(\lambda, \mu)$ since the weak form of the precedence constraints is used. Our goal therefore is to relax as few as possible the precedence constraints but still obtain activity clusters small enough to solve efficiently as a maximum flow problem. This can be formulated as the Min-Cut Clustering problem (MCC) as in Johnson et al. (1993)

$$\text{Min.} \sum_{g=1}^{U} \sum_{e \in E} z_{eg} \tag{14.27}$$

$$\text{s.t.} \quad \sum\nolimits_{g=1}^{U} x_{ig} = 1 \quad (i \in V) \tag{14.28}$$

$$x_{ig} - x_{jg} \leq z_{eg} \quad (e = (i,j) \in E;\ g = 1, \ldots, U) \tag{14.29}$$

$$l \leq \sum\nolimits_{i \in V} x_{ig} \leq u \quad (g = 1, \ldots, U) \tag{14.30}$$

$$\text{all variables binary} \tag{14.31}$$

where U is the upper bound of the number of clusters, x_{ig} is 1 if activity i is included in the cluster g, and otherwise 0. The set partitioning constraints (14.28) make sure that each activity is contained in only one cluster; Constraints (14.29) and the minimisation of (14.27) imply that the binary variable z_{eg} is 1 if the predecessor activity of e is included in the cluster g but the successor activity is in a different cluster, and otherwise 0; Constraints (14.30) ensure that the cluster size is within the specified range $[l, u]$.

MCC is also $\mathcal{NP}$-hard, and only small problems can be solved to optimality. For our purposes the cluster size constraints (14.30) are just soft constraints. We can use heuristics to generate good partitions very quickly. Our experimentation with METIS (Karypis 2011) shows that the project with 11,000 activities can be partitioned into 100 balanced parts within 0.1 s and only 384 precedence constraints need to be relaxed.

The subgradient algorithm tends to converge slowly for problems of high dimensions due to the zig-zag phenomenon (Zhao and Luh 2002). For large RCPSPDC problems we observed that the convergence of the precedence multipliers μ was extremely slow using the updating rule

$$(\lambda^{\varsigma+1}, \mu^{\varsigma+1}) = \left[(\lambda^{\varsigma}, \mu^{\varsigma}) - \delta^{\varsigma} \frac{Z_{LR}(\lambda^{\varsigma}, \mu^{\varsigma}) - LB^*}{||g_{\lambda}^{\varsigma}||^2 + ||g_{\mu}^{\varsigma}||^2} (g_{\lambda}^{\varsigma}, g_{\mu}^{\varsigma})\right]^+ \tag{14.32}$$

where $(g_{\lambda}^{\varsigma}, g_{\mu}^{\varsigma})$ is a subgradient calculated as

$$g_{\lambda}^{\varsigma}(k,t) = R_k - \sum_j r_{jk} \left(\sum_{\tau=t-p_j+1}^{t} x_{j\tau}^{\varsigma} \right) \tag{14.33}$$

and

$$g_{\mu}^{\varsigma}(i,j) = \sum_t t(x_{jt}^{\varsigma} - x_{it}^{\varsigma}) - p_i \quad ((i,j) \in \hat{E}) \tag{14.34}$$

The reasons could be

- The contribution of the precedence multipliers in the objective function value Z_{LR} is trivial. It can be even smaller than Δ^{ς} which is used to test if the upper bound is improved. Too small Δ^{ς} can only lead to excessive iterations before δ^{ς} can be reduced.
- In Constraints (14.32) the resource component of the subgradient $||g_{\lambda}^{\varsigma}||^2$ is much larger than the precedence component $||g_{\mu}^{\varsigma}||^2$, which may lead to steps too small for the convergence of μ.

Good μ can lead to near-feasible solutions with respect to the precedence constraints, which is important for the Lagrangian relaxation based heuristics to produce good lower bounds.

To accelerate the convergence of precedence multipliers we introduce a *hierarchical subgradient algorithm* (HSA) which has two levels. At the first level we update the multipliers according to (14.32) for a certain number of iterations n_{iter}^1 and then move to the second level by just updating the precedence multipliers as

$$\mu^{\varsigma+1} = \left[\mu^{\varsigma} - \delta^{\varsigma} \delta_{\mu}^{\varsigma} \frac{Z_{LR}(\lambda^{\varsigma}, \mu^{\varsigma})}{||g_{\mu}^{\varsigma}||^2} g_{\mu}^{\varsigma}\right]^+ \tag{14.35}$$

Only δ_μ^ς is reduced at the second level if Z_{LR} is not improved after n_{iter}^{max} iterations. After a certain number of iterations n_{iter}^2 the algorithm will switch back to the first level. This process is repeated until some stopping criterion are met.

14.6 Computational Results

We report the results of CP, Lagrangian relaxation and scatter search on small benchmark problems first in Sect. 14.6.1. The capability of Lagrangian relaxation for very large problems is presented in Sect. 14.6.2. We implemented the algorithms in C++. BOOST version 1.49.0 (Siek et al. 2001) is used for the max-flow solver and multi-threading. METIS version 5.0 (Karypis 2011) is used to solve the MCC problem. We implemented our CP based approach using the LCG solver Chuffed. For the subgradient algorithm we use $\delta^0 = 1$ in Constraints(14.16, 14.32), $\delta_\mu^0 = 0.01$ in Constraints (14.35), the threshold for significant objective value improvement is $\Delta^\varsigma = 0.0001 \times Z_{LR}(\lambda^\varsigma, \mu^\varsigma)$, the number of iterations for reducing δ^ς is $n_{iter}^{max} = 3$ and the factor $\rho = 0.5$, the maximal number of iterations for each level of HSA is $n_{iter}^1 = n_{iter}^2 = 10$. All tests were run on a computing cluster of which each node has two 2.8 GHz AMD 6-Core CPU.

14.6.1 Comparison of CP, LR, and Scatter Search on Benchmark Problems

We carried out extensive experiments on the benchmark set of Vanhoucke (2010). The benchmark set consists of 17280 RCPSPDC instances which are split in four problem sizes, i.e., 25-, 50-, 75-, and 100-activities. A more detailed specification of these instances can be found in Vanhoucke (2010). We set a time limit of 5 min for both CP and LR. The *NPV* and CPU time for each instance is also available for the scatter search method in Vanhoucke (2010) which terminates when a maximal number of schedules are generated using a computer with a Dual Core processor 2.8 GHz.

We illustrate the effects of our improvements on the LRH in Kimms (2001) in Table 14.1. We report the percentage of instances which have feasible schedule found (p_{feas}), the number of instances on which the best *NPV* is achieved (n_{best}) and the average relative deviation (Δ_{UB}) of the NPV found by the heuristic from upper bound UB (only instances for which all methods find a feasible schedule are considered). Δ_{UB} is defined as $abs((UB - LB)/UB)$ (see Vanhoucke 2010). Δ_{Vh} is calculated with the upper bound in Vanhoucke (2010), while Δ_{LR} uses the LR upper bound. The prefix S-(P-) stands for the serial (parallel) SGS. It can be seen that LRH has serious problems with feasibility. Parallel SGS is superior to serial SGS in terms of feasibility. The use of α-point further improves both feasibility and *NPV*. Since LR can produce much stronger upper bounds than Vanhoucke (2010) we only use Δ_{LR} for the remaining tests.

Table 14.1 Comparison of feasibility results on 100-activities instances

	p_{feas}	Δ_{Vh}	Δ_{LR}	n_{best}
LRH	72.9	203.3	76.1	836
S-Best-α(1)	77.3	203.0	74.7	1,005
P-Best-α(1)	91.9	207.6	79.5	450
P-Best-α(10)	95.8	197.1	69.4	1,919
P-Random-α(2000)	**99.5**	**197.1**	**69.4**	**2,304**

Values in bold are the best in their groups.

Table 14.2 Comparison of Scatter search, CP, and LR

	Scatter(5000)			CP-VSIDS			P-Random-α(2000)		
Size	p_{feas}	Δ_{LR}	n_{best}	p_{feas}	Δ_{LR}	n_{best}	p_{feas}	Δ_{LR}	n_{best}
25	**100.0**	73.1	2,507	**100**	**72.8**	**3,663**	99.8	81.6	795
50	**99.9**	91.6	1,556	98.4	104.8	1,124	99.9	**82.4**	937
75	99.7	106.9	1,196	90.3	—	–	**99.8**	**98.3**	**1,836**
100	**99.6**	100.2	**1,612**	76.8	—	–	99.5	**95.3**	1,524

Values in bold are the best in their groups.

Table 14.3 Comparison with best scatter search results on size 100

	p_{feas}	Δ_{LR}	n_{best}	$t_{cpu}^{\varnothing}$	t_{cpu}^{max}
Scatter(50000)	**99.6**	89.9	**2,003**	**26.2**	**139.8**
P-Random-α(2000)	99.5	**86.5**	1,283	167	607

Values in bold are the best in their groups.

We compare the reported results for scatter search (Vanhoucke 2010) with at most 5,000 schedules, with CP (Schutt et al. 2012), and LR-FBI in Table 14.2. Scatter search is very fast (average computation time is 4.2 s for size 100) and almost always finds a feasible solution. CP performs very well on the smallest instances but does not scale well. LR-FBI is highly competitive when problem sizes increase, generally finding better solutions, but requires more time than scatter search (82 s on average for size 100).

We also compare with the best results of scatter search with 50,000 schedules in Table 14.3, showing also average ($t_{cpu}^{\varnothing}$) and maximum solving time (t_{cpu}^{max}) in seconds. The time limit for LR-FBI is set to 10 min. The LR-FBI has better deviation. It was reported in Gu et al. (2013) that the hybridisation of LR and CP can produce better results on more instances.

14.6.2 Lagrangian Relaxation for Large Problems

Table 14.4 shows six of the eight test cases we obtained from a underground mining consulting company. The other two have no resource constraints and can be solved to optimality within 2 min. The first column in the table is the name of each case. The next two columns are the number of activities $|V|$ and the average number of successors of each activity in the precedence constraints E. The fourth column is the number of resources K. The fifth column is the number of Natural Clusters (NCl) which is the number of clusters without relaxing any precedence constraints. The remaining columns give the pairs of the number of obtained clusters (U) after relaxing the number of precedence constraints ($\hat{E}$) by solving MCC. These test cases ranges from about 1,400 activities to about 11,000 activities. The average number of successors per activity is small for all of these test cases. However only the smaller caNZ_def and caGL have natural clusters. Even for these two cases the natural clusters are not balanced in size. For example 38 precedence constraints have to be relaxed for caNZ_def to have 10 balanced clusters which is smaller than the number of natural clusters. It can be seen that more precedence constraints have to be relaxed when the number of clusters required increases. We omit here the running times of METIS because all MCC instances for our six test cases in Table 14.4 can be solved within 0.1 s.

14.6.2.1 Relaxing Resource Constraints Only

Without relaxing precedence constraints we can only solve the three smaller test cases and the results are shown in Table 14.5. The makespan of the feasible solution with the best *npv*, the upper bound (*UB*), and lower bound (*LB*) are reported in columns two to four. The fifth column is the optimality gap calculated as $\Delta_{UB} = (UB - LB)/UB$. The next two columns are the total number of iterations for the subgradient algorithm, and the total cpu time (t_{cpu}). The last two columns are the number of nodes $|V^{NM}|$ and number of edges $|E^{NM}|$ in the network model for solving the Lagrangian relaxation problem. All times are in seconds. Entries in bold are the best over entries in Tables 14.5, 14.6, 14.7, and 14.8. It can be seen that

Table 14.4 Test cases for large RCPSPDC

Case name	$\|V\|$	$\|E\|/\|V\|$	K	$\overline{d}$	NCl		$(U,\|\hat{E}\|)$	
caNZ_def	1,410	1.18	7	3,040	14	(10,38)	(50,126)	(100,233)
caW	2,817	1.26	2	3,472	1	(10,120)	(50,314)	(100,578)
caGL	3,838	1.16	5	2,280	17	(10,59)	(50,174)	(100,269)
caZ	5,032	1.36	5	8,171	1	(50,274)	(100,427)	
caCH	8,284	1.24	4	7,251	1	(100,623)	(200,866)	
caZF	11,769	1.16	5	6,484	1	(100,384)	(200,595)	

Table 14.5 Test results for relaxing resource constraints only

Case name	Makespan	*UB*	*LB*	Δ_{UB}	n_{iter}	t_{cpu}	$\|V^{NM}\|$	$\|E^{NM}\|$
caNZ_def	3,040	**1.199E9**	**1.140E9**	0.0496	81	10,370	2,874,917	5,854,335
caW	3,471	**6.014E8**	4.681E8	0.2217	72	12,336	6,178,671	13,449,822
caGL	2,269	**1.055E9**	**1.021E9**	0.0318	86	60,885	6,371,416	13,224,808

Values in bold are the best in their groups.

Table 14.6 Test results of SSA for comparison with HSA

Case name	U	Makespan	UB	LB	Δ_{UB}	n_{iter}	t_{cpu}
caNZ_def	10	3,035	1.202E9	1.047E9	0.128	96	2,261
caW	10	–	6.036E8	–	–	62	2,103
caGL	10	2,237	1.058E9	1.010E9	0.046	89	7,887
caZ	100	7,969	3.003E8	1.259E8	0.581	100	19,357
caCH	200	**7,251**	3.031E9	2.326E9	0.232	100	18,885
caZF	200	**6,337**	3.979E8	2.091E8	0.474	100	11,371

Values in bold are the best in their groups.

Table 14.7 Test results of HSA for comparison with SSA

Case name	U	Makespan	UB	LB	Δ_{UB}	n_{iter}	t_{cpu}
caNZ_def	10	3,033	**1.200E9**	1.136E9	0.054	100	6,330
caW	10	**3,456**	6.017E8	**4.803E8**	0.202	100	6,657
caGL	10	2,254	1.056E9	1.019E9	0.035	100	9,523
caZ	100	7,931	**2.919E8**	**1.735E8**	0.406	100	23,076
caCH	200	**7,251**	3.060E9	**2.449E9**	0.200	100	44,353
caZF	200	6,368	3.952E8	2.394E8	0.394	100	17,318

Values in bold are the best in their groups.

Table 14.8 Test results for effects of the number of clusters

Case name	U	Makespan	UB	LB	Δ_{UB}	n_{iter}	t_{cpu}
caNZ_def	10	3,033	**1.200E9**	1.136E9	0.054	100	6,330
caNZ_def	50	3,030	1.202E9	1.125E9	0.064	100	1,349
caNZ_def	100	**3,025**	1.203E9	1.100E9	0.086	100	967
caW	10	**3,456**	6.017E8	**4.803E8**	0.202	100	6,657
caW	50	3,457	6.025E8	4.615E8	0.234	100	4,390
caW	100	3,460	6.036E8	4.380E8	0.274	100	3,699
caGL	10	2,254	1.056E9	1.019E9	0.035	100	9,523
caGL	50	2,228	1.060E9	1.017E9	0.040	100	3,574
caGL	100	**2,218**	1.060E9	1.010E9	0.047	100	2,337
caZ	50	**7,909**	2.856E8	1.714E8	0.400	100	31,180
caZ	100	7,931	**2.919E8**	**1.735E8**s	0.406	100	23,076
caCH	100	**7,251**	**3.023E9**	2.365E9	0.218	100	64,758
caCH	200	**7,251**	3.060E9	**2.449E9**	0.200	100	44,353
caZF	100	6,427	**3.860E8**	**2.398E8**	0.379	100	27,663
caZF	200	6,368	3.952E8	2.394E8	0.394	100	17,318

Values in bold are the best in their groups.

caNZ_def and caGL have very good optimality gaps which are under 5 %, while caW has quite a large gap. Although caW and caGL have similar sizes of network flow model, caGL is much slower to solve. The reason could be that caGL has larger edge capacities which can also affect the performance of max-flow algorithm.

14.6.2.2 Relaxing Both Resource and Precedence Constraints

We first study how convergence can be affected after relaxing precedence constraints by comparing the standard subgradient algorithm (SSA) and the hierarchical subgradient algorithm (HSA). The maximal number of iterations is set to be 100 for all tests. We use ten cores to speed up the tests.

The results for SSA and HSA are reported in Tables 14.6 and 14.7 respectively. For the three smaller test cases it can be seen clearly that the upper bounds are trivially worsened by relaxing precedence constraints. However the lower bounds become significantly worse if SSA is used. The test case caW could not even find a feasible solution. In sharp contrast HSA finds a better lower bound for caW than without precedence constraint relaxation. This shows that HSA makes the Lagrangian multipliers associated with precedence constraints converge much faster. For the three larger cases HSA produces much better lower bounds than SSA especially for caZ. However SSA got a better upper bound for caCH. The reason is that HSA only uses half the number of iterations on updating the Lagrangian multipliers associated with the resource constraints. All the following tests will use HSA because of the overwhelming advantages on lower bounds over SSA.

We study how the quality of bounds are affected by the number of clusters. The results are reported in Tables 14.7 and 14.8. For the three smaller test cases, both upper bounds and lower bounds consistently become worse with the number of clusters increased. The lower bounds are more adversely affected than the upper bounds. For the three larger test cases the worsening of upper bounds is more than 1 % after the number of clusters is doubled. However the lower bounds become significantly better on caZ and caCH. The reason could be that list scheduling is not robust. But it can also be that the HSA does not converge well on the Lagrangian multipliers related to the precedence constraints. By setting $n_{iter}^1 = n_{iter}^2 = 50$ and keeping the maximal number of iterations the same we get lower bound 1.86E8 for caZ with $U = 50$. This suggests more work need to be done to adaptively tune parameters of HSA.

We next study the impact on solution time from parallelization. We calculate the speedup factor ρ_{LD} due to solving DLRP instead of LRP as

$$\rho_{LD} = t_{cpu}^{\o,L} / t_{cpu}^{\o,D}$$

where $t_{cpu}^{\o,D}$ is the average solution time of DLRP while $t_{cpu}^{\o,L}$ is the average solution time of LRP. We also calculate the speedup factor ρ_{DM} due to solving DLRP using multi-cores as

$$\rho_{DM} = t_{cpu}^{\o,D} / t_{cpu}^{\o,M}$$

where $t_{cpu}^{\o,M}$ is the average solution time of DLRP with multi-cores. We implemented a thread pool using the BOOST thread library. If the number of clusters is larger than the number of cores available the longest solution time first rule (Dósa 2004) is used

Table 14.9 Test results for speedups due to decomposition and parallelization as an effect of the number of clusters

Case name	$U = 10$			$U = 50$			$U = 100$		
	ρ_{LD}	ρ_{DM}	$\rho_{LD} * \rho_{DM}$	ρ_{LD}	ρ_{DM}	$\rho_{LD} * \rho_{DM}$	ρ_{LD}	ρ_{DM}	$\rho_{LD} * \rho_{DM}$
caNZ_def	0.94	2.16	2.02	1.68	5.63	9.49	2.46	5.38	13.24
caW	0.92	2.81	2.57	0.63	6.18	3.90	0.76	6.11	4.63
caGL	1.50	4.97	7.43	3.32	5.97	19.81	5.11	5.93	30.29

to schedule the threads. Solution time is estimated based on the previous iterations. If the estimation of the solution time produces the same list of threads as using the real solution time, this rule has performance guarantee $4/3$. The results for smaller cases are reported in Table 14.9 using ten threads. We cannot calculate ρ_{LD} for the larger cases. ρ_{DM} is similar to those of the smaller cases.

14.7 Conclusions

RCPSPDC is a difficult $\mathscr{NP}$-hard problem. For small problems the LCG based CP approach clearly outperforms the state of the art meta-heuristic and Lagrangian relaxation method. For larger problems the Lagrangian relaxation method becomes highly competitive in terms of solution quality if the forward-backward improvement heuristic is employed. It also produces the tightest upper bound for the RCPSPDC. Lagrangian relaxation can be also applied for very large industrial problems with further decomposition on the precedence constraints. The ability for parallelisation is especially attractive due to the ever-increasing computing power of modern parallel computers. The hybridization of the different approaches, i.e., constraint programming, decomposition method, and meta-heuristic is a promising future direction.

Acknowledgements NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

Aggoun A, Beldiceanu N (1993) Extending CHIP in order to solve complex scheduling and placement problems. Math Comput Model 17(7):57–73

Baroum S, Patterson J (1996) The development of cash flow weight procedures for maximizing the net present value of a project. J Oper Manag 14(3):209–227

Brucker P, Drexl A, Möhring R, Neumann K, Pesch E (1999) Resource-constrained project scheduling: notation, classification, models, and methods. Eur J Oper Res 112(1):3–41

Chaudhuri S, Walker R, Mitchell J (1994) Analyzing and exploiting the structure of the constraints in the ILP approach to the scheduling problem. IEEE Trans VLSI Syst 2(4):456–471

Debels D, Vanhoucke M (2007) A decomposition-based genetic algorithm for the resource-constrained project-scheduling problem. Oper Res 55(3):457–469

Demeulemeester EL, Herroelen WS (1992) A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. Manag Sci 38(12):1803–1818

Demeulemeester EL, Herroelen WS (1997) New benchmark results for the resource-constrained project scheduling problem. Manag Sci 43(11):1485–1492

Doersch R, Patterson J (1977) Scheduling a project to maximize its present value: a zero-one programming approach. Manag Sci 23(8):882–889

Dósa G (2004) Graham's example is the only tight one for P||C_{max}. Ann Univ Sci Budapest Sec Math 47:207–210

Fisher M (1981) The lagrangian relaxation method for solving integer programming problems. Manag Sci 27(1):1–18

Gu HY (2008) Computation of approximate alpha-points for large scale single machine scheduling problem. Comput Oper Res 35(10):3262–3275

Gu H, Xi Y, Tao J (2007) Randomized Lagrangian heuristic based on Nash equilibrium for large scale single machine scheduling problem. In: Proceedings of the 22nd IEEE international symposium on intelligent control, pp 464–468

Gu H, Stuckey P, Wallace M (2012) Maximising the net present value of large resource-constrained projects. In: Milano M (ed) CP 2012. Lecture notes in computer science, vol 7514. Springer, Heidelberg, pp 767–781

Gu H, Schutt A, Stuckey P (2013) A Lagrangian relaxation based forward-backward improvement heuristic for maximising the net present value of resource-constrained projects. In: Gomes C, Sellmann M (eds) CPAIOR 2013. Lecture notes in computer science, vol 7874. Springer, Heidelberg, pp 340–346

Hartmann S, Briskorn D (2010) A survey of variants and extensions of the resource-constrained project scheduling problem. Eur J Oper Res 207(1):1–14

Hartmann S, Kolisch R (2000) Experimental evaluation of state-of-the-art heuristics for resource constrained project scheduling. Eur J Oper Res 127(2):394–407

Harvey W, Stuckey PJ (2003) Improving linear constraint propagation by changing constraint representation. Constraints 8(2):173–207

Icmeli O, Erengüç SS (1996) A branch and bound procedure for the resource constrained project scheduling problem with discounted cash flows. Manag Sci 42(10):1395–1408

Johnson E, Mehrotra A, Nemhauser G (1993) Min-cut clustering. Math Program 62(1–3):133–151

Karypis G (2011) METIS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices, Version 5.0. URL http://glaros.dtc.umn.edu/gkhome/views/metis

Kimms A (2001) Maximizing the net present value of a project under resource constraints using a Lagrangian relaxation based heuristic with tight upper bounds. Ann Oper Res 102(1–4):221–236

Kolisch R, Hartmann S (2006) Experimental investigation of heuristics for resource-constrained project scheduling: an update. Eur J Oper Res 174(1):23–37

Li K, Willis R (1992) An iterative scheduling technique for resource-constrained project scheduling. Eur J Oper Res 56(3):370–379

Möhring RH, Schulz AS, Stork F, Uetz M (2003) Solving project scheduling problems by minimum cut computations. Manag Sci 49(3):330–350

Neumann K, Zimmermann J (2002) Exact and truncated branch-and-bound procedures for resource-constrained project scheduling with discounted cash flows and general temporal constraints. Cent Eur J Oper Res 10(4):357–380

Ohrimenko O, Stuckey PJ, Codish M (2009) Propagation via lazy clause generation. Constraints 14(3):357–391

Padman R, Smith-Daniels DE, Smith-Daniels VL (1997) Heuristic scheduling of resource-constrained projects with cash flows. Nav Res Log 44(4):365–381

Russell AH (1970) Cash flows in networks. Manag Sci 16(5):357–373

Savelsbergh M, Uma R, Wein J (2005) An experimental study of LP-based approximation algorithms for scheduling problems. INFORMS J Comput 17(1):123–136

Schulte C, Stuckey PJ (2008) Efficient constraint propagation engines. ACM Trans Program Lang Syst 31(1):1–43 (Article No. 2)

Schutt A, Feydy T, Stuckey PJ, Wallace MG (2011) Explaining the cumulative propagator. Constraints 16(3):250–282

Schutt A, Chu G, Stuckey PJ, Wallace MG (2012) Maximizing the net-present-value for resource constrained project scheduling. In: Beldiceanu N, Jussien N, Pinson E (eds) CPAIOR 2012. Lecture notes in computer science, vol 7298. Springer, Heidelberg, pp 362–378

Selle T, Zimmermann J (2003) A bidirectional heuristic for maximizing the net present value of large-scale projects subject to limited resources. Nav Res Log 50(2):130–148

Siek JG, Lee LQ, Lumsdaine A (2001) The boost graph library: user guide and reference manual. Addison-Wesley, Boston

Sprecher A, Kolisch R, Drexl A (1995) Semi-active, active, and non-delay schedules for the resource-constrained project scheduling problem. Eur J Oper Res 80(1):94–102

Vanhoucke M (2010) A scatter search heuristic for maximising the net present value of a resource constrained project with fixed activity cash flows. Int J Prod Res 48(7):1983–2001

Vanhoucke M, Demeulemeester EL, Herroelen WS (2001) On maximizing the net present value of a project under renewable resource constraints. Manag Sci 47(8):1113–1121

Zhao X, Luh PB (2002) New bundle methods for solving Lagrangian relaxation dual problems. J Optim Theory App 113(2):373–397

Zhu D, Padman R (1999) A metaheuristic scheduling procedure for resource-constrained projects with cash flows. Nav Res Log 46(8):912–927

Chapter 15
Exact Methods for the Resource Availability Cost Problem

Savio B. Rodrigues and Denise S. Yamashita

Abstract In this chapter, an exact method for the RACP problem is built from a combination of an RCPSP exact solver and RCPSP heuristic. In the RACP, the objective is to find the resource values that yield the least cost while finishing the project before the deadline. In the present method, the project feasibility is assessed by fixing the RACP resources and solving the underlying RCPSP with an exact algorithm. The idea is to reduce the number of RCPSP subproblems to be solved by sweeping the search space with a branching strategy that generates good bounds along the search. This approach is called the modified minimum bounding algorithm (MMBA). In the algorithm, we also employ a heuristic method in order to find an upper bound for the project cost. We fully describe the MMBA including possible alternative implementations. We also include an integer programming formulation of the RACP to be used directly or in subproblem solvers.

Keywords Exact algorithm • Heuristic bound • Hybrid method • Project scheduling • Resource availability

15.1 Introduction

In this chapter we address an exact method called the modified minimum bounding algorithm (MMBA) for solving the resource allocation cost problem (RACP); the problem classification is $PS|prec,\overline{d}|\sum c_k \max r_{kt}$. In the RACP, the objective is to complete the project before the deadline and to determine resource levels in order to minimize the total cost. The approach of the method is to repeatedly fix resource levels and solve the underlying project scheduling problem (RCPSP) where, of course, it is desirable to solve as few RCPSP as possible. The MMBA

S.B. Rodrigues (✉)
Department of Mathematics, Federal University of São Carlos, São Carlos, Brazil
e-mail: savio@dm.ufscar.br

D.S. Yamashita
Department of Production Engineering, Federal University of São Carlos, São Carlos, Brazil
e-mail: denisesy@dep.ufscar.br

C. Schwindt, J. Zimmermann (eds.), *Handbook on Project Management and Scheduling Vol. 1*, International Handbooks on Information Systems,
DOI 10.1007/978-3-319-05443-8_15

is a method to organize the search so that good bounds are obtained along the search. We remark that, as long as an efficient subproblem solver is available, the MMBA can be applied to other project scheduling problems found in this book. For example, the MMBA can be applied to solve problems where activities have different restrictions: multi-mode, preemption, sequence dependence, etc; nevertheless the formulation must have a project deadline and a resource availability cost. In principle, the MMBA can be adapted to solve problems with time varying resources; the dimension of the search space increases and this increases the computational effort.

The RACP is formulated as follows: the decision variables are the availabilities of resources R_k, $k = 1, \ldots, K$ and the starting times S_i of activities, $i = 1, \ldots, n$. The *objective* is to minimize the total cost

$$c \cdot R = \sum_{k=1}^{K} c_k R_k$$

where $R = (R_1, \ldots, R_K)$ denotes the vector of resource availabilities, which we also call *resource vector* for short, and $c = (c_1, \ldots, c_K)$ denotes the *cost vector* with $0 < c_1 \leq c_2 \leq \cdots \leq c_K$. As usual, each activity i has a duration p_i and requires r_{ik} quantities of the renewable resource k during its processing time. These activities are subjected to a set E of precedence relations; if $(i, j) \in E$, activity i must precede activity j. The sum of resource k required by all the activities being processed at a given time must not exceed the availability R_k, for $k = 1, \ldots, K$. All activities of the project must finish before or on the deadline $\overline{d}$. A complete integer programming definition is given in Sect. 15.7.

There are a few articles with exact solution methods for the RACP. It was first studied by Möhring (1984) motivated by a bridge construction project. Later, Demeulemeester (1995) proposed an exact algorithm (called minimum bounding algorithm—MBA) to solve the RACP and showed that the MBA was faster on the same bridge project instances proposed by Möhring (1984). Rangaswamy (1998) proposed a branch-and-bound for the RACP and applied it to the same instance set used by Demeulemeester (1995). Computational results were not conclusive due to the difference in the computer platforms used in the experiments. Drexl and Kimms (2001) proposed two lower-bound procedures for the RACP based on Lagrangian relaxation and column generation methods. Moreover, feasible solutions were provided by both procedures. The authors pointed out that the solution of the RACP for various deadlines provides time/cost tradeoffs, which is valuable information for negotiating the price for the project. Rodrigues and Yamashita (2010) brings an exact method with a branching procedure similar to the MBA but where heuristics are used to reduce the search space. Computational experiments show an improvement over the MBA algorithm.

An important concept is the feasibility of the resources assigned to the project. A resource vector $R = (R_1, \ldots, R_K)$ is said to be *feasible* if there is a schedule that finishes the project before or on the deadline. If such a schedule does not exist, we say R is an *unfeasible* resource vector. Once a resource vector R is given, an algorithm that establishes if R is feasible/unfeasible is called a *subproblem solver* (SPS) or simply a *solver*. Any algorithm that solves the RCPSP can be used as a subproblem solver; in particular, the algorithm may be stopped as soon as a feasible schedule is found. We remark that a heuristic method may speed up the proof that R is feasible however only exact methods and lower bounds on duration (cf. Chap. 3 of this handbook) can prove that R is unfeasible. We use heuristics to find an upper bound on the project cost, the methods found in Chaps. 4 and 16 of this handbook are also suitable for this purpose.

The goal is to sweep the search space requiring a small number of subproblem calls. The MMBA accomplishes this by generating good cuts along the search whenever a resource vector R^u is found to be unfeasible. Of course, any resource vector R with fewer resources than R^u is necessarily unfeasible too. We state that the inequality $R \leq R^u$ holds whenever $R_i \leq R_i^u$ for every index i.

The algorithm MMBA works with any choice of subproblem solver and constructive heuristic. In this chapter, as in Rodrigues and Yamashita (2010), we use the solver proposed by Demeulemeester and Herroelen (1992) and the constructive heuristic method from Tormos and Lova (2003) which employs dispatch rules. Other exact subproblem solvers, bounds on project completion time, and heuristics can be used instead; see for example Chaps. 1–4 of this handbook.

The present chapter is structured as follows. Section 15.2 explains the search strategies and other features of the MMBA. In Sect. 15.3 we give the pseudo-code of the MMBA and details of its implementation. An important subroutine of the MMBA called FCC is explained in Sect. 15.4; it determines the resource vector to be evaluated by the subproblem solver. Two theoretical results related to the MMBA are described in Sect. 15.5. In Sect. 15.6 we give a summary of computational results to show the size of instances that can be dealt with this algorithm, also we show that the cuts introduced by the MMBA yield a significant time reduction in many instances. In Sect. 15.7 we give an integer programming formulation to aid possible subproblem constructions and other direct methods for the RACP.

15.2 Strategies for Searching an Optimal Resource Vector

There are several strategies for searching an optimal resource vector. The details of the MMBA are given in Sect. 15.3, but first we explain its motivation. The MMBA is designed to yield good cuts while performing a branching scheme that searches for an optimal solution. Loosely speaking, the algorithm bisects the search space at a certain objective value, it makes one call of the subproblem solver, and it cuts from the search space either a set of points with higher objective value or a set of points with lower ones. The larger the set of points being cut the better.

We illustrate how the MMBA generates good cuts along the search using Fig. 15.1 where we consider an instance with two resources. The points with integer coordinates represent the resource vectors; the horizontal line BD and the vertical lines BC represent the minimal quantities of resources 1 and 2 respectively. The slant

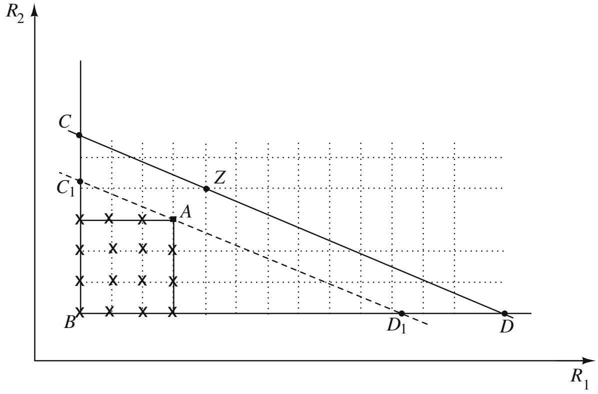


Fig. 15.1 The *vertical (horizontal) line BC* (*BD*) represents the least amount of resource R_2 (R_1). The segment *CD* indicates the points with equal objective function value as the point Z

line connecting *CD* represents points with the same objective cost as the incumbent solution Z; this slant line is called the *incumbent line*. Assume we do not have information about the feasibility of the resource vectors inside the triangle *BCD* (including its border). A resource vector must be selected inside the triangle *BCD* and evaluated by the subproblem solver. By selecting the point A, we can be sure to eliminate many points from the search space. If A is feasible then a new incumbent solution is found and the line C_1D_1 is the new incumbent line; all points above this line are eliminated from the search. Else, if A is unfeasible, all the points marked with X are eliminated because their resources are strictly smaller than A. Thus, the point A is the pivot of a win-win strategy: either feasible or unfeasible, many points are cut from the search space. Of course, this is only an illustration of the first step of the method. The procedure needs to be repeated by trying to eliminate as many points from the search space as possible in each subproblem evaluation. Also, we need a branching scheme which we explain next.

The MMBA employs a branching scheme to search an optimal solution by tracking the point of the search space with the least objective function value, this point is called the *best candidate point*. For example, considering Fig. 15.1 and assuming *BCD* to be the initial search space, the point B is the best candidate point because it has the smallest objective function value. Now assume point A is found to be unfeasible, then all the points marked with X in Fig. 15.1 are known to be unfeasible too. The point B is deleted as the best candidate point and the search proceeds by finding the next best candidate point. Figure 15.2 shows that the point B branches into points B_1 and B_2 where either B_1 or B_2 is the best candidate point. In Fig. 15.2 the dotted line through B_1 shows that it has a lower objective function value than B_2. Both points B_1 and B_2 are kept on a list ordered by least cost, this list is called list of candidate points (LCP) and it is comprised of $\{B_1, B_2\}$ in this example. The best candidate point is now B_1 and the MMBA seeks to prove it is unfeasible by evaluating the feasibility of point A_1 with larger resources than B_1. A detailed description of the algorithm is given in the next two sections.

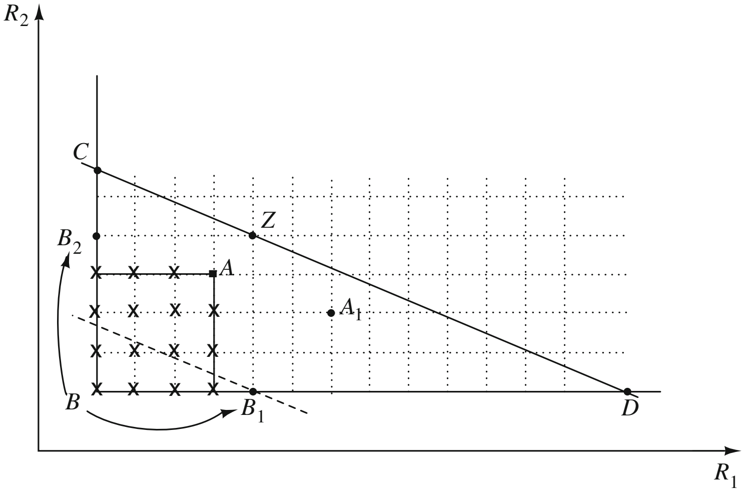


Fig. 15.2 When A is known to be unfeasible, B_1 and B_2 branch from the point B. If A_1 is unfeasible then B_1 is also unfeasible

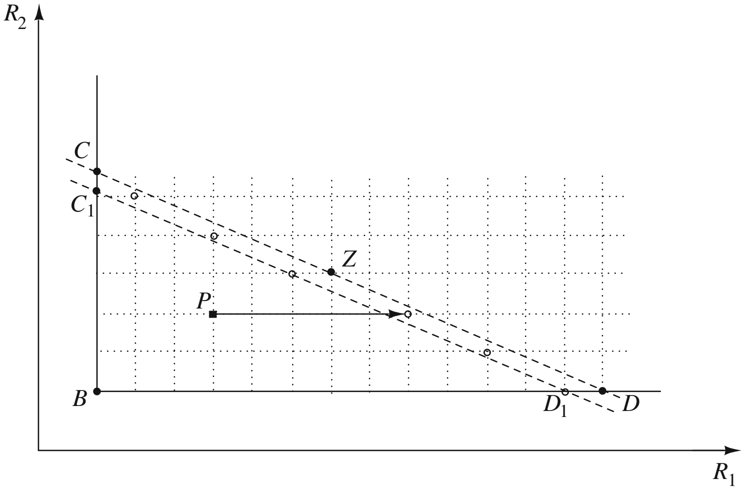


Fig. 15.3 The incumbent slice is the region between the *dashed lines*, any point in the triangle BC_1D_1 is dominated by at least one point in the incumbent slice as illustrated by the point P

The incumbent slice is an important concept introduced in Rodrigues and Yamashita (2010). Given an incumbent resource vector $R^z = (R_1^z, \ldots, R_K^z)$ (represented by the point Z in Fig. 15.3), the *incumbent slice* comprises all resource vectors R of the search space such that

$$c \cdot R^z - c_1 \leq c \cdot R < c \cdot R^z$$

The incumbent slice is illustrated in Fig. 15.3; it comprises the region between the two parallel dashed lines. We remark that any resource vector R^p in the triangle BC_1D_1 can be dominated by a resource vector R^o that is located in the incumbent slice, i.e., there is a resource vector R^o with $R^p \leq R^o$. To find a point marked o from point P, simply increase the first resource of P until it reaches the incumbent slice as indicated by the horizontal arrow in Fig. 15.3. As a consequence, *if all the points on the incumbent slice are unfeasible then the incumbent solution Z is optimal*; this result is stated in Theorem 1 of Rodrigues and Yamashita (2010). A top-down search strategy consists in evaluating points on the incumbent slice before any other points. Although valid, this strategy is fragile because if there is a feasible point on the incumbent slice then the incumbent slice is redefined with a slightly lower incumbent cost; repeatedly finding new incumbent solutions may hinder efficiency.

We remark that the choice of the pivot point A in the example of Fig. 15.1 is rather arbitrary, other choices are possible. In fact, the point A can be moved either closer to the best candidate point B or close to the incumbent line. These choices of A are controlled in the MMBA by a parameter we call θ, $0 \leq \theta \leq 1$. The MMBA selects $A = B$ when $\theta = 0$ and it selects a point in the incumbent slice when $\theta = 1$. Of course, these two extreme values give antagonistic strategies and either choices $\theta = 0$ or $\theta = 1$ may perform extremely poorly in some instances. To see this, consider again Fig. 15.3. If $\theta = 0$ and if the incumbent solution Z happens to be an optimal solution, then the MMBA evaluates every point in the triangle BCD before concluding that the incumbent point Z is optimal. Of course, a slight increase in θ avoids this undesirable outcome. We have performed computational tests in Rodrigues and Yamashita (2010) and we have found that $\theta = 3/4$ is a good choice. In Sect. 15.4 we suggest how to select θ dynamically so that the method avoids the pitfalls of the worst case scenario.

By guessing if an optimal solution is closer to the candidate point or closer to the incumbent slice one may improve the MMBA. A correct guess may significantly reduce the number of subproblem calls and the total computational time but a wrong guess may hinder any computational gain. As a general rule, we recommend to heuristically adapt parameters using additional information. For example, it is possible to gather information from recent outcomes of the subproblem calls and attribute a change in the parameter θ accordingly; if recent subproblem calls return feasible/unfeasible then θ should be decreased/increased. With the same goal, one can use heuristics to evaluate if an instance is easy or difficult before calling the subproblem solver. On devising new algorithms one should keep in mind that the computational time required by each call of the subproblem solver varies with the resource vector. In computational experiments the number of subproblem calls is strongly correlated to the total computational time of the MMBA, see Sect. 15.6.

We remark that the MMBA can be used with time varying resources. The fundamental idea is to compare the amount of each resource k at each time t, i.e., regard $R_k(t)$ as a resource matrix with dimensions $K \times \overline{d}$ and compare two matrices $R \leq R^z$ when the inequality $R_k(t) \leq R^z_k(t)$ holds for each entry. The incumbent slice is defined in a search space with dimension $K\overline{d}$. This dimension is larger than the dimension K of constant resources and this increases the computational effort. Nevertheless, there is one computational saving: once an incumbent resource matrix R^z is found, the time varying resources can be rearranged to find an optimal resource matrix R^o for the incumbent schedule; this is a new incumbent solution. An example of optimization of the resource matrix can be found in Nübel (2001) where it is applied to a resource renting problem.

15.3 The Modified Minimum Bounding Algorithm (MMBA)

The algorithm for solving the RACP is explained in four parts: data structure and its initialization, branching scheme, selection of cut candidate, and the pseudo-code. The heart of the MMBA is the selection of a cut candidate which reduces the need for new search branches.

The *data structure* consists of two lists of vectors: the *list of candidate points* (LCP) which stores the data for the branching scheme that searches an optimal solution; and the *list of known cuts* (LKC) which keeps the points that are known to be unfeasible. Initially LKC is empty and assume the LCP is initially comprised of a single vector R^{min} where all the resources are at their *theoretical* minimum

$$R_k^{min} = \max_{1 \leq i \leq n} r_{ik} \tag{15.1}$$

$k = 1, \ldots, K$. We say that the *optimal candidate property* (OCP) is satisfied if *feasibility of the first element of LCP implies it is also an optimal solution*. Clearly the initialization of LCP with Eq. (15.1) satisfies the OCP. There is another initialization of LCP that leads to a better overall computational time; this is explained at the end of this section.

The *branching scheme* of the MMBA is organized so that LCP satisfies the OCP along the algorithm. To explain how the branching scheme works together with the LCP we describe the pseudo-code of the *minimum bounding algorithm* (MBA); we remark that the MBA originates from Demeulemeester (1995). Let the variable R^f store the first element of LCP at any time during the execution of the MBA and let the subroutine SPS(R^f) return true/false if the resource vector R^f is feasible/unfeasible. Assume that the LCP is initialized so that the OCP holds true. The *while* statement of the MBA ends when R^f is feasible and, therefore, R^f is an optimal solution. If R^f is unfeasible then K new branches are created, each new branch increases the availability of a distinct resource of R^f by one unit; this is done on lines 2 through 5 of the pseudo-code. The new branches are inserted into the LCP; namely, K resource vectors

$$R^k = (R_1^f, R_2^f, \ldots, R_{k-1}^f, R_k^f + 1, R_{k+1}^f, \ldots, R_K^f)$$

for $k = 1, \dots, K$, are inserted in LCP in increasing order of cost function. By following this method, the fist element of LCP always maintains the OCP.

Algorithm—MBA

```
Initialize the LCP with the optimal candidate property (OCP)
Attribute to $R^f$ the first element of LCP
*while* SPS($R^f$) = false
    *for* $k := 1, \dots, K$
        $R^k := (R_1^f, \dots, R_{k-1}^f, R_k^f + 1, R_{k+1}^f, \dots, R_K^f)$
        Insert the vector $R^k$ in LCP (ordered by increasing cost)
        unless there is an $R$ in LCP such that $R \le R^k$
    *end*
    Remove the first element of LCP
    Attribute to $R^f$ the first element of LCP
*end*
```

The above algorithm exemplifies the bottom-up approach where an optimal solution can be found quickly if it is close to the minimal resource vector of Eq. (15.1) initializing LCP.

Next we explain the selection of the cut candidate R^c, this is done in a subroutine called *find cut candidate* (FCC). The MMBA modifies the while loop of the MBA by calling the subroutine SPS(R^c) with a resource vector R^c such that $R^c \ge R^f$, the resource vector R^c is called a *cut candidate*. If R^c is found to be unfeasible so is R^f; the idea is to select $R^c > R^f$ and thus encompass many elements of the search space simultaneously. Moreover, R^c must have a lower cost than the incumbent solution, $c \cdot R^c < c \cdot R^z$. The choice of R^c is heuristic and the intuition on how to choose this element is explained in Fig. 15.1. In Rodrigues and Yamashita (2010) we have made a selection based on the volume of the space that is being cut; this is explained in more detail in Sect. 15.4. The following pseudo-code describes the MMBA.

Algorithm—MMBA

```
Initialize the LCP with the optimal candidate property
Attribute to $R^f$ the first element of LCP
Initialize $R^z$ as a feasible resource vector
Initialize LKC as empty
If $c \cdot R^f = c \cdot R^z$ then return $R^z$ (*optimal solution*)
*repeat*
    is_feasible := TRUE
    *while* is_feasible
        $R^c$ := FCC
        is_feasible := SPS($R^c$)
        *if* is_feasible
            $R^z := R^c$ (*new incumbent is found*)
```

```
13                If $c \cdot R^f = c \cdot R^z$ then return $R^z$ (*optimal solution*)
14          *end*
15    *end*
16    Insert $R^c$ in LKC ($R^c$ *is unfeasible*)
17    branch := TRUE
18    *while* branch
19          *for* $k := 1, \ldots, K$
20                $R^k := (R_1^f, \ldots, R_{k-1}^f, R_k^c + 1, R_{k+1}^f, \ldots, R_K^f)$
21                If $c \cdot R^k < c \cdot R^z$ and there is no $R$ in LCP such
                  that $R \leq R^k$, then insert the vector $R^k$ in
                  LCP ordered by increasing cost
22          *end*
23          Remove the first element of LCP
24          If LCP is empty then return $R^z$ (*optimal solution*)
25          Attribute to $R^f$ the first element of LCP
26          If $c \cdot R^f \geq c \cdot R^z$ then return $R^z$ (*optimal solution*)
27          *if* $R^f \leq R^l$ for some $R^l$ in LKC
28                $R^c := R^l$ ($R^c$ *is a known cut for* $R^f$)
29          *else*
30                branch := FALSE
31          *end*
32    *end*
33 *end*
```

The MMBA accounts for the information of R^c being either feasible or unfeasible. If R^c is unfeasible then the new search branches are the resource vectors R^k on line 20; they have one unit of resource greater than R^c. If R^c is feasible then the incumbent solution is updated on line 12. The current cost of the incumbent solution is an upper bound on the objective value of new branches, the bound is imposed on line 21, and it also influences the selection of R^c by the FCC. The LKC is used to find out if R^f, the first element of LCP, is unfeasible by searching through LKC for an element R^u such that $R^u \geq R^f$. Of course, the search for R^u is much faster than one call of SPS.

We remark that by always selecting $R^c := R^f$ on line 9, the MMBA becomes equivalent to the MBA. Because of this connection, it is possible to theoretically compare these two algorithms and obtain a theorem which, in essence, states that if the MMBA makes more SPS calls than the MBA for a given instance then the difference of SPS calls cannot be too large. The statement and proof of the theorem are found in Rodrigues and Yamashita (2010), we explain the ideas of the proof in Sect. 15.5.

Finally, we describe efficient methods to initialize LCP. Initializing it with the resource vector of Eq. (15.1) is inefficient; there are two improvements that can be implemented in sequence. The first improvement is to find computationally the *least global value for each resource* k , i.e., find the least value of $R_k^{MIN} \geq R_k^{min}$ such that the resource vector $(+\infty, \dots, +\infty, R_k^{MIN}, +\infty, \dots, +\infty)$ is feasible for each $k = 1, \dots, K$. Then, the LCP can be initialized with the resource vector $R^{MIN} = (R_1^{MIN}, \dots, R_K^{MIN})$. The algorithm for finding R_k^{MIN} is simple: take an initial guess R_k^g and call SPS with the resource k limited by R_k^g and with all other resources being unlimited; if SPS returns a feasible/unfeasible schedule then decrease/increase R_k^g. The second improvement is to compute tradeoff curves among pairs of resources, for example, resource 1 and k, while all the other resources are unlimited. The *tradeoff curve* is comprised of all pairs of values $(R_1; R_k)$ such that

$$R^{1k} = (R_1, +\infty, \dots, +\infty, R_k, +\infty, \dots, +\infty)$$

is feasible but neither R_1 nor R_k can be reduced and preserve feasibility. An algorithm to obtain the tradeoff curve is in Demeulemeester (1995) where, in essence, to construct the curve, the resource R_k is lowered by one unit while R_1 is increased by one unit until a feasible schedule is found. For a precise notation, let us indicate $R_k(R_1)$ to say R_k is a function of R_1 on the tradeoff curve, i.e., if resource 1 is fixed at level R_1 and all resources except the k-th resource are unbounded, then the least value of R_k that yields a feasible schedule is $R_k(R_1)$. Once the resource pairs $(R_1; R_k(R_1))$ are found for $k = 2, \dots, K$, the LCP is initialized with the resource vectors at the lowest possible values. The following expression summarizes the set of resource vectors in the initial LCP:

$$\bigcup_{R_1^{MIN} \le R_1 \le R_1^{MAX}} (R_1, R_2(R_1), \dots, R_K(R_1))$$

where R_1^{MAX} is so large that $R_k^{MIN} = R_k(R_1^{MAX})$ for $k = 2, \dots, K$. The elements in LCP need to be ordered by increasing cost, then its first element satisfies the OCP. In essence, this procedure repeatedly solves instances of the RACP where each instance has two resources only. Computational experiments in Demeulemeester (1995) and Rodrigues and Yamashita (2010) use this initialization (we have experimented with simpler initializations of LCP but the computational time was clearly worse, thus, these results were not reported in the article).

As an example of the initialization of LCP, we consider an instance with three resources with costs $c_1 = 2, c_2 = 3$, and $c_3 = 5$. First, a simple algorithm can be used to determine three feasible resource vectors $(R_1^{MIN}, +\infty, +\infty)$, $(+\infty, R_2^{MIN}, +\infty)$, $(+\infty, +\infty, R_3^{MIN})$ where each resource is at its minimum possible value. In this example we assume the numerical values $R_1^{MIN} = R_2^{MIN} = R_3^{MIN} = 3$. A simple but inefficient initialization of LCP is the set $\{(3, 3, 3)\}$. An efficient initialization requires the determination of the tradeoff curves between R_1 and each other resource. In Fig. 15.4 we exemplify the construction of the tradeoff curve between R_1 and R_2 while keeping resource $R_3 = +\infty$. The arrows show the sequence of points that are evaluated for feasibility using the SPS; the feasible points along the path are marked with circles while the unfeasible points are marked with squares. We start the algorithm with the initial point $(R_1^{MIN}, R_2^{MIN}) = (3, 3)$ and

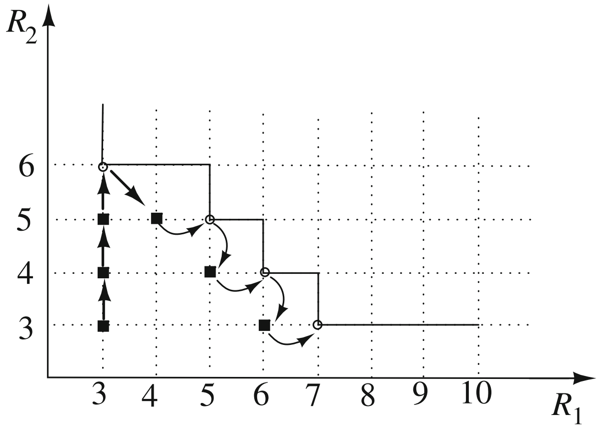


Fig. 15.4 Tradeoff curve of R_1 and R_2

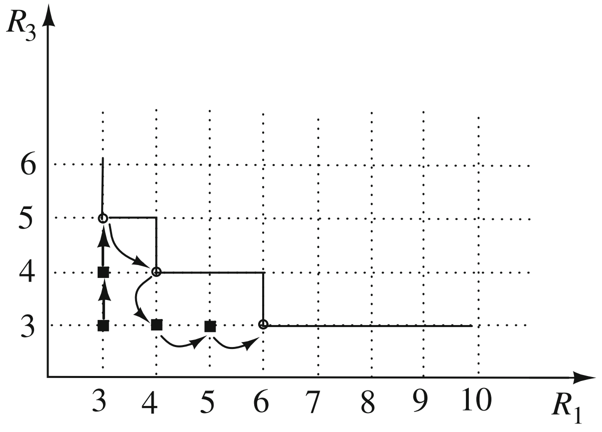


Fig. 15.5 Tradeoff curve of R_1 and R_3

then we increase resource R_2 while the point (3, R_2) is unfeasible. This generates the vertical sequence of points up to $(3,6)$. Now decrease R_2 by one unit to $(3,5)$ and proceed with the algorithmic loop: increase R_1 by one unit until a feasible point is found; then decrease R_2 by one unit until an unfeasible point is found or until it reaches R_2^{MIN}. The tradeoff curve connecting the feasible points is represented by the continuous line in Fig. 15.4. An example of tradeoff curve between R_1 and R_3 is shown in Fig. 15.5; the path of points evaluated by the SPS is also shown.

The LCP is initialized by varying R_1 from 3 to 7 while the tradeoff curves give the values for the other two resources; this yields the following resource vectors: $(3,6,5)$; $(4,6,4)$; $(5,5,4)$; $(6,4,3)$; $(7,3,3)$. Finally, these resource vectors are ordered by increasing cost, $\{(7,3,3); (6,4,3); (5,5,4); (4,6,4); (3,6,5)\}$.

There is a balance between investing too little or too much computational time to initialize LCP. We emphasize that the preceding example illustrates one possibility to initialize LCP, but this may be far from an optimal procedure. In the example we use R_1 as the pivot resource to construct the tradeoff curves. In principle one could compute tradeoff curves between other resource pairs to improve the initialization of LCP, e.g. R_2 and R_3. Nevertheless, our intuition is that it should be possible to design a clever initialization of LCP spending less computational time. For example, initializing the LPC with a shorter list, e.g. $\{(7,3,3);(6,4,3)\}$, may require less SPS calls and the overall efficiency of the MMBA may be improved. Unfortunately, alternative initializations of LCP have not been thoroughly examined.

15.4 Choice of Cut Candidate

The subroutine FCC (find cut candidate) heuristically selects a cut candidate R^c on line 9 of the MMBA pseudo-code; two conditions must be fulfilled $R^c \geq R^f$ and $c \cdot R^c$ is smaller than the incumbent cost. Here we describe the FCC used in Rodrigues and Yamashita (2010) and in the end of the section we describe other possibilities.

The cut candidate can be selected either closer to the R^f or closer to the incumbent slice; a parameter called θ, $0 \leq \theta \leq 1$, controls this selection. The parameter θ sets $R^c = R^f$ when $\theta = 0$ and when $0 < \theta \leq 1$ it sets the objective value $c \cdot R^c$ within the interval

$$\theta c \cdot R^z + (1-\theta)c \cdot R^f - c_1 \leq c \cdot R^c < \theta c \cdot R^z + (1-\theta)c \cdot R^f \tag{15.2}$$

When $\theta = 1$ the cut candidate R^c is selected in the incumbent slice. In Rodrigues and Yamashita (2010) we perform all the computational results with $\theta = 3/4$. Once $\theta > 0$ is given, there is a slice of points where R^c can be placed; these points are marked with a circle on Fig. 15.3 where $\theta = 1$ (if $\theta < 1$ the slice is displaced towards the point B). Intuitively, one should choose R^c as the point in the slice that maximizes the number of integer points in the rectangle with lower left corner in B and upper right corner in the slice. This choice maximizes, in a greedy sense, the number of points being cut from the search space. The heuristic idea is to maximize the K-dimensional volume of the box where the points R^f and R^c are at the diagonal of the box and R^c is restricted to the $K-1$-dimensional plane

$$c \cdot R^c = \theta c \cdot R^z + (1-\theta)c \cdot R^f$$

of the slice in Eq. (15.2). Of course, we need to regard R^c as a vector of $\mathbb{R}^K$ for a moment. Using Lagrange multipliers, it is a simple matter to discover R^c should be chosen as

$$R^c = R^f + sv$$

where s is a real positive parameter and $v = (v_1, \ldots, v_K)$ is a vector with $v_k = 1/c_k$. The continuous volume is an indication to the number of integer points inside this box, not a certainty, and the direction v is a clue on how to maximize the number of integer points inside the K-dimensional box. In short, v is used as a biased direction to increase R^f. The FCC starts with $R^c = R^f$ then spins a biased roulette wheel that selects one of the resources of R^c to be increased by one unit; this procedure is repeated until R^c verifies the inequality (15.2). With this direction, the pseudo-code for FCC is given below.

Algorithm—FCC

```
01 Initially set $u := R^f$
02 *while* true
03     Let $k$, $1 \leq k \leq K$, be the maximum index $k$ such that
               $c \cdot (u_1, \ldots, u_{k-1}, u_k + 1, u_{k+1}, \ldots, u_K) < \theta c \cdot R^z + (1 - \theta) c \cdot R^f$
04     If no such index exists, return $u$ and break loop (*exit procedure*)
05     $s := \sum_{h=1}^{k} v_h$
06     Select one resource $l$, $1 \leq l \leq k$, with probability $v_l / s$
07     $u_l := u_l + 1$
08 *end*
```

The right hand side of the inequality on line 3 guarantees that the objective function $c \cdot R^c$ remains within the desired value controlled by Eq. (15.2). *By fixing* $\theta = 0$ the FCC subroutine always selects $R^c = R^f$ and thus the *MMBA becomes equal to the MBA* as remarked in the previous section. From now on we associate the MMBA with $\theta = 0$ as being the MBA even if the MBA is not mentioned.

The computational results of Rodrigues and Yamashita (2010) were performed with $\theta = 3/4$. A summary of these results is in Sect. 15.6. It is possible to show theoretically that fixing $\theta > 0$ is a good choice when compared to fixing $\theta = 0$, this is explained in the next section.

Finally, we remark that depending on the RACP instance one may decide to risk exchanging some robustness for the possibility of a faster solution. This can be done by dynamically varying the parameter θ at each call of FCC. One possibility is to choose an integer $\kappa > 0$ and update, for example, $\theta := (\theta - 1/\kappa)^+$ whenever R^c turns out to be feasible on line 10 of the MMBA. When R^f is indeed an optimal solution, this reduction in θ accelerates the MMBA; i.e., θ decreases towards zero so that the MMBA selects $R^c = R^f$ earlier than it would do if θ remained a fixed value. By the same reason, one can reassign θ a larger value $\theta := \min\{\theta + 1/\kappa, 1\}$ in order to increase θ towards 1 if R^c is unfeasible. Also, it may be a good idea to allow θ slightly greater than 1 because it allows R^c to encompass a larger K-dimensional box. Of course, if R^c turns out to be feasible then nothing is gained because it will not be a new incumbent solution (some logical changes have to be made on MMBA to account for this possibility).

15.5 Theoretical Results

In this section we present two theorems about the effectiveness of using either $\theta = 0$ or $\theta = 1$ in the MMBA. The first one states that if all the points in the incumbent slice are unfeasible then the incumbent solution is optimal. The second one states that the upper bound on the number of subproblem calls when fixing $\theta > 0$ is slightly higher than the number of subproblem calls when fixing $\theta = 0$. These two theorems have practical implications on how to select the parameter θ in the MMBA: the overall conclusion is that a robust method should avoid the extremes of the interval $\theta \in [0, 1]$. The details about these theorems are found in Rodrigues and Yamashita (2010), here we explain their fundamental reasons.

The key step of theorem 1 is that any point in the search space is dominated by a point in the incumbent slice as illustrated in Fig. 15.3. We remark that P could be any point in the search space because it is associated to a point in the incumbent slice obtained by increasing the first resource of R^p by a suitable integer σ so that the resource vector $R^s = (R_1^p + \sigma, R_2^p, \dots, R_K^p)$ belongs to the incumbent slice. Thus, if all points in the incumbent slice are unfeasible then the same is true for all points in the search space and the incumbent solution is optimal. By setting $\theta = 1$, the MMBA can be used to verify if the incumbent solution is indeed optimal. Thus, fixing $\theta = 1$ sets the MMBA on a top-down search which is efficient if the incumbent solution is near the optimal objective value.

For any given instance of the RACP, Theorem 2 in Rodrigues and Yamashita (2010) compares the number of SPS calls needed when $\theta = 0$ and $\theta > 0$. Let μ and ν denote the number of SPS calls of the MMBA when executing the algorithm with θ fixed at 0 and θ fixed in $(0, 1)$, respectively. In simple terms, the theorem states that the difference $\nu - \mu$ is bounded by

$$\nu - \mu \leq \left\lceil \log_{1/\theta} \frac{f(R^z) - f^*}{(1-\theta)c_1} \right\rceil \tag{15.3}$$

where $f(R^z) = c \cdot R^z$ is the objective cost of the incumbent solution obtained on line 3 (MMBA pseudo code) and f^* is the objective cost of an optimal solution (usually unknown a priori) and we assume $f(R^z) > f^*$.

This bound becomes tighter as θ approaches zero but when θ approaches 1 the bound is not tight because the right hand expression tends to infinity. With an intermediate value of θ, $\theta = 1/2$ for example, the logarithm base 2 guarantees that the bound cannot be too large even if $f(R^z)$ and f^* are far apart.

Here we argue that the preceding bound can be sightly improved to yield

$$\nu - \mu \leq \left\lceil \log_{1/\theta} \frac{f(R^z) - f^*}{c_1} \right\rceil \tag{15.4}$$

The argument is essentially the same as in Rodrigues and Yamashita (2010) but here we use a sharper inequality and we simplify the notation. The bound follows by observing that the MMBA with $\theta > 0$ gains an advantage over the case $\theta = 0$ whenever line 10 of algorithm finds an unfeasible R^c with $R^c > R^f$ because then more than one point is cut from the search space. This advantage for fixing $\theta > 0$ disappears only if R^c fails to be unfeasible. It follows that the worst case for fixing $\theta > 0$ happens when R^c is a new incumbent solution at every loop of the MMBA. This situation occurs only if an optimal solution is the first element of LCP after it is initialized on line 1 of the MMBA. Denote the sequence of incumbent solutions by $R_1^z, R_2^z, \ldots, R_X^z, R_{X+1}^z$, where it finishes with an optimal solution $R_{X+1}^z = R^* = R^f$. Inequality (15.4) is obtained by counting the number of loops (SPS evaluations) it takes until the X-th incumbent cost $c \cdot R_X^z$ is low enough to yield $R^c = R^f$ as the result of the FCC procedure; i.e., the value of $c \cdot R_X^z$ is such that on line 3 of the FCC algorithm

$$c \cdot R^f + c_1 \geq \theta c \cdot R_X^z + (1 - \theta) c \cdot R^f$$

This inequality simplifies to

$$c_1 \geq \theta (c \cdot R_X^z - f^*) \tag{15.5}$$

where $f^* = c \cdot R^f$ because we are assuming $R^* = R^f$. Also from line 3 of the FCC algorithm, we find a bound for the difference of the objective value of the incumbent solution to the optimal objective value

$$c \cdot R_\xi^z - f^* < \theta (c \cdot R_{\xi-1}^z - f^*)$$

which is valid for every $\xi = 2, \ldots, X$. By applying this inequality repeatedly through all X loops of the MMBA, we find

$$c \cdot R_X^z - f^* < \theta^{X-1} (c \cdot R_1^z - f^*) \tag{15.6}$$

Now assume X is large enough to satisfy

$$c_1 \geq \theta^X (c \cdot R_1^z - f^*) \tag{15.7}$$

in which case inequality (15.5) is certainly true due to inequality (15.6). The bound (15.4) follows by taking the logarithm of inequality (15.7)

$$\log \frac{c_1}{c \cdot R_1^z - f^*} \geq X \ \log \theta$$

and recalling that: $f(R^z) = c \cdot R_1^z$, $\log \theta < 0$, and $\log 1/\theta = -\log \theta$.

15.6 Computational Results

Computational results reported in the literature show that solving the RACP is challenging. Rodrigues and Yamashita (2010) evaluated the performance of the MMBA in 384 instances with 30 activities and 4 resource types. All computational experiments were performed on a PC Pentium 4 (2.80 GHz with 1.0 GB of RAM). Two important parameters used for generating the instances are the network complexity (*NC*) and resource factor (*RF*), Kolisch et al. (1995). *NC* reflects the average number of immediate successors of an activity. *RF* varies between 0 and 1, reflecting the density of the different resource types needed by an activity. For example, if $RF = 1$, each activity requires all types of resources, while $RF = 0$ indicates that activities do not require any type of resources. It is also necessary to determine a deadline for the project. Drexl and Kimms (2001) compute the deadline $\overline{d}$ for the project as a function of the project critical path. Specifically, $\overline{d} = DF \max_i EC_i$, where DF is the deadline factor and EC_i is the earliest completion time of activity i. The cost c_k of a resource type k was drawn from a uniform distribution $U[1, 10]$. For each combination of $DF = 1.0, 1.2, 1.4$, and 1.6; $RF = 0.25, 0.5, 0.75$, and 1.0; and $NC = 1.5, 1.8$, and 2.1; eight instances were generated, resulting in a total of $4 \times 4 \times 3 \times 8 = 384$ instances. All instances used $\theta = 3/4$. The stopping criteria for MMBA was set to 7,200 s for each instance. All procedures were coded in C++ language.

Figure 15.6 shows the time spent to solve 288 instances that were solved within the 7,200 s time limit. The horizontal axis shows the total number of SPS calls and the vertical axis shows the total computational time (in seconds). Both axes are in $\log_{10}$ scale because the computational effort to solve the RACP can vary significantly from instance to instance. The legend on the bottom right corner of the graph indicates the instances classified by *DF*. Some features are evident from the graph: (i) a large cloud of clustered points indicates a strong correlation between the number of subproblem calls and the total computational time; (ii) on the cluster cloud, the instances with larger *DF* demand more computational time per SPS call (this feature is more pronounced for instances that required less than 10^2 SPS calls); (iii) there are scattered points above the cluster cloud indicating some instances required much more computational time per SPS call (in this group the points with $DF = 1.0$ are absent). We remark that common features are found when examining either the computational time of instances that were solved or the number of instances not solved within the time limit: it is found that the difficulty increases when *RF* increases, and the difficulty increases when *NC* decreases; Rodrigues and Yamashita (2010) brings a table with this data.

For the hardest instances, we identified the difficulty in solving the RCPSP subproblem as the cause for the increase in computational time. Each SPS call takes a longer time because the number of possible schedules that need to be searched has increased. Of course, an instance becomes less restricted if *DF* is increased or *NC* is lowered; more possible schedules cause more branches to be examined inside the SPS. It is less clear why the difficulty should increase when *RF* increases;

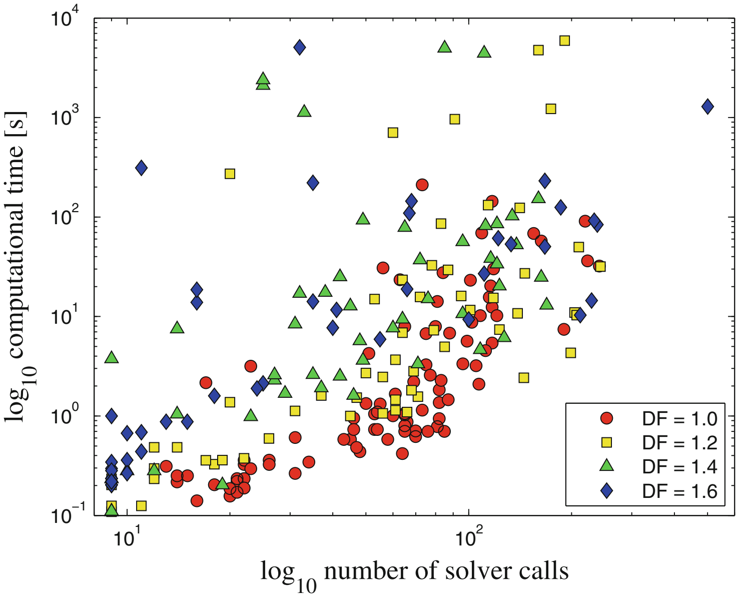


Fig. 15.6 Each *symbol* represents one of the instances solved within the time limit. The total number of SPS calls and the total computational time (in seconds) are in $\log_{10}$ scale

this occurs possibly because the activities become rather indistinct from each other allowing them to be interchanged. Not only is the difficulty of the problem affected by these three factors but also it is affected by the number of activities and the number of resources.

Figure 15.7 shows a comparison of the computational time to reach optimality in each instance using two different parameters $\theta = 3/4$ (MMBA) and $\theta = 0$ (MBA). The vertical axis is the *time gain* which is defined as the absolute difference between the computational time to find an optimal solution using each parameter, i.e., the time gain is $|t_{cpu}(\theta = \frac{3}{4}) - t_{cpu}(\theta = 0)|$ where $t_{cpu}(\theta = \frac{3}{4})$ and $t_{cpu}(\theta = 0)$ denote the total computational time for each parameter. The marks above the dashed line is the time gain when the MMBA was faster while the marks below the dashed line is the time gain when the MBA was faster. Within each group, the instances are sorted in order of decreasing time gain; they are numbered 1 through 118 and 1 through 46 for the MMBA and the MBA respectively. The legend indicates the relative gain which is defined as the time gain divided by the total computational time of the slower algorithm: $|t_{cpu}(\theta = \frac{3}{4}) - t_{cpu}(\theta = 0)|/\max\{t_{cpu}(\theta = \frac{3}{4}), t_{cpu}(\theta = 0)\}$.

We observe several features in Fig. 15.7 that show the advantage of using the MMBA with $\theta = 3/4$: it is faster in more instances (118 against 46 of the MBA); it consistently shows a larger time gain (up to 10^3 s); it frequently shows a relative gain above 40 % (circles) and sometimes above 80 % (squares). Also, it can be observed that the time gain of the MMBA correlates with its relative

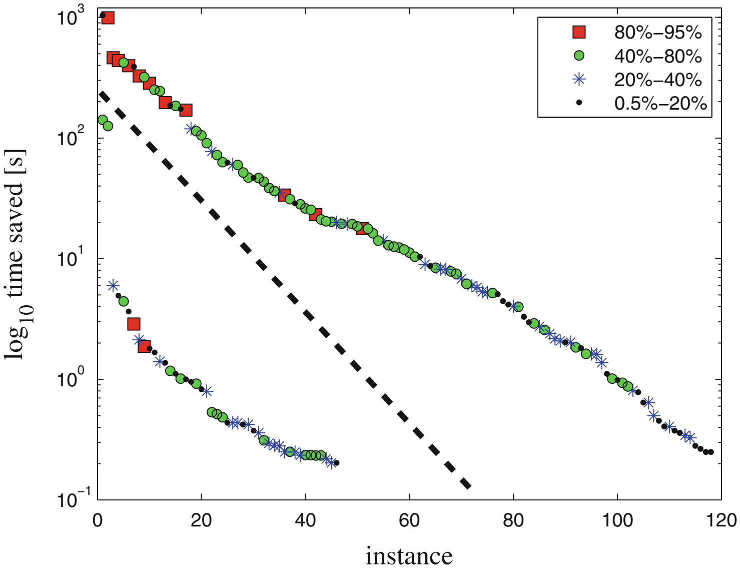


Fig. 15.7 Time gain when the best parameter is either $\theta = 3/4$ (above *dashed line*) or $\theta = 0$ (below it). The legend indicates the range of relative gain of each data point

gain: as one follows the figure from left to right, the marks above the dashed line show the predominance of squares, circles, stars, and dots, in this order; the marks appear in clusters. In contrast, the marks below the dashed line show that circles and dots are more or less evenly spread. The aforementioned correlation is a subtle feature; it may be interpreted as a sign that MMBA has slightly "lower complexity" then the MBA. For example, imagine the comparison between two algorithms where one algorithm is linear and the other is quadratic with respect to algorithmic complexity. In this case, both the time gain and the relative gain of the linear algorithm compared to the quadratic one increase as the size of the instances increases. Unfortunately, we cannot be certain that this is the real reason for the correlation observed in Fig. 15.7 because the effect is subtle and stochastic, also, there are several parameters involved when generating the instances.

We remark that the data in Fig. 15.7 consider 164 instances where the time gain was greater than 0.2 s; other factors unrelated to the MMBA algorithm may influence significantly the time gain below this threshold, e.g., implementation issues. In total, there are 285 instances where an optimal solution was found using both parameters (there are three instances where the MMBA finds an optimal solution while the MBA exceeds the maximum computational time).

As an overall conclusion, the MMBA does provide a robust and efficient framework to solve RACP instances. Moreover, the difficult RACP instances are hard because the underlying RCPSP is difficult to solve at each SPS call. To improve the SPS subroutine one may incorporate new developments described in Chaps. 1–4 of this handbook.

15.7 Integer Programming Model

To complete the description of exact methods for the RACP, we include an integer programming formulation of the problem. This formulation may be helpful either for solving the RACP directly or for creating the SPS subroutine for the MMBA. The model is as follows,

$$\text{Min.} \sum_{k=1}^{K} c_k R_k \tag{15.8}$$

$$\text{s.t.} \quad \sum_{t=ES_i}^{LS_i} x_{it} = 1 \quad (i = 0, \ldots, n+1) \tag{15.9}$$

$$\sum_{t=ES_i}^{LS_i} (t + p_i) x_{it} \leq \sum_{t=ES_j}^{LS_j} t x_{jt} \quad ((i,j) \in E) \tag{15.10}$$

$$\sum_{i=1}^{n} \sum_{b=(t-p_i)^+}^{t-1} r_{ik} x_{ib} \leq R_k \quad (k = 1, \ldots, K;\ t = 1, \ldots, \overline{d}) \tag{15.11}$$

$$x_{it} \in \{0,1\} \quad (i = 0, \ldots, n+1;\ t = ES_i, \ldots, LS_i) \tag{15.12}$$

$$R_k \geq 0 \tag{15.13}$$

$$R_k \in \mathbb{Z} \quad (k = 1, \ldots, K) \tag{15.14}$$

where the binary decision variable is x_{it} (1 if activity j starts at time t and 0 otherwise). Equation (15.9) enforces that each of the $n+2$ real and dummy activities starts exactly once. Equation (15.10) enforces the precedence relations of the network edges E, the processing time of activity j is p_j. Equation (15.11) enforces that activities never exceed the resource capacity at any time. The remaining equations represent integer constraints. We remark that the latest starting time of activity j, LS_j, and the earliest starting time of activity j, ES_j, are pre-computed from the precedence graph G and the deadline $\overline{d}$ of the project. Finally, an important observation, this model assumes an activity with processing time p_i which starts at S_i consumes resources at $t = S_i + 1, \ldots, S_i + p_i$, i.e., the activity does not consume resources at $t = S_i$.

15.8 Conclusions

The MMBA is an exact algorithm designed to solve the RACP using RCPSP subproblem solvers in its main loop. The algorithm introduces the parameter $\theta \in [0,1]$ which can accelerate the solution significantly; θ is a very robust parameter. The computational results show that using $\theta = 3/4$ is significantly better than using $\theta = 0$. The computational gain originates from a combination of constructive heuristics and good cuts which reduce the number of subproblem calls; the parameter θ controls how the cuts are achieved. A theorem shows that using $\theta > 0$ can require, in the worst case, a few more subproblem calls than using $\theta = 0$. For the hardest instances of the RACP we have found that the solution of the RCPSP subproblem is the main source of difficulty. The less restricted instances are computationally more demanding due to the large number of possibilities that need to be searched; this conclusion is supported by the computational experiments where we have detailed the parameters that influence the computational time. Finally, as long as an efficient subproblem solver is available, the MMBA can be used for other resource availability problems with different objective functions and different types of precedence relations.

References

Demeulemeester E (1995) Minimizing resource availability costs in time-limited project networks. Manag Sci 41:1590–1598

Demeulemeester E, Herroelen S (1992) A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. Manag Sci 38:1803–1818

Drexl A, Kimms A (2001) Optimization guided lower and upper bounds for the resource investment problem. J Oper Res Soc 52:340–351

Kolisch R, Sprecher A, Drexl A (1995) Characterization and generation of a general class of resource-constrained project scheduling problems. Manag Sci 41:1693–1703

Möhring RF (1984) Minimizing costs of resource requirements in project networks subject to a fixed completion time. Oper Res 32:89–120

Nübel H (2001) The resource renting problem subject to temporal constraints. OR Spektrum 23:359–381

Rangaswamy B. (1998) Multiple resource planning and allocation in resource-constrained project networks. Ph.D. dissertation, Graduate School of Business, University of Colorado, USA

Rodrigues SB, Yamashita DS (2010) An exact algorithm for minimizing resource availability costs in project scheduling. Eur J Oper Res 206:562–568

Tormos P, Lova A (2003) An efficient multi-pass heuristics for project scheduling with constrained resources. Int J Prod Res 41:1071–1086

Chapter 16
Heuristic Methods for the Resource Availability Cost Problem

Vincent Van Peteghem and Mario Vanhoucke

Abstract In this chapter, an Invasive Weed Optimization (IWO) algorithm for the resource availability cost problem is presented, in which the total cost of the (unlimited) renewable resources required to complete the project by a prespecified project deadline should be minimized. The IWO algorithm is a new search strategy, which makes use of mechanisms inspired by the natural behavior of weeds in colonizing and finding a suitable place for growth and reproduction. All algorithmic components are explained in detail and computational results for the RACP are presented. The procedure is also executed to solve the RACP with tardiness (RACPT), in which lateness of the project is permitted with a predefined penalty.

Keywords Heuristic algorithms • Invasive weed • Project scheduling • Resource availability • Tardiness cost

16.1 Introduction

The aim of project scheduling is the allocation of time intervals to the processing of activities, which can be executed by using a set of scarce resources. In the classical resource-constrained project scheduling problem (RCPSP), this set of available resources is limited and is not allowed to exceed the available resources. The main focus lies in the minimization of the total duration of the project subject to precedence relations between the activities and the limited renewable resource availabilities. Various exact and (meta-)heuristic procedures for the RCPSP are already proposed in the literature. For an overview of the literature on the RCPSP, see Kolisch and Hartmann (2006) and Hartmann and Briskorn (2010), amongst others.

V. Van Peteghem (✉)
EDHEC Business School, Lille, France
e-mail: vincent.vanpeteghem@edhec.edu

M. Vanhoucke
Ghent University, Ghent, Belgium
e-mail: mario.vanhoucke@ugent.be

C. Schwindt, J. Zimmermann (eds.), *Handbook on Project Management and Scheduling Vol. 1*, International Handbooks on Information Systems,
DOI 10.1007/978-3-319-05443-8_16

Another problem in project scheduling, the resource availability cost problem (RACP), focuses on the minimization of the resource cost. In contrast to the "problem of scarce resources" (Möhring 1984), the resources are not constrained by limited capacities, but a predefined deadline is imposed on the project duration. Möhring (1984) refers to this problem as the "problem of scarce time". The aim is to reduce the cost which is associated to the use of resources. The objective is to schedule the activities such that all precedence constraints are observed, the deadline for project termination is met, and the total resource availability cost is minimized. An extension of this problem is the resource availability cost problem with tardiness (RACPT). In this problem, a due date for the project is set, but tardiness of the project is permitted with a predefined penalty. The total project cost is than equal to the sum of the resource availability cost and the tardiness cost. If a large penalty cost is considered, the problem is equal to the RACP.

In this study, an Invasive Weed Optimization (IWO) algorithm is used to solve the RACP and the RACPT. IWO is a new search algorithm inspired by the principles of weed ecology. To the best of our knowledge, IWO has never been used before to solve a (project) scheduling problem in general and the RACP(T) specifically.

The remainder of this chapter is organized as follows. Section 16.2 describes the RACP(T) formulation, while in Sect. 16.3 an overview is given of the available metaheuristic solution procedures for the problem under study. In Sect. 16.4, the principles of the weed ecology are presented while the proposed solution algorithm and the various parameters are described in Sect. 16.5. Section 16.6 reports detailed comparative computational results and Sect. 16.7 contains the conclusions of this study.

16.2 Problem Formulation

The RACP can be stated as follows. A set of activities V of project network N, numbered from a dummy start node 0 to a dummy end node $n + 1$, is to be scheduled without pre-emption on a set $\mathscr{R}$ of renewable resource types. Each activity $i \in V$ has a deterministic duration p_i and requires r_{ik} units of resource type $k \in \mathscr{R}$ which has a constant availability R_k throughout the project horizon. We present a project network $N = (V, E, \delta)$ in an activity-on-node format where E is the set of pairs of activities between which a finish-start precedence relationship with time lag 0 exists, and a dummy start node 0 and end node $n + 1$ representing the start and completion of the project. These dummy nodes have zero duration while the other activities have a non-zero duration; the dummies also have zero resource usage. We assume the directed graph $G = (V, E)$ to be acyclic. A schedule S is defined by a vector of activity start times and is said to be feasible if all precedence and renewable resource constraints are satisfied. The objective of the RACP is to find a feasible schedule within a prespecified project deadline $\overline{d}$ such that the total resource cost is minimized. We define c_k as the unit cost of resource k resulting in a discrete non-decreasing total resource cost function $c_k R_k$ associated with the availability

R_k of the resource type k. The variables are the resource availability values R_k and the start times S_i of each project activity i. The resource availability cost problem can be represented as $m, 1|cpm, \delta_n|rac$ using the classification scheme of Herroelen et al. (1999) or as $PS\infty|prec, \overline{d}|\sum c_k \max r_{kt}$ following the classification scheme of Brucker et al. (1999).

The RACP can be conceptually formulated as follows:

$$\text{Min.} \sum_{k=1}^{K} c_k R_k \tag{16.1}$$

$$\text{s.t.} \quad S_i + p_i \leq S_j \quad ((i,j) \in E) \tag{16.2}$$

$$r_k(S,t) \leq R_k \quad (k \in \mathscr{R}, t = 0, \ldots, \overline{d} - 1) \tag{16.3}$$

$$S_{n+1} \leq \overline{d} \tag{16.4}$$

$$S_0 = 0 \tag{16.5}$$

The objective function (16.1) minimizes the total resource cost of the project. Constraints (16.2) take the finish-start precedence relations with a time lag of zero into account. The renewable resource constraints are satisfied thanks to Constraints (16.3), where $r_k(S,t)$ represents the amount of resource k used at time t given schedule S. Constraint (16.4) imposes a prespecified deadline to the project and Constraint (16.5) forces the project to start at time instance zero.

If the RACPT is considered, the tardiness cost is added to the objective function, which can then be formulated as follows:

$$\text{Min.} \left(\sum_{k=1}^{K} c_k \cdot R_k + w_{n+1}^T \cdot \max\{0, S_{n+1} - \overline{d}\}\right) \tag{16.6}$$

with w_{n+1}^T the predefined penalty cost. The constraints remain the same, only Constraint (16.4) does not need to be considered anymore.

Consider an example project with ten non-dummy activities, two resources ($K = 2$) and a project deadline $\overline{d} = 20$ (which is 25 % higher than its critical path length). The resource unit cost equals $c_1 = 2$ and $c_2 = 3$. Figure 16.1 shows the activity-on-node network where the number above the node denotes the activity duration and the numbers below the node the resource requirements for each resource type. In Fig. 16.2, optimal resource profiles of the two resource types are displayed, resulting in a total resource cost of $7 \cdot 2 + 8 \cdot 3 = 38$ and a project makespan of 20 time units.

If tardiness is allowed and the penalty cost is set $w_{n+1}^T = 2$, the optimal solution changes. By finishing the project 2 days later than the predefined due date, the resource use for resource 2 decreases to 6. The total cost thus decreases to 36, 32 for the resource cost ($7 \cdot 2 + 6 \cdot 3$) and 4 for the tardiness cost. In Fig. 16.3, optimal resource profiles for the two resource types are displayed.

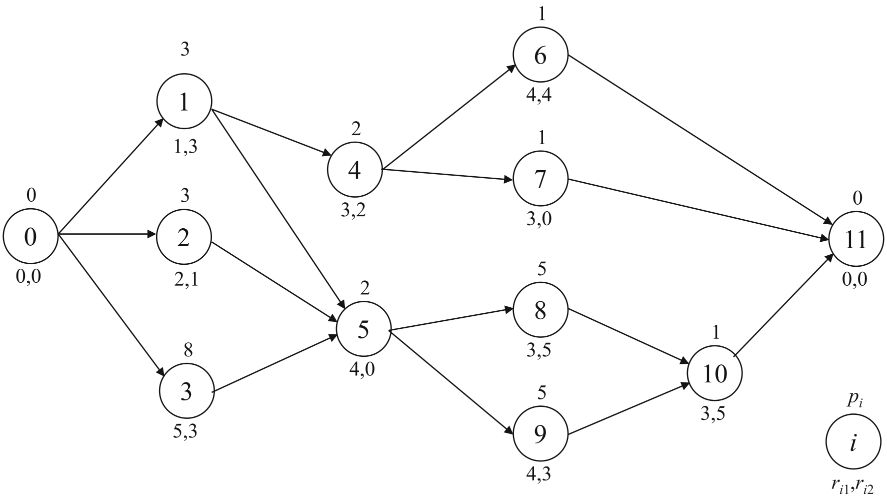


Fig. 16.1 Network of the example project

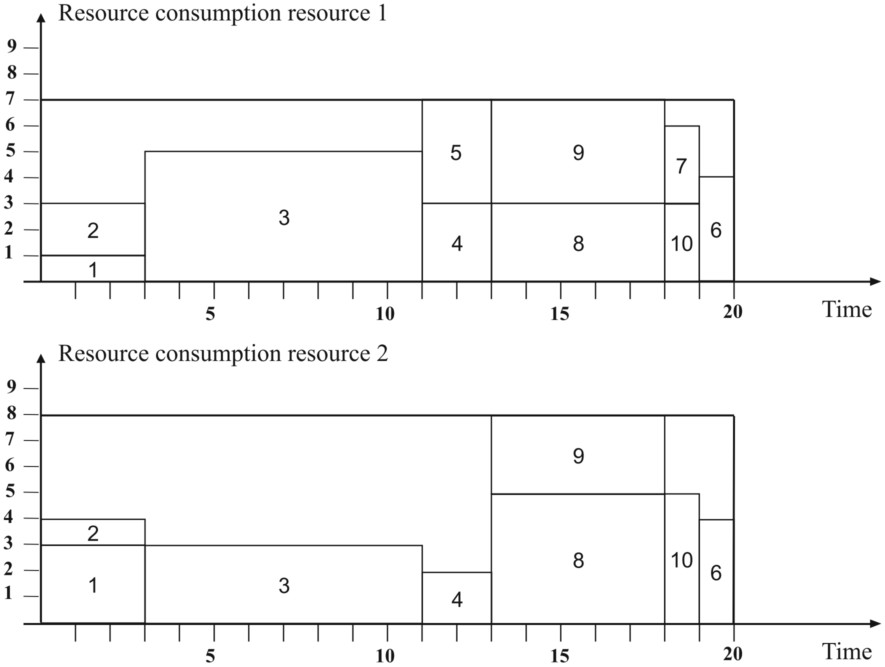


Fig. 16.2 Optimal resource profiles RACP if tardiness is not allowed

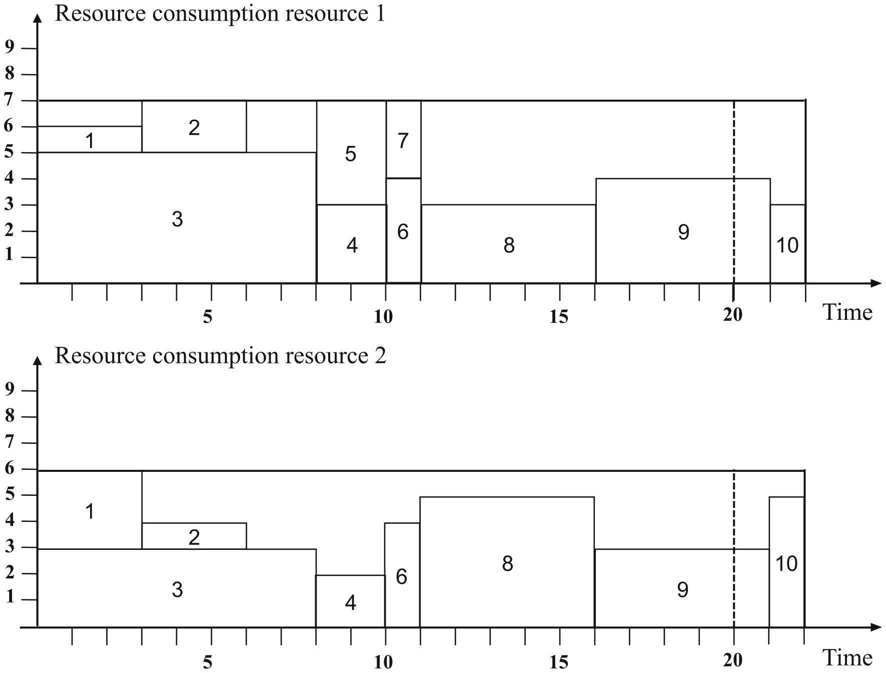


Fig. 16.3 Optimal resource profiles RACPT if tardiness is allowed

16.3 (Meta)heuristic Solution Procedures

The resource availability cost problem was introduced by Möhring (1984), analyzing the construction of a bridge as a practical case study. In order to finish the construction in a predefined deadline, sufficient resources need to be provided in such a way that the total financial investment was minimized. For that reason, the problem was initially known as the resource investment problem. To solve this problem, Möhring (1984) proposes an exact procedure based on graph theoretical algorithms for comparability graph and interval graph recognition and orientation. Later, Demeulemeester (1995) and Rodrigues and Yamashita (2010), amongst others, both developed an effective optimal algorithm for the RACP. For an overview of the existing exact procedures, the reader is referred to Chap. 15 of this handbook.

Since the RACP is an $\mathcal{NP}$-hard problem (Möhring 1984), meaning that optimal solution procedures can only be used for relatively simple and small problem instances, a handful of (meta)heuristic solution procedures were developed in the past decade in order to solve the problem for large sized projects.

Drexl and Kimms (2001) presented two algorithms in order to obtain lower bounds: an algorithm based on Lagrangian relaxation combined with subgradient optimization and an algorithm that employs column generation techniques. Both methods are able to obtain improved lower bounds, compared to the rather simple lower bounds based on the resource demand and availability for each resource type, and improved upper bounds, compared to the upper bound defined by the earliest start schedule.

Yamashita et al. (2006) were the first to propose a metaheuristic solution procedure for the resource availability cost problem. In their paper, they presented a multi-start heuristic and a scatter search procedure. In both procedures, a solution vector is represented by an activity list and a capacity list. A two-stage improvement heuristic is developed in order to make infeasible solution vectors feasible (first stage) and to improve them (second stage). This improvement heuristic is applied in combination with the path relinking method, which generates new population elements by gradually reducing the distance between two solution vectors, i.e. the initiating and guiding solution (Glover et al. 2000). Computational results show the robustness of the scatter search procedure, both for small and large project sizes.

Shadrokh and Kianfar (2007) presented a genetic algorithm to solve the RACP when tardiness of the project is permitted with defined penalty costs. If a large penalty cost is considered, the algorithm can also be used to solve the RACP. The algorithm makes use of chromosomes, which are represented by a priority list and an availability list of resources and is using both the serial and parallel scheduling schemes to construct schedules.

Ranjbar et al. (2008) propose a path relinking method to solve the RACP, as well as a genetic algorithm. Solution vectors are only represented by an activity list. Based on this list, a schedule is generated using a schedule-generation scheme which is based on the procedure of Burgess and Killebrew (1962), trying to level the use of the different resource types. Computational results revealed that the path relinking method outperformed the genetic algorithm procedure.

Van Peteghem and Vanhoucke (2013) developed an artificial immune system algorithm, which makes use of mechanisms inspired by the vertebrate immune system. A solution vector is represented by an activity list and a capacity list. The procedure includes a new fitness function, a probability function for the composition of the capacity lists, and a *K-means* density function in order to avoid premature convergence. Although comparison is difficult, the authors show that their algorithm performs better than the procedure of Shadrokh and Kianfar (2007).

Several extensions for the resource availability cost problem were presented as well. Shadrokh and Kianfar (2007) proposed a genetic algorithm to solve the RACPT when tardiness of the project is permitted with defined penalty costs. In this problem, a delay in the completion of the project is allowed, but for every time unit the project finishes later than the predefined deadline $\overline{d}$, a fixed penalty cost increases the total cost. The objective of this problem is to minimize the sum of the resource availability costs and this tardiness cost. Yamashita et al. (2006) extended their scatter search procedure and multi-start heuristic for the RACP with scenarios (RACPS), in which uncertainty was added to the activity durations. Finally, a priority rule heuristic for multi-mode version of the RACP was presented by Hsu and Kim (2005).

Problems similar to the RACP are the resource renting problem (RRP) and the time-constrained project scheduling problem (TCPSP). In the RRP, renewable resources have to be rented in such a way that the total renting cost is minimized. This cost contains per resource type a fixed cost per unit and a variable renting cost per unit and per time period. In case the variable renting cost is equal to 0, the RRP is similar to the RACP. Solution procedures for this problem are proposed by Nübel (2001) and Ballestín (2007). In the TCPSP, only additional resources that need to be hired above the available resources such that the project can be executed in the predefined deadline have a cost. The objective in this problem is to minimize these extra hiring costs. Procedures for this problem are proposed by Kolisch (1995), Neumann and Zimmermann (1999), Guldemond et al. (2008) and Kreter et al. (2014).

16.4 Invasive Weed Optimization Algorithm

The Invasive Weed Optimization algorithm is a computational algorithm proposed by Mehrabian and Lucas (2006) and inspired by the natural behavior of weeds in colonizing and finding a suitable place for growth and reproduction. The algorithm originates from the idea that weeds are one of the most robust plants in agriculture and that after thousands of years of hand-weeding and decades of herbicides, weeds are still in the same fields. This property has led to the fact that the expression “the weeds always win” has become a common belief in agronomy. Weeds are able to adapt themselves to their environment and change their behavior and get better and fitter, due to the weed biology and ecology.

The reproduction of weeds depends on the type of plant. In most cases, a plant begins its life history when the egg in the pistil is fertilized by pollen and forms a seed in the parent plant. These seeds invade a cropping system by means of dispersal (*spatial dispersal*) and occupy opportunity spaces between the crops. The seeds germinate and grow to flowering weeds, each reproducing new weeds, independently. The number of new weeds depends on the fitness of the flowering weed in the colony: the better the weeds adapt to the environment and the more unused resources are taken, the faster the seeds grow and the more seeds are produced. Since the carrying capacity of the environment is limited and restricted, only those weeds with better fitness can survive and produce new weeds. Due to this competitive contest between the weeds (*competitive exclusion*), the weeds become well adapted and improved over time.

16.5 An Invasive Weed Algorithm for the RACP(T)

These efficient mechanisms of the weed ecology make invasive weed optimization algorithms useful for optimization problems. Recently, IWO is proposed for electromagnetic applications (Karimkashi and Kishk 2010) and for the design of encoding

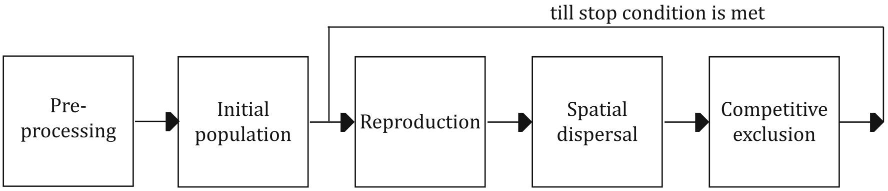


Fig. 16.4 Invasive Weed Optimization Algorithm: procedure

sequences for DNA computing (Zhang et al. 2009). However, to the best of our knowledge, this search strategy has not been used to optimize scheduling problems. In this section, a problem-solving technique for the RACP and RACPT based on the principles of the invasive weed ecology is presented. The different generic steps in our algorithm are presented in Fig. 16.4 and will be discussed along the following subsections.

16.5.1 Representation and Schedule-Generation Scheme

A solution procedure for the RACP does not operate directly on a schedule, but on a representation of a schedule that is convenient and effective for the functioning of the algorithm. Therefore, each individual I in the weed colony is constructed from two lists: an activity list and a capacity list.

The activity list determines the sequence in which the activities are scheduled. In this text, the activity list (AL) representation is used, in which the position ℓ_i of an activity i determines the relative priority of that activity versus the other activities. In order to avoid infeasible solutions, the activity list is always precedence feasible, which means that the precedence relations between the different activities are met. Ranjbar et al. (2008) indicate such an AL as a precedence feasible activity list (PFAL). Next to the activity list, a resource capacity list R is used to determine the available resource capacities. This capacity list is resource feasible, i.e. $R_k \geq \max_{i=1,\ldots,n}\{r_{ik}\}, \forall k$.

A schedule-generation scheme (SGS) translates the individual $I = (\ell^I, R^I)$ into a schedule S. In this study, the serial schedule-generation scheme is used (Kelley Jr. 1963). This scheme sequentially adds activities to the schedule one-at-a-time. In each iteration, the next activity in the activity list is chosen and that activity is assigned to the schedule as soon as possible within the precedence and resource constraints.

16.5.2 Preprocessing

Before the algorithm is started, a preprocessing procedure is applied. This procedure minimizes the search space for the capacity list and sets a minimum availability R_k^{min} and maximum availability R_k^{max} to each resource k and calculates the initial best cost f^{best}. The preprocessing procedure that is used in this algorithm is the one as described in Van Peteghem and Vanhoucke (2013). The procedure contains three subroutines and can be shortly explained along the following lines.

1. *Minimal resource demand* Based on the required resource demand for each activity, the minimal resource availability R_k^{min} can be determined for each renewable resource k.

   $$R_k^{min} = \max(\frac{\sum_{i=1}^{n} r_{ik} p_i}{\overline{d}}, \max_{i=1,\ldots,n}\{r_{ik}\})$$

   The minimum resource availabilities are tightened with the critical sequence lower bound method of Stinson et al. (1978).
2. *Heuristic upper bound solution* The heuristic upper bound solution is determined by applying the resource leveling procedure of Burgess and Killebrew (1962). This procedure (further denoted as BK62) starts with the earliest start schedule and iteratively decreases the objective function (i.e. min. $\sum_{t=0}^{\overline{d}} \sum_{k=1}^{K} r_k^2(S,t)$, with $r_k(S,t)$ the use of resource k at time t) by delaying activities according to a predefined activity list. The upper bound solution is calculated as $\sum_{k=1}^{K} c_k u_k^{max}$, where u_k^{max} is the maximal resource use for each resource k in the obtained schedule. The upper bound solution is used as initial best cost f^{best}.
3. *Maximal resource availability* To calculate the maximum availability per resource type, Van Peteghem and Vanhoucke (2013) developed an inverse BK62 procedure. This heuristic procedure, based on the original procedure of Burgess and Killebrew (1962), starts from the earliest start schedule, but activities are only delayed if they increase the objective function, i.e. max. $\sum_{t=0}^{\overline{d}} \sum_{k=1}^{K} r_k^2(S,t)$, which means that peaks in the project schedule are stimulated. The activities are delayed according to a predefined activity list. In order to avoid the escape of the optimal solution, the procedure is repeated with random generated activity lists until J consecutive unsuccessful trials to increase the maximal resource availability R_k^{max} have been made (in this chapter, J is equal to 5).

   The obtained value for R_k^{max} can further be tightened by using the following formula:

   $$v_k^{max} = \left\lfloor \frac{f^{best} - f_{k \notin \mathscr{R}}^{min}}{c_k} \right\rfloor \quad \forall k \in \mathscr{R}$$

where $f^{min}_{k \notin \mathscr{R}}$ is the minimal cost determined by the minimum resource availabilities of all resources, excluding the cost of resource k. The maximal resource availability R^{max}_k then can be calculated as the minimum of u^{max}_k and v^{max}_k, i.e. $R^{max}_k = \min(u^{max}_k, v^{max}_k), \forall k$.

By applying this preprocessing procedure, an initial best cost f^{best} and the feasible ranges for each resource k, $[R^{min}_k, R^{max}_k]$, are determined.

16.5.3 Fitness Function

In order to measure the fitness of an individual weed, a fitness value is calculated. In the RCPSP, the fitness value normally equals the makespan of the project C_{max}. It is a good measure for sequencing the different individuals according to their contribution to the objective of the problem. However, using the makespan or even the cost as fitness value for an individual is inappropriate in the RACP. Since most individuals will be infeasible (if feasible, it is unnecessary to search further to individuals which result in the same cost), some authors also use the term *unfitness function* (Shadrokh and Kianfar 2007).

In literature, three different fitness functions are proposed.

1. Yamashita et al. (2006) use the objective function of the problem, i.e. the total resource usage cost, to differentiate between the different feasible solutions. The fitness function is only calculated if the solution vector itself is feasible, i.e. if the project can be scheduled within the predefined deadline $\overline{d}$. If this fitness function is used, infeasible solutions can not be part of the population of solution vectors.

$$F_I = \sum_{k=1}^{K} c_k \cdot R_k \tag{16.7}$$

2. Shadrokh and Kianfar (2007) introduced the unfitness function, which can be formulated as follows

$$F_I = \sum_{k=1}^{K} c_k \cdot R_k + w^T_{n+1} \cdot \max(0, S^I_{n+1} - \overline{d}) \tag{16.8}$$

   with w^T_{n+1} the tardiness cost. This penalty function allows infeasible solutions to remain part of the population of solution vectors.

3. Van Peteghem and Vanhoucke (2013) state that an individual which can be scheduled in a time $S_{n+1} > \overline{d}$, but close to the deadline and at a relatively low cost must be favored with respect to an individual which can be scheduled within the deadline but at a relatively higher cost. They therefore formulated a fitness function as follows:

$$F_I = \frac{f^I}{f^{min}} \cdot \frac{\overline{d} + \max(0, S^I_{n+1} - \overline{d})}{\overline{d}}$$

The first part indicates the relative cost of the individual compared to the minimal cost (based on the minimal resource use). A second part indicates the deadline feasibility. If the value of this second part is equal to 1, the individual is deadline feasible (i.e. $S^I_{n+1} \leq \overline{d}$). If the individual weed is deadline infeasible ($S^I_{n+1} > \overline{d}$), the value of this part increases the more the finish time exceeds the predefined deadline. If F_I is equal to 1, the best cost is equal to the minimal cost and the algorithm can be stopped.

Previous computational tests have shown that the fitness function of Van Peteghem and Vanhoucke (2013) outperformed the fitness function of Shadrokh and Kianfar (2007). However, for the RACPT-procedure, the former fitness function is not applicable and therefore the fitness function of Shadrokh and Kianfar (2007) will be used for these problems.

16.5.4 Initial Population Generation

During the initial population generation, a set of σ_{pop} population elements (i.e. weeds) is generated. In order to obtain a diversified initial population of solution vectors, the initial starting solutions are generated randomly. For each individual I in the population, a precedence feasible activity list (PFAL) ℓ^I is determined randomly and a resource capacity list R^I is set. This resource capacity list is generated by searching a resource availability R^I_k for each resource k such that the total cost $f^I = \sum_{k=1}^{K} c_k R_k$ is smaller than or equal to f^{best} and such that $R^{min}_k \leq R^I_k \leq R^{max}_k, \forall k$.

16.5.5 Reproduction

Based on the fitness value of the different elements in the population, the number of seeds (n^I_{seed}) is determined for each colony element I. This number depends on the fitness of the plant itself (F_I) and on the colony's lowest and highest fitness (F_{min} and F_{max}, respectively). The number decreases linearly from the maximum possible seed production n^{max}_{seed} to its minimum n^{min}_{seed}. The number of seeds can be calculated as follows:

$$n^I_{seed} = \frac{F_I - F_{min}}{F_{max} - F_{min}} (n^{max}_{seed} - n^{min}_{seed}) + n^{min}_{seed}$$

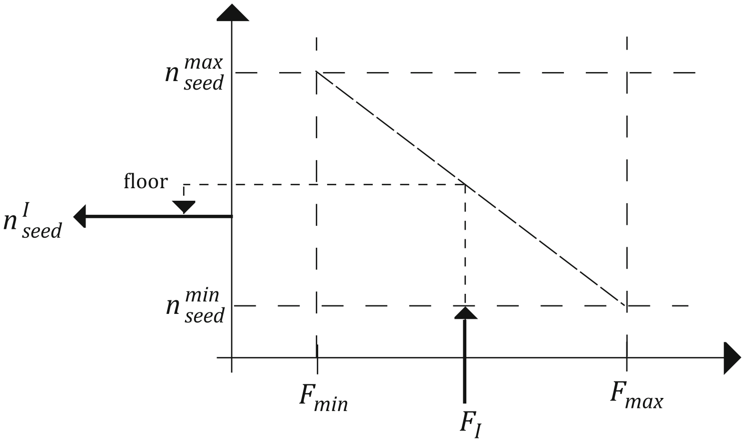


Fig. 16.5 Seed reproduction procedure

In Fig. 16.5, the reproduction technique is illustrated. This procedure gives a chance to infeasible individuals to survive and reproduces new seeds, similar to the mechanism happening in nature (Mehrabian and Lucas 2006). Since all weeds should have a chance to reproduce, the minimum number of seeds n_{seed}^{min} is equal to 1.

16.5.6 Spatial Dispersal

The generated seeds are randomly distributed over the search space. This process is applied in two phases for each new seed: first, a mutation procedure is applied on the activity list and second, a new capacity list is generated.

16.5.6.1 Activity List

Each of the activity lists is mutated for a certain number of times. However, the number of mutations will be reduced from a predefined maximum number of mutations n_{mut}^{max} to a final minimal number of mutations n_{mut}^{min}, in every generation. The number of mutations is calculated as follows:

$$n_{mut}^{I} = \frac{n_{iter}^{max} - n_{iter}^{I}}{n_{iter}^{max}}(n_{mut}^{max} - n_{mut}^{min}) + n_{mut}^{min}$$

where n_{iter}^{max} is the maximum number of iterations and n_{iter}^{I} is the current number of iterations. By using this formula, the neighborhood of the different elements is narrowed in every new generation, until a minimum number of mutations is obtained. The minimum number of mutations is equal to 1.

16.5.6.2 Capacity List

For each new seed which is created, a new capacity list is determined. The cost of the new capacity list is always set smaller than or equal to the best cost f^{best}. The capacity list is generated based on a probability function P_{kR_k}, which sets for every resource availability level R_k $(R_k^{min} \leq R_k \leq R_k^{max}, \forall k)$ the chance that the specific value is chosen for a specific resource type k. This chance is determined by the inverse of the sum of all the differences between the finish time of a project S_{n+1} and the deadline $\overline{d}$, if $S_{n+1} > \overline{d}$, obtained by the individuals which have a value of R_k in their capacity list for resource k. By this manner, the resource availabilities which create feasible schedules have a larger chance to be chosen than resource availabilities resulting in infeasible schedules (i.e. $S_{n+1} > \overline{d}$). However, the set of differences is smoothed every generation with a smooth factor SF $(0 < SF < 1)$, in order to decrease the impact of the differences from the previous generations.

Each new seed (combination of an adapted activity list and a new capacity list) is evaluated using the serial schedule-generation scheme, as explained in Sect. 16.5.1. On the obtained solution, a local search procedure is applied.

16.5.7 Local Search

A local search procedure is applied on each element in the weed colony. If the duration of the schedule is smaller than or equal to the deadline $(S_{n+1} \leq \overline{d})$, the *resource leveling local search* is applied, in order to improve the total resource cost. If the duration of the schedule is larger than the deadline $(S_{n+1} > \overline{d})$, a *forward-backward local search* is applied, in order to decrease the project finish time.

16.5.7.1 Resource Leveling Local Search

The resource leveling local search is an extension of the Burgess and Killebrew procedure, as explained in Sect. 16.5.2. The procedure starts from the schedule obtained from the SGS and iteratively tries to improve the objective function by delaying activities according to the activity list of the weed. The objective function of this adapted BK62 procedure is min. $\sum_{t=0}^{\overline{d}} \sum_{k=1}^{K} c_k r_k^2(S,t)$, which means that the more expensive resources outweigh the less expensive resources. The procedure is repeated until no further improvements occur.

16.5.7.2 Forward-Backward Local Search

This technique, proposed by Li and Willis (1992), transforms left-justified schedules (where all activities are scheduled as soon as possible) into right-justified schedules (where all activities are scheduled as late as possible) and vice versa. During the backward (forward) scheduling stage, the makespan of the schedule is tried to be reduced by shifting the activities, in a sequence determined by the finish (start) times, as much as possible to the right (left), without affecting the project completion time. During the forward-backward procedure only improvements can occur. This procedure is applied until no further improvements occur.

16.5.8 Competitive Exclusion

All weeds (both new weeds and parent weeds) are then ranked according to their fitness value. Weeds with higher values are eliminated from the colony and only the weeds with the better fitness survive and are allowed to replicate.

However, in order to maintain diversity in the colony, only the $\zeta_{best}\%$ weeds are included. For every other individual weed that is added to the new colony, the weed should be divers with respect to all the other members which are already in the new colony. The diversity function is defined as the number of differences in ℓ^I and ℓ^J, the activity lists of weeds I and J, respectively, and can be formulated as follows:

$$\Delta_{IJ} = \sum_{i=1}^{n} \begin{cases} 1 \text{ if } \ell_i^I = \ell_i^J \\ 0 \text{ otherwise} \end{cases}$$

where the value Δ_{IJ} should be smaller than a threshold value TV (which is calculated as a percentage ζ_{excl} of the total number of real activities n) to allow membership in the new colony. If the number of weeds in the population is smaller than σ_{pop}, the colony is seeded with new random elements in order to maintain the diversity in the population.

16.6 Computational Results

In order to prove the efficiency of our algorithm, we have tested our IWO solution procedure on a test set which is defined in Sect. 16.6.1. In Sect. 16.6.2, the Taguchi method is used to configure the algorithmic parameter settings, while in Sect. 16.6.3 the performance and effectiveness of the algorithm on the RACP and RACPT is tested. In Sect. 16.6.4, a comparison with other solution procedures is made on the proposed dataset and on the PSPLIB dataset instances.

16.6.1 Benchmark Problem Set

The procedure is coded in Xcode 3.2 and all computational experiments are performed on an Apple computer with 2 GB of internal memory and an Intel Core 2 Duo (2.26 GHz) processor. We have validated the procedure on a dataset with projects of 30 activities and 4 resources. The order strength (*OS*), which gives an indication of the network complexity (Mastor 1970), equals 0.25, 0.50 or 0.75. The resource factor (*RF*), which indicates the density of the different resource types needed by an activity (Pascoe 1966), equals 0.25, 0.50, 0.75 or 1. For each project setting, 20 instances were generated. In total, 240 different project instances were generated. For the generation of the instances, we have used the RanGen project scheduling instance generator developed by Vanhoucke et al. (2008). The deadline $\overline{d}$ of the project is set at $\overline{d} = (1 + \theta)ES_{n+1}$, with ES_{n+1} the earliest finish time of the project and θ equal to 0, 0.1, 0.2, 0.3, 0.4, and 0.5. The cost of the resources is drawn randomly from a uniform distribution U[1,5] for each resource. In total, 1,440 different project situations are tested. The dataset (called *RACP30* in the remainder of this chapter) and the best results found for each instance are available on the website http://www.projectmanagement.ugent.be and can be used by researchers to compare the results of their solution procedures.

In order to make a fair, platform-independent comparison possible, the evaluation is stopped after a predefined number of generated schedules. In this chapter, the number is set at 5,000 schedules. Kolisch and Hartmann (2006) state in their RCPSP review paper that one schedule corresponds to (at most) one start time assignment per activity, as done by a SGS. According to the authors, the advantage of the number of schedules as stop criterion is twofold: first, it is *platform-independent* and second, future studies can easily make use of the *benchmark results* by applying the same stop criterion. However, the stop criterion also has a few shortcomings. First, it cannot be applied to all heuristic strategies. Second, the required time to compute one schedule might differ between metaheuristics. Nevertheless, Kolisch and Hartmann conclude that limiting the number of schedules is the best criterion available for a broad comparison, which motivated us to use this stop criterion.

16.6.2 Parameter Setting

In order to determine a suitable set of parameters for our invasive weed optimization algorithm, the Taguchi method is used (Montgomery 2005). This method involves reducing the variation in the process through robust design of experiments (DOE).

The algorithm contains the following six key parameters: the values σ_{pop}, n_{seed}^{max}, n_{mut}^{max}, SF, ζ_{best} and ζ_{excl}. According to the number of levels (five for each parameter), the orthogonal array L25 is chosen. The values which are chosen for each parameter are shown in Table 16.1.

Table 16.1 Different parameter values

Factor level	σ_{pop}	n_{seed}^{max}	n_{mut}^{max}	SF	ζ_{best}	ζ_{excl}
1	20	5	2	25 %	20 %	3 %
2	30	6	3	30 %	30 %	5 %
3	40	7	4	35 %	40 %	7 %
4	50	8	5	40 %	50 %	9 %
5	60	9	6	45 %	60 %	11 %

Table 16.2 Response table with 5,000 schedules

Factor level	σ_{pop}	n_{seed}^{max}	n_{mut}^{max}	SF	ζ_{best}	ζ_{excl}
1	91.43	90.06	94.63	90.31	**88.60**	90.73
2	88.57	**89.02**	91.27	91.27	89.76	89.97
3	**87.59**	89.03	**86.80**	89.61	90.36	89.94
4	90.65	89.98	88.52	89.07	88.92	89.54
5	90.01	90.17	87.04	**88.01**	90.62	**88.09**

In Table 16.2, an overview is given of the average of the average deviation from the best found solution after 5,000 schedules for each of the different parameters and levels (expressed in ‰). According to the factor level trends, the best combination of parameter values for the algorithm is determined, as indicated in bold in Table 16.2.

16.6.3 Performance

In this section, the effectiveness and performance of the algorithm is tested. In Sect. 16.6.3.1, the performance of the algorithm on the resource availability cost problem will be shown, while in Sect. 16.6.3.2, results are discussed with respect to the performance of the algorithm on the problem with tardiness.

16.6.3.1 RACP

In Table 16.3, a detailed overview is given for each of the different values of θ and the different values of OS and RF. The result for each setting is the percentage improvement of the upper bound, according to the formula presented by Drexl and Kimms (2001). As can be seen, the percentage improvement increases for decreasing values of OS and increasing values of RF. The results also improve for larger values of θ.

Table 16.3 Detailed results IWO algorithm (5,000 schedules)

		θ						
		0	0.1	0.2	0.3	0.4	0.5	Global
$OS = 0.25$	$RF = 0.25$	35.49	39.96	45.73	49.51	53.12	55.02	46.47
	$RF = 0.5$	40.32	44.64	50.05	53.63	56.91	59.44	50.83
	$RF = 0.75$	39.50	44.66	49.35	53.65	56.59	59.16	50.49
	$RF = 1$	40.09	44.62	49.61	53.16	56.49	58.97	50.49
$OS = 0.5$	$RF = 0.25$	29.10	36.62	41.00	43.35	44.31	45.15	39.92
	$RF = 0.5$	32.22	40.52	47.50	52.32	55.20	57.52	47.55
	$RF = 0.75$	31.81	39.29	45.42	49.66	52.65	55.49	45.72
	$RF = 1$	33.94	41.58	46.81	50.64	54.20	57.07	47.37
$OS = 0.75$	$RF = 0.25$	18.41	29.96	34.76	36.69	36.82	37.02	32.28
	$RF = 0.5$	19.06	30.19	38.85	43.76	47.69	50.80	38.39
	$RF = 0.75$	28.89	38.99	44.48	47.91	51.21	53.79	44.21
	$RF = 1$	24.90	36.68	42.79	47.64	50.62	53.20	42.64
Overall		31.14	38.98	44.70	48.49	51.32	53.55	44.70

Table 16.4 Detailed results IWO algorithm (5,000 schedules)

τ	0	12.5	25	37.5	50	62.5	75	87.5	100	M
Av. dev. RACP (%)	−37.55	−15.98	−5.93	−2.66	−1.27	−0.80	−0.51	−0.28	−0.16	0.00
$S_{n+1} > \overline{d}$ (%)	93.26	84.79	67.57	50.56	32.85	23.26	15.28	10.00	2.50	0.00
Av. dev. $\overline{d}$ (%)	145.49	69.46	19.04	6.28	2.02	0.91	−0.01	−0.43	−0.82	−0.92

16.6.3.2 RACPT

Table 16.4 shows the results of the algorithm if the problem with tardiness is solved. The tardiness cost w_{n+1}^T is equal to τ percent of the sum of the resource unit costs, $\sum_{k=1}^{K} c_k$, with τ equal to 0, 12.5, 25, 37.5, 50, 62.5, 75, 87.5, 100 and M (with M a big value). If the value of τ is equal to M, the problem can be seen as the RACP without tardiness. If τ is equal to 0, the value of w_{n+1}^T is equal to 0, otherwise the following formula can be used:

$$w_{n+1}^T = \max(1, \lfloor \frac{\tau}{100} \sum_{k=1}^{K} c_k \rfloor)$$

For each of the different values of τ the average deviation from the RACP-cost is given (Av. dev. RACP), the percentage of times the finish time is larger than the deadline ($S_{n+1} > \overline{d}$) and the average percentage deviation from the deadline (Av. dev. $\overline{d}$).

When τ is equal to 0, the solution is equal to the minimum cost f^{min}. As can be seen, the average deviation from the RACP-solution is almost 38 %. This percentage

Table 16.5 Comparison between AIS and IWO on RACP30 dataset (5,000 schedules)

		θ						
Cost	Algorithm	0	0.1	0.2	0.3	0.4	0.5	Global
$c_k = 1$	IWO	30.84	38.54	44.09	47.94	50.76	53.00	44.19
	AIS	30.83	38.49	44.00	47.80	50.61	52.88	44.10
$c_k \in [1, 5]$	IWO	31.14	38.98	44.70	48.49	51.32	53.55	44.70
	AIS	31.19	39.05	44.74	48.53	51.36	53.55	44.74

decreases significantly for an increasing value of τ. The percentage of solutions where the finish time is larger than the deadline also decreases for an increasing tardiness costs. The negative average deviation from the deadline $\overline{d}$ for large values of τ can be explained by the fact that—on average—in a limited number of cases the finish time is smaller than the predefined deadline. Moreover, the value of this variable for the RACP-procedure is −0.92 %.

16.6.4 Comparison

Finally, the results obtained from the IWO are compared with other existing metaheuristic solution procedures in literature on the RACP30 dataset presented in Sect. 16.6.1 and on the well-known PSPLIB dataset (Kolisch et al. 1995).

16.6.4.1 RACP30

First, a comparison is performed on the RACP30 dataset. The algorithm is compared with the artificial immune system (AIS) algorithm of Van Peteghem and Vanhoucke (2013). In Table 16.5, a detailed overview of the average percentage deviation from the upperbound ($\Delta^{\emptyset}_{UB}$) for both algorithms is given for each of the different values of θ. The test is also performed using two different cost structures: one where all resource costs are equal to 1, and one where the cost of the resources is drawn randomly from a uniform distribution U[1,5] for each resource.

In case where $c_k \in [1, 5]$, the AIS algorithm outperforms the IWO algorithm for all values of θ, except for $\theta = 50$. The inverse is true for the case where $c_k = 1$, where the IWO algorithm outperforms the AIS algorithm for all cases.

16.6.4.2 PSPLIB

Second, the algorithm is tested on the well-known PSPLIB dataset (Kolisch et al. 1995). This dataset contains instances involving 4 resource types and 30, 60, 90, and 120 activities, with different parameters for the network complexity

Table 16.6 Comparison between metaheuristics on J30 dataset (5,000 schedules)

	θ	0	0.1	0.2	0.3	0.4	0.5
Shadroh and Kianfar	Av. Imp.	29.57	38.11	42.63	46.36	48.97	50.95
IWO	Av. Imp.	30.33	39.05	44.33	47.78	50.33	52.31
	St. Dev.	0.16	0.13	0.13	0.11	0.11	0.10
	Min.	30.02	38.83	44.09	47.63	50.15	52.14
	Max.	30.63	39.33	44.59	47.97	50.53	52.52
AIS	Av. Imp.	30.34	39.03	44.25	47.67	50.23	52.19
	St. Dev.	0.17	0.16	0.14	0.15	0.15	0.16
	Min.	29.87	38.64	43.93	47.34	49.90	51.87
	Max.	30.66	39.31	44.49	47.89	50.51	52.42

($NC \in \{1.5, 1.8, 2.1\}$) and the resource factor ($RF \in \{0.25, 0.5, 0.75, 1.0\}$). The instances with 30 activities constitute the J30 dataset.

In the paper of Shadrokh and Kianfar (2007), the average percentage of improvement of the upper bound (Av. Imp.) is given for the J30 dataset (with a stop criterion of 2 s). Shadrokh and Kianfar (2007) set the unit resource cost for each resource equal to a random number from the interval [1, 10]. Although the set of costs which is used in their experiments is not available, the following experiment is set up in order to prove the effectiveness of the IWO algorithm, similar as done in the paper of Van Peteghem and Vanhoucke (2013).

The IWO algorithm is performed 20 times on each problem instance of the J30 dataset, each time with a different value for the resource cost c_k ($\forall k$), randomly chosen from the uniform distribution U[1,10]. In Table 16.6, the average and the standard deviation of the average percentage deviation from the upper bound is shown, as well as the results obtained from the paper of Van Peteghem and Vanhoucke (2013). Moreover, the minimum and maximum average percentage deviation is also shown.

Compared to the results of Shadrokh and Kianfar (2007) and Van Peteghem and Vanhoucke (2013), the following conclusions can be drawn:

- In all cases, the IWO algorithm performs better than the genetic algorithm of Shadrokh and Kianfar (2007) and the AIS algorithm of Van Peteghem and Vanhoucke (2013).
- Moreover, the algorithm of Shadrokh and Kianfar (2007) has a stop criterion of 2 s (on a computer with 1 GHz processor), while the stop criterion for the IWO and AIS algorithm is 5,000 schedules with an average CPU-time of less than 0.5 s (on a computer with a 2.26 GHz processor).
- Based on the number of experiments, the average improvement and the standard deviation, a 99.7 % confidence interval can be found with a stochastic lower endpoint which is larger than the average improvement obtained by Shadrokh and Kianfar (2007).

16.7 Conclusions

In this chapter, an invasive weed optimization algorithm is presented for the resource availability cost problem. In this scheduling problem, the total cost of the (unlimited) renewable resources required to complete the project by a prespecified project deadline should be minimized. This search strategy is inspired by the natural behavior of weeds in colonizing and finding a suitable place for growth and reproduction.

The proposed procedure can also be used to solve the resource availability cost problem with tardiness. In this problem, a due date for the project is set, but tardiness of the project is permitted with a predefined penalty. The total project cost is than equal to the sum of the resource availability cost and the tardiness cost. The proposed procedure proved being successful for both the RACP and RACPT.

Future research should focus on the use of this search strategy for other project scheduling problems such as the discrete time-resource trade-off problem (DTRTP) or extensions of the RACP. Moreover, the examination of the influence of varying cost parameters and the introduction of bonus and penalty costs could lead to interesting managerial insights.

References

Ballestín F (2007) A genetic algorithm for the resource renting problem with minimum and maximum time lags. In: Cotta C, van Hemert J (eds) EvoCOP 2007. Lecture notes in computer science, vol 4446. Springer, Heidelberg, pp 25–35

Brucker P, Drexl A, Möhring R, Neumann K, Pesch E (1999) Resource-constrained project scheduling: notation, classification, models, and methods. Eur J Oper Res 112:3–41

Burgess A, Killebrew J (1962) Variation in activity level on a cyclic arrow diagram. J Ind Eng 2:76–83

Demeulemeester E (1995) Minimizing resource availability costs in time-limited project networks. Manag Sci 41:1590–1598

Drexl A, Kimms A (2001) Optimization guided lower and upper bounds for the resource investment problem. J Oper Res Soc 52:340–351

Glover F, Laguna M, Marti R (2000) Fundamentals of scatter search and path relinking. Control Cybern 29:653–684

Guldemond TA, Hurink JL, Paulus JJ, Schutten JMJ (2008) Time-constrained project scheduling. J Sched 11:137–148

Hartmann S, Briskorn D (2010) A survey of variants and extensions of the resource-constrained project scheduling problem. Eur J Oper Res 207:1–15

Herroelen W, De Reyck B, Demeulemeester E (1999) A classification scheme for project scheduling. In: Węglarz J (ed) Handbook of recent advances in project scheduling. Kluwer, Dordrecht, pp 1–26

Hsu C-C, Kim D (2005) A new heuristic for the multi-mode resource investment problem. J Oper Res Soc 56:406–413

Karimkashi S, Kishk A (2010) Invasive weed optimization and its features in electro-magnetics. IEEE Trans Antennas Propag 58:1269–1278

Kelley J Jr (1963) The critical-path method: resources planning and scheduling. Prentice-Hall, Englewood Cliffs

Kolisch R (1995) Project scheduling under resource constraints. Physica, Berlin

Kolisch R, Hartmann S (2006) Experimental investigation of heuristics for resource-constrained project scheduling: an update. Eur J Oper Res 174:23–37

Kolisch R, Sprecher A, Drexl A (1995) Characterization and generation of a general class of resource-constrained project scheduling problems. Manag Sci 41:1693–1703

Kreter S, Rieck J, Zimmermann J (2014) The total adjustment cost problem: applications, models, and solution algorithms. J Sched 17:145–160

Li K, Willis R (1992) An iterative scheduling technique for resource-constrained project scheduling. Eur J Oper Res 56:370–379

Mastor A (1970) An experimental and comparative evaluation of production line balancing techniques. Manag Sci 16:728–746

Mehrabian AR, Lucas C (2006) A novel numerical optimization algorithm inspired from weed colonization. Ecol Inform 1:355–366

Möhring R (1984) Minimizing costs of resource requirements in project networks subject to a fixed completion time. Oper Res 32:89–120

Montgomery D (2005) Design and analysis of experiments. Wiley, Hoboken

Neumann K, Zimmermann J (1999) Resource levelling for projects with schedule-dependent time windows. Eur J Oper Res 117:591–605

Nübel H (2001) The resource renting problem subject to temporal constraints. OR Spektrum 23:574–586

Pascoe T (1966) Allocation of resources: CPM. Revue Française de Recherche Opérationnelle 38:31–38

Ranjbar M, Kianfar F, Shadrokh S (2008) Solving the resource availability cost problem in project scheduling by path relinking and genetic algorithm. Appl Math Comput 196:879–888

Rodrigues S, Yamashita D (2010) An exact algorithm for minimizing resource availability costs in project scheduling. Eur J Oper Res 206:562–568

Shadrokh S, Kianfar F (2007) A genetic algorithm for resource investment project scheduling problem, tardiness permitted with penalty. Eur J Oper Res 181:86–101

Stinson J, Davis E, Khumawala B (1978) Multiple resource-constrained scheduling using branch-and-bound. AIIE Trans 10:252–259

Van Peteghem V, Vanhoucke M (2013) An artificial immune system algorithm for the resource availability cost problem. Flex Serv Manuf J 25:122–144

Vanhoucke M, Coelho J, Debels D, Maenhout B, Tavares L (2008) An evaluation of the adequacy of project network generators with systematically sampled networks. Eur J Oper Res 187:511–524

Yamashita D, Armentano V, Laguna M (2006) Scatter search for project scheduling with resource availability cost. Eur J Oper Res 169:623–637

Yamashita D, Armentano V, Laguna M (2007) Robust optimization models for project scheduling with resource availability cost. J Sched 10:67–76

Zhang X, Wang Y, Cui G, Niu Y, Xu J (2009) Application of a novel IWO to the design of encoding sequences for DNA computing. Comput Math Appl 57:2001–2008

Chapter 17
Exact Methods for Resource Leveling Problems

Julia Rieck and Jürgen Zimmermann

Abstract Resource leveling problems arise whenever it is expedient to reduce the fluctuations in resource utilization over time, while maintaining a prescribed project completion deadline. Several resource leveling objective functions may be defined, whose consideration results in resource profiles with desired properties, e.g., well-balanced resource profiles or profiles with a minimum number of jump discontinuities. In this chapter, we concentrate on three resource leveling problems that are known from the literature. In order to solve medium-scale instances of the considered problems, an enumeration scheme that uses problem structures is presented. Furthermore, mixed-integer (linear) programming models are introduced, and resource leveling instances are solved using CPLEX 12. In a comprehensive computational study, the performance of the described methods is analyzed.

Keywords Minimum and maximum time lags • Mixed-integer programming • Renewable resources • Resource leveling • Tree-based branch-and-bound method

17.1 Introduction

In order to face the challenges of decreasing product lifecycles, rapid advances in technologies, and tougher competitions in markets, many companies focus on an efficient usage of resources. To avoid the extremes of resource overloads or resource underloads, associated with an increase in costs, expensive renewable resources such as special machines, equipment, or highly qualified manpower have to be utilized in a "leveled" form. Scheduling problems in which interdependent activities are to be scheduled such that the resource utilization will be as smooth as possible or will contain as few as possible jump discontinuities over a medium-term planning horizon are called *resource leveling problems* (Demeulemeester and Herroelen 2002, Sect. 5).

J. Rieck (✉) • J. Zimmermann
Institute of Management and Economics, Clausthal University of Technology,
Clausthal-Zellerfeld, Germany
e-mail: julia.rieck@tu-clausthal.de; juergen.zimmermann@tu-clausthal.de

C. Schwindt, J. Zimmermann (eds.), *Handbook on Project Management and Scheduling Vol. 1*, International Handbooks on Information Systems,
DOI 10.1007/978-3-319-05443-8_17

Resource leveling problems (RLPs) are interesting from both practical and theoretical points of view. On the one hand, the levels of resource utilization are generally linked to quality of products, reject rates, and balanced material flows. On the other hand, resource leveling problems are $\mathcal{NP}$-hard in the strong sense and difficult to solve to optimality (Neumann et al. 2003, Sect. 3.4). Thus, many researchers have developed heuristic algorithms for RLPs in order to generate good solutions for practical relevant problem sizes (Ballestín et al. 2007; Raja and Kumanan 2007). However, RLPs also have nice structural properties, whose consideration can be used to find optimal solutions for medium-scale instances in moderate computation time.

A multitude of resource leveling functions may be defined, where the minimization will provide, for example, well-balanced resource profiles. In what follows, we investigate three special resource leveling problems that are known from the literature. In particular, we consider the "classical resource leveling problem", where variations in resource utilizations within the project duration are to be minimized (Burgess 1962). In addition, we study the "overload problem", where costs are incurred if either a given supply of some renewable resources or a threshold for the resource utilization is exceeded (Easa 1989). Finally, we examine the "total adjustment cost problem" that coincides with the cumulative costs arising from increasing or decreasing the utilizations of resources (Kreter et al. 2014).

The remainder of this chapter is organized as follows: In Sect. 17.2, we formally describe resource leveling problems using the three objective functions and present the mathematical background. Section 17.3 contains structural properties of RLPs. Section 17.4 is devoted to a literature review on exact solution methods for resource leveling, where we sketch the most common approaches. In Sect. 17.5 a tree-based branch-and-bound procedure is introduced, and in Sect. 17.6 known mathematical model formulations are presented. Based on those models, we proceed to describe methods for linearizing the corresponding objective functions and improving the quality of the resulting formulations in terms of computation time and solution gap. The results of a comprehensive performance analysis are given in Sect. 17.7 and finally, conclusions are presented in Sect. 17.8.

17.2 Problem Description

In what follows, we consider projects specified by activity-on-node networks $N = (V, E; \delta)$, where V is the set of vertices and E is the set of arcs with weights $\delta \in \mathbb{Z}$. Vertex set $V := V^r \cup V^m \cup \{0, n+1\}$ consists of real activities, milestones that specify significant events, and two fictitious activities, 0 and $n+1$, that represent the beginning and completion of the underlying project, respectively. Each activity i has a given processing time $p_i \in \mathbb{Z}_{\geq 0}$ and has to be carried out without interruption. For activities 0 and $n+1$, as well as for milestones, the processing time is set to zero.

We denote the start time of activity $i \in V$ by S_i. Temporal relationships between start times of activities are represented by minimum and maximum time lags. If activity j cannot be started earlier than $d_{ij}^{min} \in \mathbb{Z}_{\geq 0}$ time units after the start of activity i (minimum time lag), i.e. $S_j - S_i \geq d_{ij}^{min}$, we introduce an arc (i, j) having weight $\delta_{ij} := d_{ij}^{min}$ into network N. If activity j must be started no later than $d_{ij}^{max} \in \mathbb{Z}_{\geq 0}$ time units after activity i (maximum time lag), i.e. $S_j - S_i \leq d_{ij}^{max}$, we insert a backward arc (j, i) with weight $\delta_{ji} := -d_{ij}^{max}$. At a medium-term planning level, a deadline $\overline{d}$ for the termination of the project is usually given. In order to ensure that the prescribed deadline will not be exceeded, an arc $(n + 1, 0)$ having weight $\delta_{n+1,0} := -\overline{d}$ is further added to the network. Hence, the set of feasible start times of activity $i \in V$ forms a proper time window $[ES_i, LS_i]$, where ES_i is the earliest and LS_i the latest start time of activity i with respect to the given temporal constraints. By definition, $ES_0 = LS_0 := 0$. We assume that the underlying project is "well-defined", i.e., the conditions $S_i \geq 0$ and $S_i + p_i \leq S_{n+1}, i \in V$, will always be observed by the underlying minimum and maximum time lags.

Let $\mathscr{R}$ be the set of (discrete) renewable resources available at each point in time, independent of their former utilization. The amount of resource $k \in \mathscr{R}$ used constantly during execution of activity $i \in V$ (resource requirement) is represented by $r_{ik} \in \mathbb{Z}_{\geq 0}$. We suppose that the resource requirements of activities 0 and $n + 1$, as well as of milestones, are zero. In order to avoid that two activities i and j with $S_i + p_i = S_j$ compete for some renewable resource required, we assume that each real activity is executed during the half-open time interval $[S_i, S_i + p_i[$.

Given some schedule $S = (S_i)_{i \in V}$, the "active set" $\mathscr{A}(S, t)$ contains all real activities in progress at time t, i.e., $\mathscr{A}(S, t) := \{i \in V^r | S_i \leq t < S_i + p_i\}$. Thus, $r_k(S, t) := \sum_{i \in \mathscr{A}(S,t)} r_{ik}$ represents the total amount of resource $k \in \mathscr{R}$ required for those activities in progress at time t. The resulting resource profiles $r_k(S, \cdot) : [0, \overline{d}] \to \mathbb{R}_{\geq 0}$ are step functions continuous from the right at their jump points. In project scheduling, we distinguish between time-based, financial, and resource-based objectives. A resource leveling approach always considers some non-regular resource-based objective function that evaluates the resource utilization over time.

In practice, companies often want to realize smooth resource profiles in the course of the project, and aim at penalizing high resource utilizations more than low resource utilizations. Let $c_k \geq 0$ be the cost incurred per unit of resource k and per time unit. The "classical resource leveling objective function" represents the total squared utilization cost for a given schedule S and will be given by

$$f(S) := \sum_{k \in \mathscr{R}} c_k \int_{t \in [0, \overline{d}]} r_k^2(S, t)\, dt \tag{C-RL}$$

(Burgess 1962; Harris 1990). A possible application can be found in make-to-order manufacturing operations, where an even-workload distribution of resources is desirable (Ballestín et al. 2007).

Employers are usually required to pay overtime premiums to employees who work more than the standard hours. Additional costs for covering the positive deviations from the desired resource utilizations $Y_k, k \in \mathscr{R}$, will therefore be incurred (Bandelloni et al. 1994; Easa 1989). In order to take only the positive deviations into account, we consider the "total overload cost function"

$$f(S) := \sum_{k\in\mathscr{R}} c_k \int\limits_{t\in[0,\overline{d}]} (r_k(S,t) - Y_k)^+ dt \qquad \text{(O-RL)}$$

where $(z)^+ := \max(z, 0)$. In case no thresholds Y_k, e.g., the standard weekly hours, have been prescribed, Y_k may be chosen equal to the (rounded) average resource utilizations, i.e., $Y_k := \sum_{i\in V}(r_{ik}\, p_i/\overline{d}\,)$. Objective function (O-RL) may be modified by replacing $(\ldots)^+$ by $|\ldots|$ in order to take positive and negative deviations into account. Furthermore, if the condition $(\ldots)^+$ is substituted by $(\ldots)^2$, we obtain a problem that is equivalent to the classical resource leveling problem. Assuming that time t is discrete, the integrals appearing in (C-RL) and (O-RL) could be replaced by summations and the three-field notation for the classical RLP could be given by $PS\infty|temp, \overline{d}|\Sigma c_k \Sigma r_k^2$ and for the overload problem by $PS\infty|temp, \overline{d}|\Sigma c_k \Sigma o_{kt}$ with $o_{kt} := (r_k(S,t) - Y_k)^+$.

If resources represent different kinds of manpower, where changing the size of work force from period to period is associated with high costs, the "total adjustment cost function" is used to identify the cumulative costs arising from increasing or decreasing the resource utilizations. Given some schedule S, the set of points in time, at which at least one activity $i \in V$ is started or completed, is denoted by DT (decision times concerning schedule S). Let $\sigma_t := \max\{\tau \in DT | \tau < t\}$ be the decision time preceding some $t \in DT \setminus \{0\}$ and

$$\Delta r_{kt} := \begin{cases} r_k(S,t) - r_k(S,\sigma_t), & \text{if } 0 < t \leq \overline{d} \\ r_k(S,0) & \text{otherwise} \end{cases}$$

the jump difference in the resource profile of resource $k \in \mathscr{R}$ at time $t \in DT$. Moreover, $\Delta^+ r_{kt} := (\Delta r_{kt})^+$ and $\Delta^- r_{kt} := (-\Delta r_{kt})^+$ are the increase and decrease, respectively, in utilization of resource k at time t. The parameters $c_k^+ \geq 0$ and $c_k^- \geq 0$ denote the costs for positive and negative jumps in resource utilization of resource k by one unit. Since the condition $\sum_{t\in DT} \Delta^+ r_{kt} = \sum_{t\in DT} \Delta^- r_{kt}$ holds for all resources $k \in \mathscr{R}$, the "total adjustment cost function" may be formulated by

$$f(S) := \sum_{k\in\mathscr{R}} c_k \sum_{t\in DT\setminus\{\overline{d}\}} \Delta^+ r_{kt} \qquad \text{(A-RL)}$$

where $c_k := c_k^+ + c_k^-, k \in \mathscr{R}$. Hence, it is sufficient to consider only the positive jump differences in the resource profiles. In order to strengthen the resource leveling effect, $\Delta^+ r_{kt}$ can be squared in the objective function. The three-field notation for the total adjustment cost problem is typically given by $PS\infty|temp, \overline{d}|\Sigma c_k \Sigma \Delta r_{kt}$.

The problem of finding an optimal (time-feasible) schedule for a resource leveling problem with objective function (C-RL), (O-RL), or (A-RL) can now be formulated as follows:

$$\left.\begin{array}{ll} \text{Min. } f(S) & \\ \text{s.t. } S_j - S_i \geq \delta_{ij} & ((i,j) \in E) \\ S_0 = 0 & \end{array}\right\} \qquad \text{(RL)}$$

The feasible region $\mathscr{S}_T$ of the described problem is non-empty, iff network N contains no cycle of positive length (Bartusch et al. 1988). As shown in Neumann et al. (2003) by a polynomial reduction to three-partition, problem (RL) is $\mathscr{NP}$-hard in the strong sense for all three resource leveling variants presented, even if only one single resource is considered (i.e., $|\mathscr{R}| = 1$).

17.3 Structural Properties of Resource Leveling Problems

Structural properties can be exploited in order to restrict the search for an optimal solution to the finite subset of so called “quasistable” schedules. In what follows, some basic definitions and concepts are introduced to facilitate understanding of exact procedures from literature presented further on (Neumann et al. 2000).

A “strict order” O is an asymmetric and transitive relation in (real) activity set V^r that establish time-oriented precedences among activities. Moreover, set $\mathscr{S}_T(O) := \{S \in \mathscr{S}_T \mid S_j \geq S_i + p_i$ for all $(i,j) \in O\}$ of all time-feasible schedules satisfying the precedence constraints induced by strict order O is termed the “order polytope” of O. Network $N(O)$ is called the “order network” of strict order O if it contains a precedence arc (i,j) with weight p_i for each activity pair $(i,j) \in O$.

Resource leveling objective functions have some specific characteristics, for example, objective functions (C-RL) and (O-RL) are continuous and concave on each set of schedules representing the same strict order O [i.e., (C-RL) and (O-RL) belong to the class of “locally concave” functions]. In contrast, objective function (A-RL) is not necessarily continuous, but it can be noticed that it is “lower semicontinuous”, as for all schedules $S, S' \in \mathbb{R}^{n+2}_{\geq 0}$ the condition $f(S) \leq \liminf_{S' \to S} f(S')$ holds. Moreover, function (A-RL) is also “quasiconcave” on each set of schedules representing the same strict order O [i.e., (A-RL) belongs to the class of “locally quasiconcave” functions].

Consider the resource profile in Fig. 17.1. We assume that activity 2 is delayed from its earliest start time, $ES_2 = 0$, to its latest start time, $LS_2 = 6$. The resulting course of the objective functions (C-RL), (C-RL), and (A-RL) are depicted in the three coordinate systems in Fig. 17.1. Functions (C-RL) and (O-RL) are continuous and concave on the intervals $[0, 1]$ (strict order $O = \{(2, 3)\}$), $]1, 4[$

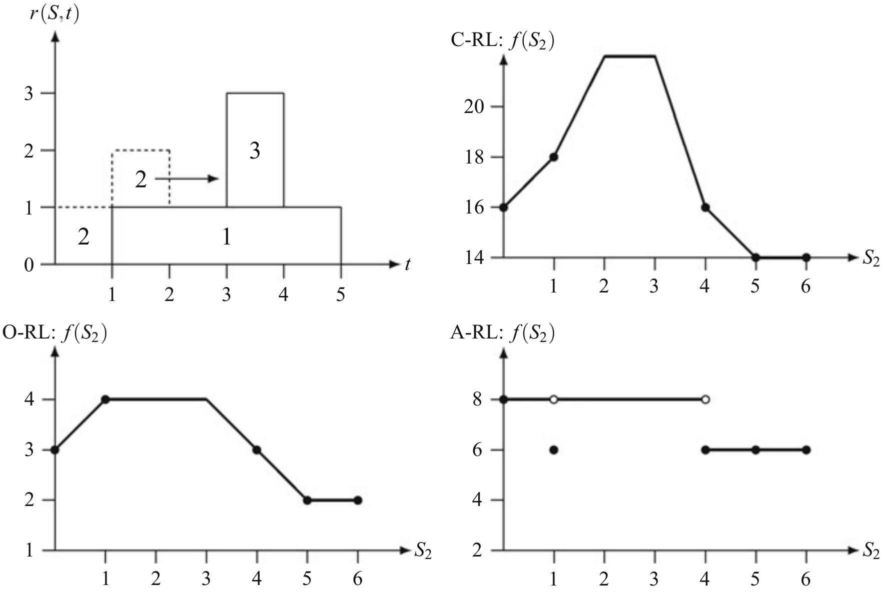


Fig. 17.1 Continuous and lower semicontinuous functions

(strict order $O = \emptyset$), $[4,5[$ (strict order $O = \{(3,2)\}$), and $[5,6]$ (strict order $O = \{(1,2),(3,2)\}$). Function (A-RL) is lower semicontinuous and has jump discontinuities at times $t = 1$ and $t = 4$. Furthermore, (A-RL) is quasiconcave on the intervals $[0,1]$, $]1,4[$, $[4,5[$, and $[5,6]$.

For a locally (quasi-)concave function f, there invariably exists a "quasistable" schedule which is optimal for the three considered resource leveling problems (Neumann et al. 2000, 2003).

Remark 17.1. A feasible schedule S is termed *quasistable* if there is no pair of opposite order-preserving shifts.

Let S be a quasistable schedule. For each activity $i \in V$, there is either an activity $j \in V$ such that $S_i + p_i = S_j$ or $S_i + \delta_{ij} = S_j$, or an activity $h \in V$ such that $S_i = S_h + p_h$ or $S_i = S_h + \delta_{hi}$. The set of quasistable schedules covers exactly the set of extreme points of all order polytopes $\mathscr{S}_T(O)$. Consequently, there must always be an extreme point S^* of some order polytope $\mathscr{S}_T(O)$ which is a minimizer of problem (RL) on $\mathscr{S}_T \neq \emptyset$. Moreover, each extreme point can be represented by a spanning tree of the corresponding order network $N(O)$, where every arc represents a binding temporal or precedence constraint (Neumann et al. 2000).

Finally, objective functions (C-RL) and (O-RL) are r-monotone, i.e., the condition $f(S) \leq f(S')$ is implicitly specified for (partial) schedules S and S' with $r_k(S,t) \leq r_k(S',t)$ for all $k \in \mathscr{R}$ and $t \in [0,\overline{d}]$ (Zimmermann 2001). This property can be used to generate good and efficient lower bounds for all schedules generated in the course of an enumeration scheme. Objective function (A-RL) is not r-monotone. Therefore, the determination of lower bounds is more difficult and requires the estimation of jump differences in resource profiles with respect to scheduled and unscheduled activities. Consequently, the values of lower bounds, e.g., the values of linear programming relaxations used in solvers like CPLEX, are relatively weak.

17.4 Literature Review

Exact solution procedures for resource leveling problems presented in the literature may be separated into three different categories:

1. Enumeration schemes that enumerate the feasible integral start times of project activities.
2. Enumeration schemes that enumerate the spanning trees (extreme points) of all feasible order networks (order polytopes) of the underlying project.
3. Mixed-integer programming (MIP) models that consider binary variables for the activities involved.

Approaches in the first and the third category mostly require a discretization of the time horizon. Since the components $S_i^*, i \in V$, of at least one optimal schedule S^* are integers (assuming that $p_i, \delta_{ij} \in \mathbb{Z}_{\geq 0}, i, j \in V$, cf. Sect. 17.2), we are able to restrict the possible start times of activity i to a set of discrete times, $W_i := \{ES_i, \ldots, LS_i\}$.

Petrovic (1969) introduced the first exact solution approach for resource leveling problems with precedence constraints that can be classified into category one. The method is based on dynamic programming and aims at minimizing objective function (C-RL). After that, Ahuja (1976, Sect. 11) presented a procedure that enumerates every combination of feasible start times in order to minimize a quadratic variant of objective function (A-RL). Younis and Saad (1996) proposed a method based on integer programming in order to treat the sum of absolute deviations of the resource utilization from a desired resource profile. Moreover, Bandelloni et al. (1994) devised a dynamic programming approach for the overload problem based on integral floats of activities. For problems with general temporal constraints, Neumann and Zimmermann (2000) proposed a time-window based branch-and-bound procedure that may be used for all objective functions mentioned in Sect. 17.2. The method continues the concepts of the aforementioned approaches.

Gather et al. (2011) considered a tree-based enumeration scheme (second category) for resource leveling problems with general temporal constraints, where different techniques for avoiding redundancies are employed. The enumeration scheme can be adopted for problems with objective functions (C-RL) or (O-RL), respectively. During the course of the algorithm structural properties are used in an efficient way in order to speed up the solution process (cf. Sect. 17.5).

Mixed-integer programming models for project scheduling problems are inspired by the work of Pritsker et al. (1969). There, a discrete time-based formulation is proposed, where binary variables x_{it} are introduced that allocate a feasible start time $t \in W_i$ to each activity $i \in V$. In the case of the overload problem with precedence constraints, a float-based model has been presented by Easa (1989). Selle (2002) introduced a discrete time-based formulation for the total adjustment cost problem. Several MIP-formulations for problems with objective functions (C-RL) and (O-RL) are presented in Rieck et al. (2012). Applying preprocessing and linearization techniques, the authors solved instances with up to 50 real activities and varying project completion deadlines to optimality using CPLEX 12. Finally, Kreter et al. (2014) developed MIP-formulations for the total adjustment cost problem that focus on the fact that a positive jump in the resource profiles occurs only if one or more real activities are started. The algorithms of Rieck et al. (2012) as well as Kreter et al. (2014) are currently the best exact approaches to tackle RLPs (cf. Sect. 17.6). The methods outperform the procedures presented by Neumann and Zimmermann (2000) as well as Gather et al. (2011) in terms of computation time and solution gap.

In order to solve RLPs approximately, many authors developed heuristic approaches, where most of them may be used for all objective functions mentioned above. Simple shifting and priority-rule methods for problems with precedence constraints can be found in Burgess (1962), Wiest (1963), Galbreath (1965), Ahuja (1976), as well as in Harris (1978, 1990). In addition, several metaheuristics either based on local search or on nature based algorithms are introduced by Takamoto et al. (1995), Savin et al. (1997), Raja and Kumanan (2007), Christodoulou et al. (2010), and Geng et al. (2011). The last two articles are relatively new, but the authors consider only small problem instances with up to eleven real activities. Furthermore, Ranjbar (2013) describes a path-relinking metaheuristic for projects with precedence constraints that is able to solve instances with 500 activities in acceptable time.

Neumann and Zimmermann (1999, 2000) proposed heuristic approaches being suitable for all aforementioned resource leveling problems as well as projects with general temporal constraints. In Neumann et al. (2003) neighborhoods, which allow schedule-improvement procedures to reach optimal solutions independently of the initial schedule chosen, are presented. Moreover, Ballestín et al. (2007) published a population-based method of the integrated greedy-type that generates the best known results for large instances containing minimum and maximum time lags with up to 1,000 activities so far.

17.5 Tree-Based Branch-and-Bound Method

The tree-based branch-and-bound procedure presented by Gather et al. (2011) is currently the best branch-and-bound method presented in the literature. Based on a directed multigraph $N(\hat{O})$ that contains arcs for every time lag and for every feasible strict order, the extreme points (quasistable schedules) of all order polytopes $\mathscr{S}_T(O)$ are enumerated by generating the corresponding spanning trees. During the enumeration, only time-feasible trees are constructed and the generation of different spanning trees leading to one and the same schedule is avoided by using the T-minimality concept of Nübel (1999). Additionally, an enhanced variation on the bridge-concept devised by Gabow and Myers (1978) is employed, which has the advantage that no partial tree, except for the current one, must be stored.

Let $\mathscr{C}$ be the completed set, i.e., the set of all activities which have already been scheduled, and let $S^{\mathscr{C}}$ be a partial schedule, where $S_i^{\mathscr{C}} \geq 0$ is satisfied for every activity $i \in \mathscr{C} \subseteq V$ and $S_0^{\mathscr{C}} = 0$. The tree-based enumeration is initiated using $\mathscr{C} = \{0\}$ and $S_0 = 0$. In each iteration, a pair $(\mathscr{C}, S^{\mathscr{C}})$ (i.e., a subtree) is removed from a set Ω. If $\mathscr{C} = V$, an extreme point of some order polytope has been found; otherwise, the current partial schedule $S^{\mathscr{C}}$ is extended as follows: For every $j \in V \setminus \mathscr{C}$, a set $\mathscr{D}_j$ of tentative start times, $t \in [ES_j(S^{\mathscr{C}}), LS_j(S^{\mathscr{C}})]$, for which there is an activity $i \in \mathscr{C}$ such that

(i) $t = S_i + \delta_{ij}$, i.e., temporal constraint $S_j - S_i \geq \delta_{ij}$ is binding, or

(ii) $t = S_i - \delta_{ji}$, i.e., temporal constraint $S_i - S_j \geq \delta_{ji}$ is binding, or

(iii) $t = S_i + p_i$, i.e., precedence constraint $S_j - S_i \geq p_i$ is binding, or

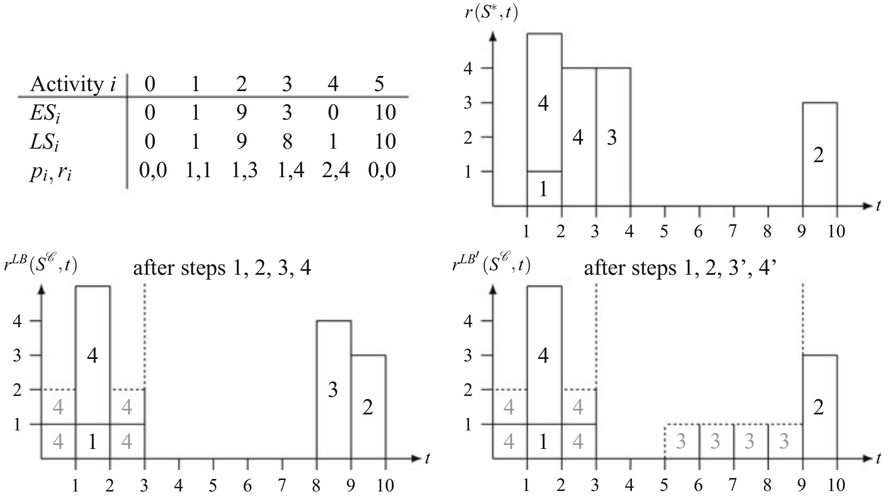
(iv) $t = S_i - p_j$, i.e., precedence constraint $S_i - S_j \geq p_j$ is binding,

is determined. Then for every $t \in \mathscr{D}_j$, the corresponding extended partial schedule $S^{\mathscr{C}'}$, where $\mathscr{C}' = \mathscr{C} \cup \{j\}$ and $S_j = t$, is added to set Ω. In order to improve the efficiency of the procedure, the enumeration scheme is embedded into a branch-and-bound approach.

The whole procedure was mainly constructed to solve the classical RLP. Therefore, the branch-and-bound elements (i.e., upper and lower bounds) must be adopted for other resource leveling variants. An upper bound on the objective function value allow for evaluating the quality of new solutions or even partial solutions. Upper bounds for the classical resource leveling and the overload problem can be calculated by using the Ballestín et al. (2007) heuristic. Furthermore, an enhanced version of the Ballestín et al. (2007) heuristic, that is suitable for the total adjustment cost problem, may be found in Kreter et al. (2014). The procedure ensures that no neighboring quasistable schedule is skipped.

A lower bound helps to avoid further examination of (partial) schedules that can be recognized as non-optimal. During the enumeration, Gather et al. (2011) use a constructive lower bound, *LB*, which is an extension of the bound devised by Engelhardt and Zimmermann (1998). Lower bound *LB* can be used for objective functions (C-RL) and (O-RL) and the calculation is based on the resource profiles for partial schedules. The following four steps have to be executed:

1. Initialize a resource profile for each resource $k \in \mathscr{R}$ by considering the resource requirements r_{ik} of all activities $i \in V^{\text{fix}} := \mathscr{C}$ having already a fixed start time S_i.



Activity i	0	1	2	3	4	5
ES_i	0	1	9	3	0	10
LS_i	0	1	9	8	1	10
p_i, r_i	0,0	1,1	1,3	1,4	2,4	0,0

Fig. 17.2 Calculation of lower bounds LB and LB'

2. Add resource requirements of all critical activities j with $ES_j = LS_j$ and update V^{fix}. Add resource requirements of all near-critical activities j with $ES_j + p_j > LS_j$ within the interval $[LS_j, EC_j[$.
3. Insert the resource requirements of a maximum set of non-scheduled activities j with disjoint time windows $[ES_j, LC_j[$ in the resource profiles as best as possible. Update V^{fix}.
4. Finally, differentiate all remaining activity resource requirements in a set of independent blocks of 1-resource-unit-for-1-time-unit (1-blocks). These 1-blocks are distributed as best as possible within the time interval $[\min\{ES_m | m \in V \setminus V^{\text{fix}}\}, \max\{LC_n | n \in V \setminus V^{\text{fix}}\}[$.

In order to illustrate the determination of lower bounds, we consider a project with four real activities and one renewable resource. The earliest and latest start times as well as the durations and resource requirements of activities are given in the upper part of Fig. 17.2. Moreover, the resource profile of an optimal solution S^* for problem (RL) with objective functions (C-RL) or (A-RL) is depicted. To calculate LB, we assume that only activity 0 is scheduled, i.e., $\mathscr{C} = \{0\}$. Therefore, the resource profile remains empty after step 1. In step 2, the resource requirements of critical activities 1 and 2 as well as of near-critical activity 4 are added. Since the remaining activity 3 does not overlap with any other activity, the resource requirements are inserted as best as possible. In the fourth step, 1-blocks of activity 4 are distributed within the time interval $[0, 3[$. The resulting resource profile $r^{LB}(S^{\mathscr{C}}, \cdot)$ is given in the lower part of Fig. 17.2.

Unfortunately, lower bound LB is not valid for the total adjustment cost problem, since A-RL: $f(r^{LB}(S^{\mathscr{C}}, t)) = 9 >$ A-RL: $f(S^*) = 8$, assuming that cost component c is equal to 1. For function (A-RL), which is not r-monotone, a new lower bound LB' must be determined in which the third and the fourth step are modified. In doing so, time intervals are determined, where activities $i \in V \setminus V^{\text{fix}}$ may be in execution due to their current time windows (step 3'). Within these intervals, 1-blocks of the corresponding activities are distributed such that as few as possible resource jumps occur (step 4').

Considering the example described above. In step 3', the intervals $[3, 9[$ (execution interval of activity 3) and $[0, 3[$ (execution interval of activity 4) are determined and in step 4' the 1-blocks of activities 3 and 4 are added. The resulting resource profile $r^{LB'}(S^{\mathscr{C}}, \cdot)$ is given on the right hand side of Fig. 17.2 and A-RL: $f(r^{LB'}(S^{\mathscr{C}}, t)) = 8 \leq$ A-RL: $f(S^*) = 8$ holds.

17.6 Mixed-Integer Programming Models

In this section, we present different MIP-models for resource leveling problems that are successful in generating optimal solutions in considerable time (Kreter et al. 2014; Rieck et al. 2012). Firstly, discrete time-based formulations are introduced in Sect. 17.6.1. Then, Sect. 17.6.2 is devoted to linear models for the classical resource leveling and the overload problem. Moreover, linear formulations for the total adjustment cost problem are presented in Sect. 17.6.3.

17.6.1 Discrete Time-Based Formulations

Most common model formulations for project scheduling problems are based on a time discretization, where the time axis is divided into equidistant subintervals. A corresponding discrete time-based model applies binary variables x_{it} that allocate a feasible start time $t \in W_i := \{ES_i, \ldots, LS_i\}$ to each activity $i \in V$, i.e.,

$$x_{it} := \begin{cases} 1, & \text{if activity } i \text{ starts at time } t \\ 0, & \text{otherwise} \end{cases} \tag{17.1}$$

The problem of finding an optimal schedule for an objective function f may then be formulated as follows:

$$\text{Min.} \quad f(x)$$

$$\text{s.t.} \quad \sum_{t \in W_i} x_{it} = 1 \qquad (i \in V) \tag{17.2}$$

$$\sum_{t \in W_j} t\, x_{jt} - \sum_{t \in W_i} t\, x_{it} \geq \delta_{ij} \qquad ((i, j) \in E) \tag{17.3}$$

$$x_{00} = 1 \tag{17.4}$$

$$x_{it} \in \{0, 1\} \qquad (i \in V, t \in W_i) \tag{17.5}$$

Constraints (17.2) ensure that each activity receives exactly one start time. Since $S_i = \sum_{t \in W_i} t\, x_{it}$ holds for all activities $i \in V$, Inequalities (17.3) guarantee that the temporal constraints given by minimum and maximum time lags will be satisfied. Finally, Condition (17.4) sets the start time for the project to zero.

In order to present a time-indexed formulation for objective functions (C-RL) and (O-RL), continuous auxiliary variables $z_{kt} \geq 0$, which indicate the total resource requirements for resource $k \in \mathscr{R}$ and time $t \in T := \{1, \ldots, \overline{d} - 1\}$, are introduced. A (real) activity i is in progress and requires resources at some time t if $S_i \in \{t - p_i + 1, \ldots, t\}$. Then, inequalities

$$z_{kt} \geq \sum_{i \in V^r} r_{ik} \sum_{\tau = \max\{ES_i, t - p_i + 1\}}^{\min\{t, LS_i\}} x_{i\tau} \qquad (k \in \mathscr{R}, t \in T) \tag{17.6}$$

estimate the resource requirements of all activities for resource k and time t and the objective functions may be specified by

$$f(x) := \sum_{k \in \mathscr{R}} c_k \sum_{t \in T} z_{kt}^2 \tag{C-RL-a}$$

$$f(x) := \sum_{k \in \mathscr{R}} c_k \sum_{t \in T} (z_{kt} - Y_k)^+ \tag{O-RL-a}$$

For the total adjustment cost objective function, continuous auxiliary variables $\Delta_{kt}^+ \geq 0$, which determine the positive jump difference in the resource profile of resource k and time t, are introduced, i.e.,

$$\Delta_{kt}^+ \geq \sum_{i \in V^r} r_{ik} \sum_{\tau = \max\{ES_i, t - p_i + 1\}}^{\min\{t, LS_i\}} x_{i\tau} - \sum_{i \in V^r} r_{ik} \sum_{\tau = \max\{ES_i, t - p_i\}}^{\min\{t - 1, LS_i\}} x_{i\tau} \quad (k \in \mathscr{R}, t \in T) \tag{17.7}$$

Consequently, the objective function has the form

$$f(x) := \sum_{k \in \mathscr{R}} c_k \sum_{t \in T} \Delta_{kt}^+ \tag{A-RL-a}$$

All formulations incorporate $\mathscr{O}(|V|^2 + |\mathscr{R}||T|)$ constraints, as well as $|\mathscr{R}||T|$ real-valued auxiliary and $\sum_{i \in V} |W_i|$ binary variables. Since all numbers depend on the time discretization, the models cannot be designated as polynomial models.

Easa (1989) proposed a float-based formulation for the overload problem that has some similarities to the discrete time-based formulation. The model uses binary variables $x_{iq} \in \{0, 1\}$ that state the extent of shifting, $q \in \{1, \ldots, TF_i\}$, of activity $i \in V$ beyond its earliest start time. Rieck et al. (2012) extended the formulation in order to deal with general temporal constraints and a set of renewable resources. However, the resulting model turned out to be less efficient in numerical tests.

Koné (2011) used the start/end event-based methodology in order to solve resource-constrained project scheduling problems with precedence constraints. For those problems, there always exists an optimal solution in which the start time of an activity is either 0 or coincides with the end time of some other activity. Therefore, the number of possible events can be defined as $E' := \min\{|V^r| + 2 + |V^m|, \overline{d}\}$ which is significantly lower than the number $E := \min\{2|V^r| + 2 + |V^m|, \overline{d}\}$ of events needed for problems with general temporal constraints. In addition, RLPs are characterized by many feasible schedules with the same objective function value. Therefore, the events may be positioned at various analogous points in time, which reduces the performance of a solver. Kreter et al. (2014) considered the formulation for the total adjustment cost problem, but it is found to be of little value as an alternative method.

Note that flow-based continuous-time and on/off event-based formulations are also available for resource-constrained project scheduling problems (Koné 2011). However, both formulations cannot be extended to a resource leveling problem as they are not able to describe total resource requirements at specific points in time, which are needed for determining the objective function values.

MIP-models with linear constraints and a linear objective function are proved to be a key factor for obtaining exact solutions to combinatorial optimization problems in reasonable time. However, objective functions (C-RL-a) and (O-RL-a) are still nonlinear. Therefore, preprocessing and linearization techniques are needed in order to reduce, or simplify, these optimization problems. The methods presented in Sect. 17.6.2 lead to the currently best-known exact approaches for RLPs.

17.6.2 Linear Formulations for the Classical Resource Leveling and the Overload Problem

Model formulations (C-RL-a), (17.2)–(17.6) and (O-RL-a), (17.2)–(17.6) use continuous auxiliary variables $z_{kt} \geq 0$ for resource $k \in \mathscr{R}$ at time $t \in T$. The domains of those variables may be restricted by considering upper resource requirements bounds. We identify $H_{kt} \geq 0$ with the maximum requirements for resource k that can occur at time t, for the case of any feasible schedule. The value H_{kt} can be computed by regarding the interval $[ES_i, LC_i[$ as execution interval for every real activity $i \in V^r$. Since $[S_i, S_i + p_i[\subseteq [ES_i, LC_i[$ holds for any feasible start time of activity i, the corresponding resource profiles $\hat{r}_k(\cdot)$ provide upper bounds on the resource requirements for resource k at time t. In order to further determine time-orientated precedences among project activities, the concept of so-called antichains may be used (Möhring 1984; Schwindt 2005). An antichain U is a set of activities, such that any two distinct activities, i and j, may overlap under a feasible schedule. For each antichain and each resource, a weight can be calculated. A *maximum-weight antichain* is thus an antichain that produces the maximum weight among all possible antichains. Weight

$$H_{kt} := \sum_{i \in U_{kt}^{max}} r_{ik}$$

of a maximum-weight antichain $U_{kt}^{max} \subseteq V$ at jump point $t \in T$ in resource profile $\hat{r}_k(\cdot)$ equals the total number of units of resource k required for project execution at time t in the most unfavorable case. H_{kt} may be efficiently determined by computing a minimum origin-destination flow on a specific constructed flow network (Rieck et al. 2012).

Until now, objective function (C-RL-a) is based on auxiliary variables

$$0 \leq z_{kt} \leq H_{kt} \qquad (k \in \mathscr{R}, t \in T) \tag{17.8}$$

In order to linearize (C-RL-a), we divide the interval of resource requirements, $[0, \ldots, H_{kt}]$, into H_{kt} equal parts, $[0,1], [1,2], \ldots, [H_{kt} - 1, H_{kt}]$. In addition, a continuous auxiliary variable, y_{kth}, is introduced for each part, where

$$0 \leq y_{kth} \leq 1 \qquad (k \in \mathscr{R}, t \in T, h \in \{1, \ldots, H_{kt}\}) \tag{17.9}$$

The auxiliary variables z_{kt} in Constraints (17.6), and the auxiliary variables y_{kth} may then be linked using the relation

$$z_{kt} = \sum_{h=1}^{H_{kt}} y_{kth} \qquad (k \in \mathscr{R}, t \in T) \tag{17.10}$$

The assumption that the resource requirements r_{ik} are non-negative, along with the fact that the objective function is convex will force the variables y_{kth} to take on values of either 0 or 1, which is why we merely need to demand that the conditions $y_{kth} \in [0,1]$, rather than the conditions $y_{kth} \in \{0,1\}$, are satisfied.

Moreover, we are readily able to compute a direction factor, $2h - 1$, for every variable y_{kth}, which equals the difference between h^2 and $(h-1)^2$. Combined with auxiliary variables $y_{kth} \in [0,1]$, we obtain an exact approximation for objective function (C-RL-a) that depends on $z_{kt} \in \mathbb{N}_0$. The stepwise-linear objective function may now be written as

$$f(x) := \sum_{k \in \mathscr{R}} c_k \sum_{t \in T} \sum_{h=1}^{H_{kt}} (2h-1)\; y_{kth} \tag{C-RL-b}$$

Another possibility to linearize objective function (C-RL-a) involves considering additional binary variables $g_{kth} \in \{0, 1\}$ designating that $h \in \{0, \dots, H_{kt}\}$ units of resource $k \in \mathscr{R}$ are required at time $t \in T$ (Gather et al. 2010). Since the resulting model turns out to be less efficient in numerical test (Rieck et al. 2012), we skipped pursuing further details.

The total overload cost function (O-RL-a) contains the term $(z_{kt} - Y_k)^+$ which may be replaced by $\max\{z_{kt} - Y_k, 0\}$. In order to linearize the resulting max-function, continuous auxiliary variables

$$0 \leq v_{kt} \leq H_{kt} - Y_k \qquad (k \in \mathscr{R}, t \in T) \tag{17.11}$$

are introduced that represent the positive deviations of total resource requirements from the desired resource rate Y_k. Then, the overload function could be replaced by linear function

$$f(x) := \sum_{k \in \mathscr{R}} c_k \sum_{t \in T} v_{kt} \tag{O-RL-b}$$

and Inequalities (17.6) must be substituted by

$$v_{kt} \geq \sum_{i \in V} r_{ik} \sum_{\tau = \max\{ES_i, t - p_i + 1\}}^{\min\{t, LS_i\}} x_{i\tau} - Y_k \qquad (k \in \mathscr{R}, t \in T) \tag{17.12}$$

Using the described linearization techniques, we obtain model

$$\mathrm{M}_1^{\text{C-RL}} : \text{(C-RL-b), (17.2)–(17.6), (17.8), (17.9), (17.10)}$$

for the classical resource leveling problem and model

$$\mathrm{M}_2^{\text{O-RL}} : \text{(O-RL-b), (17.2)–(17.5), (17.11), (17.12)}$$

for the overload problem. In our performance analysis, we will compare the results of both models to the results obtained by the tree-based branch-and-bound method.

17.6.3 Linear Formulations for the Total Adjustment Cost Problem

The discrete time-based model (A-RL-a), (17.2)–(17.5), (17.7) specified in Sect. 17.6.1 is already linear and uses continuous auxiliary variables $\Delta_{kt}^{+} \geq 0$ for resource $k \in \mathscr{R}$ at time $t \in T$. In order to restrict the domains to those variables, we introduce lower resource requirements bounds. Let $P_{kt} \geq 0$ be the minimum requirements for resource k that can occur at time t, for the case of any feasible schedule. P_{kt} may be obtained by considering the unavoidable time interval,

$[LS_i, EC_i[$, for every real activity $i \in V^r$. With bounds P_{kt} and H_{kt}, we are now able to estimate the maximum positive jump

$$\mathrm{M}_{kt} := H_{kt} - P_{k,t-1}$$

in the resource profile of resource k at time t, where $P_{k,-1} := 0$ for all k. Moreover,

$$\mathrm{M}_k := \max_{t \in T}\{H_{kt} - P_{k,t-1}\}$$

represents the maximum positive jump that may occur for resource k throughout the planning horizon. Using this information, the domains of continuous variables may be defined as

$$0 \leq \Delta^+_{kt} \leq \mathrm{M}_{kt} \qquad (k \in \mathscr{R}, t \in T) \tag{17.13}$$

In contrast to objective functions (C-RL) and (O-RL), function (A-RL) is not r-monotone and the lower bounds determined by solving the linear programming relaxation in MIP-solvers are relatively weak. Thus, a total adjustment cost problem seems to be harder to solve than the two other RLPs and the consideration of smart models, particularly polynomial models, is efficient. For that reason, we consider a start-based formulation that focuses on the fact that a positive jump in the resource profiles occurs, only if one or more real activities are started. For all activities $i, j \in V^r, i < j$, we make use of binary variables g_{ij} indicating whether real activities i and j start at the same time, i.e.,

$$g_{ij} := \begin{cases} 1 \text{ or } 0, & \text{if the start of activity } i \text{ is equal to the start of activity } j \\ 0, & \text{otherwise} \end{cases}$$

Additionally, we consider binary variables e_{ij} specifying if the start of real activity i and the completion of real activity j, $i \neq j$, occurs at the same point in time, i.e.,

$$e_{ij} := \begin{cases} 1 \text{ or } 0, & \text{if the start of activity } i \text{ is equal to the completion of activity } j \\ 0, & \text{otherwise} \end{cases}$$

Both types of variables have some "degree of freedom" when more than two activities start or end at the same time. For three activities with $S_i = S_j = S_h$, $i < j < h$, the variables g_{ij}, g_{ih}, and g_{jh} could be equal to one by definition. However, the resource requirements r_{ik}, r_{jk} and r_{hk} must be considered once in order to determine the correct jump difference in the resource profiles. That is always guaranteed in our model by setting $g_{ij} = g_{ih} := 1$ and $g_{jh} := 0$.

If only one real activity $i \in V^r$ is started at a specific point in time, then continuous auxiliary variables $0 \leq \Delta^+_{ki} \leq \mathrm{M}_k$ indicate the positive adjustment of resource $k \in \mathscr{R}$ at the start time of activity i. In case that a set of real activities, e.g., activities i, j, and h, is started at the same time, $(\Delta^+_{ki} + \Delta^+_{kj} + \Delta^+_{kh}) \geq 0$ represents the positive jump in the resource profile of resource k. Using two “big-M-values” for each activity pair $\{i, j\}$, and one “big-M-value” for each renewable resource k, the start-based formulation for the total adjustment cost problem takes the form (Kreter et al. 2014):

$$\text{Min.} \quad \sum_{k\in\mathscr{R}} c_k \sum_{i\in V^r} \Delta^+_{ki} \qquad \text{(A-RL-b)}$$

s.t.

$$S_j - S_i \geq \delta_{ij} \qquad ((i,j) \in E) \qquad (17.14)$$

$$S_i - (S_j + p_j) \leq \mathrm{M}_{ij}(1 - e_{ij}) \qquad (i, j \in V^r, i \neq j) \qquad (17.15)$$

$$S_j + p_j - S_i \leq \mathrm{M}_{ij}(1 - e_{ij}) \qquad (i, j \in V^r, i \neq j) \qquad (17.16)$$

$$\sum_{\substack{i\in V^r \\ i\neq j}} e_{ij} \leq 1 \qquad (j \in V^r) \qquad (17.17)$$

$$S_i - S_j \leq \mathrm{M}'_{ij}(1 - g_{ij}) \qquad (i, j \in V^r, i < j) \qquad (17.18)$$

$$S_j - S_i \leq \mathrm{M}'_{ij}(1 - g_{ij}) \qquad (i, j \in V^r, i < j) \qquad (17.19)$$

$$\sum_{\substack{i\in V^r \\ i<j}} g_{ij} \leq 1 \qquad (j \in V^r) \qquad (17.20)$$

$$\sum_{\substack{k\in V^r \\ j<k}} g_{jk} \leq (1 - g_{ij}) \qquad (i, j \in V^r, i < j) \qquad (17.21)$$

$$\Delta^+_{ki} \geq r_{ik} + \sum_{\substack{j\in V^r \\ i<j}} r_{jk} g_{ij} - \sum_{\substack{j\in V^r \\ j\neq i}} r_{jk} e_{ij} - \sum_{\substack{j\in V^r \\ j<i}} g_{ji} \mathrm{M}_k \qquad (i \in V^r, k \in \mathscr{R}) \qquad (17.22)$$

$$0 \leq \Delta^+_{ki} \leq \mathrm{M}_k \qquad (i \in V^r, k \in \mathscr{R}) \qquad (17.23)$$

$$S_i \in [ES_i, LS_i] \qquad (i \in V) \qquad (17.24)$$

$$g_{ij} \in \{0, 1\} \qquad (i, j \in V^r, i < j) \qquad (17.25)$$

$$e_{ij} \in \{0, 1\} \qquad (i, j \in V^r, i \neq j) \qquad (17.26)$$

Inequalities (17.14) correspond to the given minimum and maximum time lags between activities. If the start time of activity i is not equal to the completion time of activity j, decision variable e_{ij} receives the value zero, otherwise the value might be equal to one [cf. Constraints (17.15) and (17.16)]. With Constraints (17.17), it is guaranteed that every completing activity j is assigned to at most one starting activity, even if $C_j = S_i$ holds for more than one activity i. Since the strongest LP-relaxation will result from choosing the smallest possible “big-M-value”, we set $\mathrm{M}_{ij} := \max\{LS_i - (ES_j + p_j), LS_j + p_j - ES_i\}$. Inequalities (17.18) and (17.19) specify the values of decision variables g_{ij} in an analogous manner, where

$\mathrm{M}'_{ij} := \max\{LS_i - ES_j, LS_j - ES_i\}$. In the event that $S_i \neq S_j$ holds for activities i and j, variables g_{ij} will be set to zero, otherwise the values might be equal to one. Constraints (17.20) ensure that starting activity j is assigned to at most one starting activity i. If $g_{ij} = 1$, Constraints (17.21) guarantee that the start of none other activity $k > j$ can be assigned to starting activity j, i.e., only non-transitive pairs of decision variables can receive the value one and thus each resource requirement is considered once within the calculation of Δ^+_{ki}. Inequalities (17.22), together with objective function (A-RL-b), make sure that the significant decision variables are set to one and the positive adjustment of resource k at the start time of real activity i is estimated correctly. In order to ensure that no resource jump is counted more than once, the big-M_k-value reduces the irrelevant decision variables $\Delta^+_{ki} \geq 0$ to zero.

The resulting model contains $\mathscr{O}(|V|^2 + |V|\,|\mathscr{R}|)$ constraints, as well as $|V| + |V^r||\mathscr{R}|$ real-valued auxiliary and $\frac{3}{2}|V^r|^2 - \frac{3}{2}|V^r|$ binary variables. Since all numbers are independent of the scaling of the time axis, the model can be classified as a *polynomial* model.

To sum up, two efficient linear models for the total adjustment problem are specified, namely

$$\mathrm{M}_3^{\text{A-RL}} : \text{(A-RL-a), (17.2)–(17.5), (17.7), (17.13) and}$$

$$\mathrm{M}_4^{\text{A-RL}} : \text{(A-RL-b), (17.14)–(17.26)}$$

In our performance analysis, the results of both formulations will be compared to the results of the tree-based branch-and-bound method.

17.7 Computational Results

This section covers the results of computations undertaken in order to investigate the performance of the presented exact solution methods for resource leveling problems. We begin by describing the composition and generation of problem instances considered for testing the approaches (cf. Sect. 17.7.1). In an experimental performance analysis (cf. Sect. 17.7.2), we used the various MIP-formulations and CPLEX to solve medium-scale problem instances to optimality. The run times of the branch-and-cut procedures provided by CPLEX are compared to those for corresponding tree-based branch-and-bound methods.

17.7.1 Benchmark Instances

The computational tests have been performed on problem instances generated by the ProGen/max instance generator (Schwindt 1998). The test set incorporates

Table 17.1 Control parameters of the considered test set

Parameter	Test set
$\lvert V^r \rvert$	10, 15, 20, 30, 50
RT	0.3, 0.6
r_{ik}	$1, \ldots, 5$
p_i	$1, \ldots, 10$
$[\lvert \mathscr{R} \rvert, RF]$	[1, 1.0], [3, 0.7], [5, 0.6]

instances involving as many as 50 activities, where every activity may require more than a single resource for its execution, if $|\mathscr{R}| > 1$. Moreover, the design of the test set highlights several control parameters for network structure, activities, and resources that affect the behavior of scheduling algorithms. Parameters for the network structure are the number $|V^r|$ of real activities and the restrictiveness of Thesen (RT) that measures the degree to which precedence constraints restrict the total number of feasible activity sequences. We obtain $RT = 0$ for a parallel network and $RT = 1$ for a series network. The parameters for any activity $i \in V$ are the resource utilizations $r_{ik}, k \in \mathscr{R}$, and the processing time p_i. The resource parameters involved are the number $|\mathscr{R}|$ of differing renewable resources, and the resource factor (RF), which denotes the average fraction of the $|\mathscr{R}|$ resources used per activity. Table 17.1 lists the control parameter values. The test set consists of 600 instances, i.e., 20 instances for each $|V^r|$, RT, and $|\mathscr{R}|$ combination.

17.7.2 Performance Study

In a comprehensive performance analysis, we studied the resource leveling problems by considering the four different model formulations. Each instance is solved using CPLEX 12 and ILOG's Concert interface for communications with the solver. Since the RLPs considered are $\mathscr{N}\mathscr{P}$-hard optimization problems, we cannot expect that a branch-and-cut approach provided by CPLEX will terminate within a reasonable time limitation. That is why we allow a maximum computation time of 3 (or 6) h, after which the best solution found up to that point is returned. In order to determine the impact of expanding time-windows, $[ES_i, LS_i]$, for activities $i \in V$, we tested each instance using the shortest possible deadline, $\overline{d} := ES_{n+1}$, as well as deadlines $\overline{d} := \alpha ES_{n+1}$ with $\alpha \in \{1.1, 1.5, 2.0\}$. Moreover, to improve the bounding accuracy and to speed up the solver, particularly for large instances, an initial solution is generated using the Ballestín et al. (2007) heuristic and posted to the solver.

In order to investigate the increases in computation time occasioned by increasing the number of activities, the number of resources, as well as the deadline, we considered blocks of 40 instances (named by $|V^r|$-$|\mathscr{R}|$-α). In addition, we utilized extensions of the resource requirements of activities (i.e., $r_{ik} := 10\, r_{ik}$, $i \in V, k \in \mathscr{R}$), which lead to large increases in the upper bounds $H_{kt}, k \in \mathscr{R}, t \in T$. The corresponding instance names obtained an additional suffix "-10".

Table 17.2 Computation times and number of instances solved (classical RLP, $|V^r| = 10, 15, 20$)

Instances	$\mathrm{M}_1^{\text{C-RL}}$		Tree-based B&B	
	t_{cpu} [s]	$\text{Inst}_{<3\,\text{h}}$	t_{cpu} [s]	$\text{Inst}_{<6\,\text{h}}$
10-1-1.0	0.03	40	0.03	40
10-3-1.0	0.15	40	0.03	40
10-5-1.0	0.20	40	0.05	40
10-1-1.1	0.07	40	0.10	40
10-3-1.1	0.28	40	0.16	40
10-5-1.1	0.63	40	0.38	40
10-1-1.5	0.99	40	0.51	40
10-3-1.5	4.17	40	0.90	40
10-5-1.5	20.29	40	2.31	40
10-1-1.5-10	277.59	36	1.19	40
10-3-1.5-10	744.34	33	2.15	40
10-5-1.5-10	1,181.03	26	5.03	40
15-1-1.0	0.15	40	0.15	40
15-3-1.0	0.29	40	25.72	40
15-5-1.0	0.51	40	29.40	40
15-1-1.1	1.16	40	136.95	40
15-3-1.1	5.67	40	382.97	40
15-5-1.1	7.59	40	688.15	40
15-1-1.5	151.01	40	775.56	40
15-3-1.5	262.68	38	2,813.01	37
15-5-1.5	824.42	40	3,930.29	37
20-1-1.0	0.34	40	1,792.09	38
20-3-1.0	2.41	40	1,540.26	35
20-5-1.0	2.41	40	3,392.97	31
# Opt		933		938

Table 17.2 lists the performance results for the classical resource leveling problem based on model $\mathrm{M}_1^{\text{C-RL}}$ and the corresponding tree-based branch-and-bound method. Column "t_{cpu}" designates the average computation times (in seconds) of all optimally solved instances and column "$\text{Inst}_{<\beta\text{h}}$" displays the number of instances solved to proven optimality within a time limit of β hours. For the MIP-model, we considered a time limit of 3 h. In order to arrive at fair comparisons, the tree-based method is terminated after 6 h (the program uses just one processor core). Line "# Opt" indicates the total number of instances solved to proven optimality within β hours.

As expected, the tree-based algorithm performs very well for instances involving ten activities. The average run times are invariably less than six seconds. Extensions of the time-windows have a crucial impact on the performance of the MIP-approach, since both the number of binary variables, $x_{it}, i \in V, t \in T$, and the number of constraints increased with the increasing length of the time horizon. In contrast, the tree-based method turns out to be much more robust in relation to the deadlines for project completion. Considerations involving extended resource

Table 17.3 Computation times and number of instances solved (overload problem, $|V^r| = 10, 15, 20$)

	$\mathrm{M}_2^{\text{O-RL}}$		Tree-based B&B	
Instances	t_{cpu} [s]	$\text{Inst}_{<3\,\text{h}}$	t_{cpu} [s]	$\text{Inst}_{<6\,\text{h}}$
10-1-1.0	0.01	40	0.02	40
10-3-1.0	0.03	40	0.02	40
10-5-1.0	0.05	40	0.04	40
10-1-1.1	0.03	40	0.05	40
10-3-1.1	0.09	40	0.30	40
10-5-1.1	0.16	40	0.38	40
10-1-1.5	0.10	40	0.12	40
10-3-1.5	0.47	40	0.41	40
10-5-1.5	2.14	40	1.39	40
10-1-1.5-10	0.05	40	0.13	40
10-3-1.5-10	0.39	40	0.44	40
10-5-1.5-10	2.68	40	1.49	40
15-1-1.0	0.10	40	9.73	40
15-3-1.0	0.07	40	20.04	40
15-5-1.0	0.11	40	23.64	40
15-1-1.1	0.27	40	47.85	40
15-3-1.1	0.88	40	262.95	40
15-5-1.1	2.13	40	518.70	40
15-1-1.5	3.12	40	43.00	40
15-3-1.5	80.79	40	1,441.20	39
15-5-1.5	88.53	40	1,912.63	38
20-1-1.0	0.19	40	1,566.70	38
20-3-1.0	1.17	40	2,283.95	37
20-5-1.0	1.52	40	2,751.41	31
# Opt		960		943

requirements also lead to drastic increases in the number of auxiliary variables, y_{kth}, $k \in \mathscr{R}, t \in T, h \in \{1, \ldots, H_{kt}\}$. In particular, model $\mathrm{M}_1^{\text{C-RL}}$ proved incapable of terminating the enumerations for all of the respective instances. Based on the practically-relevant problem dimension of 15 activities, the performance of model $\mathrm{M}_1^{\text{C-RL}}$ must be regarded as acceptable. The computation times of the tree-based method are highly dependent upon the network structure, the activities, and the resources involved. Increasing numbers of activities lead to the existence of many feasible order networks that must be considered during enumeration. Furthermore, the long running times are a consequence of the parallelism of project networks (with $RT < 0.5$) and the existence of a large slack factor $SF \in \{0.5, 1.0\}$. A positive slack factor avoids that activities are firmly tied by temporal constraints. Since the number of instances solved to optimality is relatively low for the tree-based method, we investigated larger instances with 20 activities only with tight deadlines.

Table 17.3 summarizes the results for the overload problem in the case of model $\mathrm{M}_2^{\text{O-RL}}$ and the respective tree-based branch-and-bound method. For small instances

Table 17.4 Computation times and number of instances solved (classical RLP and overload problem, $|V^r| = 20, 30, 50$)

	$\mathrm{M}_1^{\text{C-RL}}$		$\mathrm{M}_2^{\text{O-RL}}$	
Instances	t_{cpu} [s]	$\text{Inst}_{<3\,\text{h}}$	t_{cpu} [s]	$\text{Inst}_{<3\,\text{h}}$
20-1-1.1	4.27	40	2.53	40
20-3-1.1	260.84	40	35.13	40
20-5-1.1	85.70	40	52.29	40
20-1-1.5	1,245.81	36	86.16	39
20-3-1.5	2,669.79	30	340.24	39
20-5-1.5	3,074.42	20	1,395.45	34
30-1-1.0	9.98	40	12.53	40
30-3-1.0	18.39	40	14.16	40
30-5-1.0	61.20	40	101.54	40
30-1-1.1	387.53	36	196.60	36
30-3-1.1	691.95	38	608.06	38
30-5-1.1	1,808.03	33	112.82	36
50-1-1.0	530.36	38	334.41	37
50-3-1.0	675.81	36	926.27	36
50-5-1.0	969.61	32	288.89	28
# Opt		539		563

with ten activities no superiority of the tree-based method over the procedure using model $\mathrm{M}_2^{\text{O-RL}}$ can be ascertained. Extensions of the time-windows have no relevant impacts on computation times. Moreover, the run times for instances having long project durations, as well as instances having long project durations and extended resource requirements are similar, since the number of auxiliary variables, $v_{kt}, k \in \mathscr{R}, t \in T$, depends on the number of resources and the time horizon. For larger instances involving 15 or more activities, model $\mathrm{M}_2^{\text{O-RL}}$ yields the best results. All of the problem instances under consideration are optimally solved within 1.5 min or less.

The computational studies that we have conducted thus far, demonstrate the dominance of the branch-and-cut procedures provided by CPLEX in combination with suitable model formulations. Since model $\mathrm{M}_1^{\text{C-RL}}$ should be considered in order to solve instances of the classical RLP, and model $\mathrm{M}_2^{\text{O-RL}}$ should be considered in order to solve instances of the overload problem, our further analyses involve these methods. Table 17.4 shows that most instances with up to 50 activities and tight project deadlines are solved to optimality. Note that no algorithm known from the literature has found optimal schedules for similar problem sizes or real-world instances within reasonable time.

We continue our analyses by studying the total adjustment cost problem. Since the restrictiveness of Thesen affect the behavior of procedures that take advantage of problem structures during the solution process (i.e., model $\mathrm{M}_4^{\text{A-RL}}$ as well as the tree-based branch-and-bound method), we now consider blocks $|V^r|$-$|\mathscr{R}|$-RT-α of 20 instances. Table 17.5 lists the results for models $\mathrm{M}_3^{\text{A-RL}}$ and $\mathrm{M}_4^{\text{A-RL}}$ as well as the respective tree-based method for instances with 10 real activities.

Table 17.5 Computation times and number of instances solved (total adjustment cost problem, $|V^r| = 10$)

	$M_3^{\text{A-RL}}$		$M_4^{\text{A-RL}}$		Tree-based B&B	
Instances	t_{cpu} [s]	$\text{Inst}_{<3\,\text{h}}$	t_{cpu} [s]	$\text{Inst}_{<3\,\text{h}}$	t_{cpu} [s]	$\text{Inst}_{<3\,\text{h}}$
10-1-0.3-1.0	0.02	20	0.10	20	0.06	20
10-3-0.3-1.0	0.03	20	0.05	20	0.12	20
10-5-0.3-1.0	0.04	20	0.12	20	0.52	20
10-1-0.3-1.5	0.43	20	0.20	20	1.15	20
10-3-0.3-1.5	7.89	20	0.24	20	3.52	20
10-5-0.3-1.5	5.44	20	0.65	20	14.21	20
10-1-0.6-1.0	0.02	20	0.03	20	0.01	20
10-3-0.6-1.0	0.01	20	0.03	20	0.02	20
10-5-0.6-1.0	0.02	20	0.06	20	0.02	20
10-1-0.6-1.5	0.53	20	0.09	20	0.34	20
10-3-0.6-1.5	2.96	20	0.05	20	0.57	20
10-5-0.6-1.5	12.01	20	0.13	20	1.80	20
# Opt		240		240		240

Table 17.6 Computation times and number of instances solved (total adjustment cost problem, $|V^r| = 15, 20$)

	$M_3^{\text{A-RL}}$		$M_4^{\text{A-RL}}$		Tree-based B&B	
Instances	t_{cpu} [s]	$\text{Inst}_{<3\,\text{h}}$	t_{cpu} [s]	$\text{Inst}_{<3\,\text{h}}$	t_{cpu} [s]	$\text{Inst}_{<3\,\text{h}}$
15-3-0.3-1.0	0.21	20	1.63	20	109.29	20
15-3-0.3-1.5	267.55	19	37.36	20	4,538.50	9
20-3-0.3-1.0	8.13	20	80.81	20	1,051.68	9
20-3-0.3-1.5	1,971.78	13	1,864.97	19	–	0
15-3-0.6-1.0	0.05	20	0.17	20	8.85	20
15-3-0.6-1.5	861.35	19	1.09	20	727.47	20
20-3-0.6-1.0	0.08	20	0.62	20	644.59	20
20-3-0.6-1.5	1,153.28	11	8.67	20	1,587.79	2
# Opt		142		159		100

Regarding the average computation times, both models as well as the tree-based branch-and-bound method are efficient. All average run times are lower than 15 s. In order to examine the impact of large activity resource requirements, we also solved instances with $r_{ik} := 10r_{ik}$. As the resource requirements only affect the M_k- or M_{kt}-values, as well as the domains of auxiliary variables in the models, no explicit differences concerning performance may be determined by analyzing the results.

In order to consider larger instances with 15 and 20 real activities, Table 17.6 summarizes the results. We concentrate on problems with three renewable resources and varying project completion deadlines. For instances with $RT = 0.3$ (i.e., rather parallel networks) and tight project completion deadlines ($\alpha = 1.0$), model $M_3^{\text{A-RL}}$ performs well. However, if the value α is increased, the average run times are

Table 17.7 Computation times and number of instances solved (total adjustment cost problem, $|V^r| = 15, 20, \alpha = 2.0$)

	$\mathrm{M}_3^{\text{A-RL}}$		$\mathrm{M}_4^{\text{A-RL}}$	
Instances	t_{cpu} [s]	$\text{Inst}_{<3\,\text{h}}$	t_{cpu} [s]	$\text{Inst}_{<3\,\text{h}}$
15-3-0.3-2.0	974.37	13	85.71	20
20-3-0.3-2.0	4,784.11	5	2,668.58	16
15-3-0.6-2.0	3,080.63	14	1.32	20
20-3-0.6-2.0	1,792.65	4	11.65	20
# Opt		36		76

significantly longer. In contrast, model $\mathrm{M}_4^{\text{A-RL}}$ turns out to be much more robust in relation to the deadlines for project completion. All instances with 15 and nearly all instances with 20 real activities are solved to optimality within 3 h. Furthermore, the tree-based branch-and-bound method is not able to match up to the branch-and-cut procedures provided by CPLEX. Less than 50 % of the problem instances are optimally solved. Considering a rather series network with $RT = 0.6$, model $\mathrm{M}_4^{\text{A-RL}}$ is able to strengthen its success, all instances are solved in less than 10 seconds on average. Additionally, the performance of the tree-based branch-and-bound method is better than for instances with $RT = 0.3$. However, the average run times are higher than those of the two models. In order to arrive at fair comparisons, the tree-based method might have run longer than 3 h. Nevertheless, if we consider the instances solved to proven optimality, the run times of the $\mathrm{M}_4^{\text{A-RL}}$-model and the tree-based method differ by a factor of more than 60, which is above a computation time reduction that a multi-core processor running CPLEX can induce. For the $\mathrm{M}_3^{\text{A-RL}}$-model, the restrictiveness of Thesen has no obvious impact on the run times and the number of instances solved. The procedure obtains similar results for both RT-values. As expected, the restrictiveness of Thesen influences the run times of model $\mathrm{M}_4^{\text{A-RL}}$ as well as the tree-based method. As a rule, a lower value for the restrictiveness of Thesen complicates the problem of finding and proving an optimal schedule. Rather parallel networks involving $RT = 0.3$ induce a large number of feasible activity sequences, whereas in rather series networks many (indirect) precedence constraints lead to a reduction of feasible activity sequences.

We investigate the robustness of models $\mathrm{M}_3^{\text{A-RL}}$ and $\mathrm{M}_4^{\text{A-RL}}$ in relation to varying RT-values and expanded project completion deadlines by setting $\alpha := 2.0$ for instances involving 15 or 20 real activities. Table 17.7 depicts the results. Model $\mathrm{M}_4^{\text{A-RL}}$ indeed works very well. Only four instances with $RT = 0.3$ are not proven optimally solved. Particularly, the run times for instances with $RT = 0.6$ are shorter than expected. As opposed to this, model $\mathrm{M}_3^{\text{A-RL}}$ is only able to solve less than 50 % of all problem instances to optimality within 3 h. Thus, the model, due to its non-polynomial characteristic, is not suitable for instances involving large project completion deadlines.

Table 17.8 shows the results for medium-scale instances with up to 50 real activities and project deadlines $\overline{d} = \alpha ES_{n+1}, \alpha = \{1.0, 1.5\}$. Here, the boundaries of an exact solution methodology become apparent. Within a time limit of 6 h, about half of all instances are optimally solved. Moreover, the strength of model $\mathrm{M}_3^{\text{A-RL}}$ appears

Table 17.8 Computation times and number of instances solved (total adjustment cost problem, $|V^r| = 30, 50$)

	$\mathrm{M}_3^{\text{A-RL}}$		$\mathrm{M}_4^{\text{A-RL}}$	
Instances	t_{cpu} [s]	$\text{Inst}_{<6\,\text{h}}$	t_{cpu} [s]	$\text{Inst}_{<6\,\text{h}}$
30-3-0.3-1.0	120.04	20	6,383.74	12
30-3-0.3-1.5	–	0	16,608.54	1
50-3-0.3-1.0	2,069.61	8	–	0
30-3-0.6-1.0	2.04	20	419.92	20
30-3-0.6-1.5	–	0	3,227.21	16
50-3-0.6-1.0	2,107.37	16	2,458.69	7
# Opt		64		56

by regarding the results for instances with tight project completion deadlines. In contrast, for projects with $\alpha = 1.5$, model $\mathrm{M}_4^{\text{A-RL}}$ produces significantly better results than model $\mathrm{M}_3^{\text{A-RL}}$.

17.8 Conclusions

The chapter considers exact methods for several resource leveling problems. All problems and its objective functions have nice structural properties that could be exploited on the way to an optimal solution. Particularly, the classical resource leveling and the overload function are locally concave. In contrast, the total adjustment cost function is locally quasi-concave and may be determined by considering the start and end times of activities. For all problems, tree-based branch-and-bound methods are described that use the structural conventions. In addition, discrete time-based formulations are introduced that consider binary variables to allocate a feasible start time to each activity. Furthermore, a polynomial model for the total adjustment cost problem (start-based formulation) is presented, which is independent of a scaling of the time axis.

The results of a comprehensive performance analysis show that the discrete time-based formulations are efficient for instances with tight project completion deadlines. The corresponding branch-and-cut procedures are able to solve instances with up to 50 real activities. For the total adjustment cost problem with increased project deadlines, the start-based model performs well. There are even many instances with 30 real activities and $\alpha = 1.5$ that are solved to optimality.

Future research will include the consideration of stochastic activity durations or alternative execution modes, which differ in processing time, time lags, and resource requirements (Hartmann and Briskorn 2010). Furthermore, a simultaneous consideration of (A-RL), (C-RL), and (O-RL) in a multi-criteria approach could be interesting. Since all objective functions are locally (quasi-)concave, there always exists a quasistable schedule that will be optimal for a resulting weighted objective function.

Acknowledgements The benchmarks presented herein and the results obtained (i.e., upper and lower bounds) may be downloaded from the web-page http://www.wiwi.tu-clausthal.de/abteilungen/unternehmensforschung/forschung.

References

Ahuja HN (1976) Construction performance control by networks. Wiley, New York

Ballestín F, Schwindt C, Zimmermann J (2007) Resource leveling in make-to-order production: modeling and heuristic solution method. Int J Oper Res 4:50–62

Bandelloni M, Tucci M, Rinaldi R (1994) Optimal resource leveling using non-serial dynamic programming. Eur J Oper Res 78:162–177

Bartusch M, Möhring R, Radermacher F (1988) Scheduling project networks with resource constraints and time windows. Ann Oper Res 16:201–240

Burgess AR, Killebrew JB (1962) Variation in activity level on a cyclical arrow diagram. Int J Ind Eng 2:76–83

Christodoulou SE, Ellinas G, Michaelidou-Kamenou A (2010) Minimum moment method for resource leveling using entropy maximization. J Constr Eng Manag 136:518–527

Demeulemeester EL, Herroelen WS (2002) Project scheduling: a research handbook. Kluwer, Boston

Easa SM (1989) Resource leveling in construction by optimization. J Constr Eng Manag 115:302–316

Engelhardt H, Zimmermann J (1998) Lower bounds and exact methods for resource levelling problems. Technical Report WIOR-517, University of Karlsruhe, Karlsruhe

Gabow H, Myers E (1978) Finding all spanning trees of directed and undirected graphs. SIAM J Comput 7:208–287

Galbreath RV (1965) Computer program for leveling resource usage. J Const Div Proc Am Soc Civil Eng 91:107–124

Gather T, Rieck J, Zimmermann J (2010) A linearized mixed-integer formulation for the resource levelling problem. In: Proceedings of the 12th international workshop on project management and scheduling, Tours, pp 199–202

Gather T, Zimmermann J, Bartels, J-H (2011) Exact methods for the resource levelling problem. J Sched 14:557–569

Geng J, Weng L, Liu S (2011) An improved ant colony optimization algorithm for nonlinear resource-leveling problems. Comput Math Appl 61:2300–2305

Harris RB (1978) Precedence and arrow networking techniques for construction. Wiley, New York

Harris RB (1990) Packing method for resource leveling (Pack). J Constr Eng Manag 116:39–43

Hartmann S, Briskorn D (2010) A survey of deterministic modeling approaches for project scheduling under resource constraints. Eur J Oper Res 207:1–14

Koné O, Artigues C, Lopeza P, Mongeauc M (2011) Event-based MILP models for resource-constrained project scheduling problems. Comput Oper Res 38:3–13

Kreter S, Rieck J, Zimmermann J (2014) The total adjustment cost problem: applications, models, and solution algorithms. J Sched 17:145–160

Möhring RH (1984) Minimizing costs of resource requirements in project networks subject to a fixed completion time. Oper Res 32:89–120

Neumann K, Zimmermann J (1999) Resource levelling for projects with schedule-dependent time windows. Eur J Oper Res 117:591–605

Neumann K, Zimmermann J (2000) Procedures for resource levelling and net present value problems in project scheduling with general temporal and resource constraints. Eur J Oper Res 127:425–443

Neumann K, Nübel H, Schwindt C (2000) Active and stable project scheduling. Math Method Oper Res 52:441–465

Neumann K, Schwindt C, Zimmermann J (2003) Project scheduling with time windows and scarce resources. Springer, Berlin

Nübel H (1999) Minimierung der Ressourcenkosten für Projekte mit planungsabhängigen Zeitfenstern. Gabler, Wiesbaden

Petrovic R (1969) On optimization of resource leveling in project plans. In: Lombaers HJ (ed) Project planning by network analysis. North-Holland, Amsterdam, pp 268–273

Pritsker AAB, Watters LJ, Wolfe PM (1969) Multi-project scheduling with limited resources: a zero-one programming approach. Manag Sci 16:93–108

Raja K, Kumanan S (2007) Resource leveling using petrinet and memetic approach. Am J Appl Sci 4:317–322

Ranjbar M (2013) A path-relinking metaheuristic for the resource levelling problem. J Oper Res Soc 64:1071–1078

Rieck J, Zimmermann J, Gather T (2012) Mixed-integer linear programming for resource leveling problems. Eur J Oper Res 221:27–37

Savin D, Alkass S, Fazio P (1997) A procedure for calculating the weight-matrix of a neural network for resource leveling. Adv Eng Softw 28:277–283

Schwindt C (1998) Generation of resource-constrained project scheduling problems subject to temporal constraints. Technical Report WIOR-543, University of Karlsruhe, Karlsruhe

Schwindt C (2005) Resource allocation in project management. Springer, Berlin

Selle T (2002) Untere Schranken für Projektplanungsprobleme. Shaker, Aachen

Takamoto M, Yamada N, Kobayashi Y, Nonaka H (1995) Zero-one quadratic programming algorithm for resource leveling of manufacturing process schedules. Syst Comput Jpn 26:68–76

Wiest JD (1963) The scheduling of large projects with limited resources. Unpublished Ph.D. dissertation, Carnegie Institute of Technology, Pittsburgh

Younis MA, Saad B (1996) Optimal resource leveling of multi-resource projects. Comput Ind Eng 31:1–4

Zimmermann J (2001) Ablauforientiertes Projektmanagement: Modelle, Verfahren und Anwendungen. Gabler, Wiesbaden

Chapter 18
Heuristic Methods for Resource Leveling Problems

Symeon E. Christodoulou, Anastasia Michaelidou-Kamenou, and Georgios Ellinas

Abstract A novel resource-leveling algorithm is presented based on entropy concepts, restating the resource-leveling heuristic known as the "Minimum Moment Method", as an "Entropy Maximization Method" and improving on its efficiency. The proposed resource-leveling algorithm makes use of the general theory of entropy and two of its principal properties (subadditivity and maximality) to restate resource leveling as a process of maximizing the entropy found in a project's resource histogram. Entropy in this resource-centric problem domain is defined as the ratio of allocated daily resource units over the total number of resource units to complete the project. Entropy's subadditivity and maximality properties state that if a system consists of two subdomains having *n* and *m* components respectively, then the total system entropy is less than or equal to the sum of the subdomains' entropy, and that the entropy is maximum when all admissible outcomes have equal probabilities of occurrence (maximal uncertainty is reached for the equiprobability distribution of possible outcomes).

Keywords Entropy maximization method • Heuristic methods • Project scheduling • Resource leveling

18.1 Introduction

Resource-Constrained Scheduling Problems (RCSP) refer to a class of scheduling problems the activities of which are assigned resources with limited capacity or of limited availability. The result of such resource availability constraints is that the underlying project network is constrained in its capacity to meet the constructor's

S.E. Christodoulou (✉) • A. Michaelidou-Kamenou
Department of Civil and Environmental Engineering, University of Cyprus, Nicosia, Cyprus
e-mail: schristo@ucy.ac.cy; anastasia.kamenou@cyta.com.cy

G. Ellinas
Department of Electrical and Computer Engineering, University of Cyprus, Nicosia, Cyprus
e-mail: gellinas@ucy.ac.cy

C. Schwindt, J. Zimmermann (eds.), *Handbook on Project Management and Scheduling Vol. 1*, International Handbooks on Information Systems,
DOI 10.1007/978-3-319-05443-8_18

primary objective for timely and cost-efficient project completion. As Senouci and Adeli (2001) state, the need for resource-constrained scheduling *"arises when there are definite limits on the amount of resources available. The scheduling objective is to extend the project duration as little as possible beyond the original critical path duration in such a way that the resource constraints are met. In this process, both critical and noncritical activities are shifted."*

RCSP are $\mathscr{NP}$-hard problems, the complexity of which increases substantially with the project network size. The number of activities and precedence relationships, as well as the number of resource types utilized in the project and the constraints imposed on them are the main sources of complexity in such schedules and add greatly to the difficulty in finding optimal time and resource allocation solutions. An optimal solution in this type of network problems can most often be found by employing computationally intensive analyses or exhaustive enumeration analyses in which all possible outcomes are evaluated. However, because of the aforementioned computational complexity, in most cases near-optimal solutions are sought by means of heuristics.

The resource leveling problem (RLP) is a variation of the RCSP and relates to the need for resolving over-allocations or conflicts in resource usage and/or resolving the unbalanced use of resources over time, aiming at increasing the efficiency in resource utilization. As Senouci and Adeli (2001) state, *"the resource leveling problem arises when there are sufficient resources available and it is necessary to reduce the fluctuations in the resource usage over the project duration. These resource fluctuations are undesirable because they often require a short-time hiring and firing policy. The short-term hiring and firing presents labor, utilization, and financial difficulties because (1) the clerical costs for employee processing are increased; (2) top-notch journeymen are discouraged to join a company with a reputation of doing this; and (3) new, less experienced employees require long periods of training. The scheduling objective of the resource leveling problem is to make the resource requirements as uniform as possible or to make them match a particular nonuniform resource distribution in order to meet the needs of a given project. ... In resource leveling, the project duration of the original critical path remains unchanged."* It should be noted, though, that besides the original critical path it is possible to make use of a prescribed deadline greater than the length of the critical path for the maximal project duration (cf. Chaps. 14–17 of this handbook).

Metaheuristics are computational methods that attempt the solution of problems whose solution is impossible to obtain analytically or difficult to compute (usually $\mathscr{NP}$-hard problems), and they are typically used for combinatorial optimization. Metaheuristics reach a solution by iteratively trying to improve a candidate solution with regard to a given measure of quality and based on few, if any, assumptions about the problem being optimized.

It should be noted that, even though metaheuristics do not guarantee an optimal solution they usually achieve better results and better performance than their classic counterparts (e.g., classical local search), because in reaching a global optimal solution metaheuristics have the ability to overcome local minima in the solution space. Some of the most well-known metaheuristic methods are: genetic algorithms (GA's), evolutionary programming, Tabu search, particle swarm optimization (PSO), ant colony optimization (ACO), and simulated annealing.

A definition of the resource leveling problem (RLP) is given in Chap. 17 of this handbook. The remaining of this chapter will utilize the objectives, constraints, and notation (where possible) of the notions described in Chap. 17 of this handbook, and consider activity-on-node networks with only precedence constraints. Instead of an exact solution method, though, this chapter will focus on outlining a heuristic approach to reaching a solution (optimal or near-optimal). The proposed RLP heuristic is based on the method presented by Christodoulou et al. (2010), which utilizes the general theory of entropy and two of its principal properties (subadditivity and maximality) to restate resource leveling as a process of maximizing the entropy found in a project's resource histogram.

Further to the short introduction and literature review, this chapter introduces readers to the Minimum Moment and PACK methods, prior to shifting focus to the concepts of entropy and their applicability to resource leveling. A simplified mathematical depiction of the proposed entropy maximization method is then presented and subsequently expanded to address resource leveling. The chapter moves to an example network, to demonstrate the method, and then to the listing of an algorithm on the proposed entropy maximization method. The chapter ends with concluding remarks on the proposed heuristic method.

18.2 Literature Review

There are, in general, two categories of scheduling paradigms (or else, schedule generation schemes) that can be used in addressing resource-constrained problems: serial scheduling and parallel scheduling. A serial schedule generation scheme makes use of an activity-incrementation principle while a parallel schedule generation scheme follows a time-incrementation scheme.

In the former paradigm, a priority list of activities is determined at time zero, with the order of activities in the priority list independent of the resource constraints. Given the priority list, activities are scheduled at the earliest possible time at which the precedence constraints are satisfied and the resources are available. In essence, the serial method is an activity-based schedule generation scheme, with the schedule obtained by first prioretizing activities (by their early start) and then scheduling them, one at a time, at their earliest possible time allowed by both resource and precedence constraints.

In the latter scheduling paradigm, the order of activities is not determined at time zero and scheduling decisions are made when activities are planned to begin and resources are available. In essence, this is a time-based schedule generation scheme, in which again all of the activities should be prioritized but the priority list is not required to be precedence-feasible. In terms of parallel resource-constrained scheduling, the fundamental issue is which activity among competing activities will receive the required resources first. In order to answer this question the construction planner has to dynamically evaluate his options at every junction in the construction process and decide on the activity that would maximize the constructor's "return on investment", which usually means the activity that would minimize the overall project duration.

Common approaches to solving RCSP utilize implicit enumeration and backtracking, such as branch and bound methods, or intelligent branching and evaluation techniques, such as mathematical programming, dynamic programming and zero-one programming. Common are also artificial agent techniques such as genetic algorithms (Hegazy 1999) and ant colony optimization (Christodoulou and Ellinas 2010), or heuristic techniques, examples of which are the methods proposed by Brucker et al. (1998) and Harris (1990).

In terms of resource-leveling problems (RLP) and the utilization of non-exact methods for solving such problems, one of the most widely used resource-leveling heuristic is the *Minimum Moment Method* (Harris 1978) and a variation of it called *"Packing Method (PACK)"* (Harris 1990). These methods attempt to convert an unleveled resource profile into one of rectangular shape by discretizing the overall project resource histogram into smaller time intervals and minimizing the total moment of the resulting resource-time rectangles about the histogram's horizontal axis. The methods, which are explained in detail in subsequent sections, have been computerized and extended to account for any number of different resource types utilized in a project.

Resource leveling was the subject of work presented by Burgess and Killebrew (1962), who had developed a computer program to solve RLP by moving in time activities with slack. Their algorithm relies on empirical rules and measures progress towards the objective of leveling by considering and in effect minimizing the sum of squares of the resource demands in all time periods.

In the work by Seibert and Evans (1991), a serial-type resource-leveling method is described which ranks activities based on user-defined priority rules, and then schedules the project on an activity-by-activity basis, attempting to keep the project within the given resource constraints. If that is not possible, the method attempts to exceed the availability limits in a uniform manner aiming for a leveled resource profile. A measure of variance between the resource-leveled and the unleveled resource profile can be obtained by use of a utilization factor, defined as the ratio of the resource-leveled amount during the period of interest over the initial profile value during the same period. This factor is also used in determining where the assumed resource profile should be revised for subsequent resource-leveling runs, while a metric for acceptable residuals is used during the leveling phase for benchmarking the degree of leveling obtained. The metric is defined as the assumed resource level multiplied by the acceptable utilization-factor deviation and a well-leveled resource histogram should have the sum of the squares of the residuals approaching zero.

Takamoto et al. (1995) utilize an objective function which is monotonic and which simply increases with the degree of resources leveling, and then solve the optimization problem by fixing the process start dates while minimizing the objective function. Recognizing that in many cases the network to solve is large, and that it is very difficult to obtain a global optimization solution, the researchers developed an algorithm which quickly searches for a good suboptimal solution close to the global optimal solution of a 0–1 quadratic programming problem. The developed algorithm searches the solution space by repeating a pivot operation using variable selection rules for resources leveling.

In the work by Brinkmann and Neumann (1996), heuristics for two types of resource-constrained project-scheduling problems were examined: the problem of levelling the resources consumption and the minimum project-duration problem. The researchers present two different heuristic procedures (the sequential or direct method, and the contraction method) concluding that for the minimum project-duration problem, the contraction method is superior to the direct method and that for the resource-levelling problem, both approaches behave similarly. In the direct method the nodes are processed successively without scheduling the strong components of the network separately, while in the contraction method feasible subschedules are first found for each strong component of the cyclic network and replaced by a single node, with the resulting (contracted) acyclic network then treated by the direct method.

Neumann and Zimmermann (1999) presented polynomial heuristic procedures for different types of resource leveling problems for projects with minimum and maximum time lags between project activities. Problems with and without explicit resource constraints were also investigated and reported on. The researchers also dealt with resource-constrained project scheduling problems with nonregular objective functions, where general temporal constraints given by minimum and maximum time lags between activities are prescribed, such as the case of resource leveling and net present value problems (Neumann and Zimmermann 2000). The authors presented several algorithms, along with a detailed experimental performance analysis of the algorithms, showing that the procedures examined could solve large problem instances in reasonable computing time.

Senouci and Adeli (2001) presented a mathematical model for resource leveling focused on minimizing the total project cost as opposed to previous resource scheduling formulations which traditionally focused on minimizing the project duration. In their formulation, resource leveling and resource-constrained scheduling are performed simultaneously and additional project scheduling characteristics such as precedence relationships, multiple crew-strategies, and time-cost trade-off are considered. The resource scheduling problem is formulated as a constrained optimization problem which is then transformed into an unconstrained optimization problem using the exterior penalty function method and solved by using the neural dynamics model of Adeli and Park (1995).

Zhang et al. (2005) put forward a permutation-based scheme for RCSP based on particle swarm optimization (PSO). This method uses a hybrid particle-updating mechanism coupled with a partially mapped crossover of a genetic algorithm in order to handle the permutation-feasibility and precedence-constraint problems when updating the particle-represented sequence or solution for the RCSP. The particle-represented sequence is transformed into a schedule including start times and resource assignments for all activities by means of a serial method and then evaluated against the objective of minimizing project duration. As the authors state, the permutation-based PSO *"is more robust than general analytical and heuristic methods, because it does not lead to combinatorial explosion or problem-dependent effectiveness. It is able to search for global optima as GA does. However, unlike GA, which needs to predefine the crossover probability and performs crossover between two unclassified survivals or individuals, the proposed PSO uses the dynamic probability and performs partially mapped crossover updating of a particle by mapping it with a reference particle (i.e., the particle's local best or the global best sequence found so far)."* (Zhang et al. 2005).

Similar to PSO are the methods presented by Christodoulou and Ellinas (2010) on ant colony optimization algorithms for both resource-unconstrained and resource-constrained scheduling. Ant colony optimization (ACO) is a population-based, artificial multi-agent, general-search technique with its theoretical roots based on the behavior of real ant colonies and the collective trail-laying and trail-following of its members in searching for optimal solutions when traversing multiple paths. The work presented by Christodoulou and Ellinas (2010) outlines the fundamental mathematical background of the ACO method and a suggested possible implementation strategy for resource leveling.

Work on ACO was also presented by Garmsiri and Abassi (2012), who proposed a resource leveling approach that can be used in projects with multi-mode execution activities. In the proposed method, artificial ants select an execution mode for each activity and establish start-to-end routes based on heuristics and a resource-leveling index, with the resulting project schedule established based on the ant route which has the optimal objective function.

Noteworthy are finally the works by Ballestín et al. (2007) and Ranjbar (2013) who presented additional heuristic methods for solving RLP. Ballestín et al. (2007), through an experimental performance analysis, concluded that a population-based iterated greedy method proposed by them, compared favorably to leveling heuristics from literature with respect to accuracy and computation time. Iterated greedy is a metaheuristic belonging to the class of local search methods, and the researchers utilize a large-neighborhood search method to reschedule freed activities within the time windows defined by the fixed start times of frozen activities. In the proposed method, an initial solution is first constructed and the associated *ES* and *LS* dates computed. After the computation of the initial solution, the iterated greedy method performs a stochastic local search in the set of feasible solutions by iterating cycles of destruction and construction phases, during which partial schedules (destruction phase) and complete schedules (construction phase) are generated.

Ranjbar (2013) presented a path-relinking metaheuristic algorithm which, instead of directly producing new solutions when combining original ones, it generates paths between and beyond the selected solutions in the neighborhood space. Thus, relinking explores trajectories connecting elite solutions obtained by heuristic methods. The relinking method constructs these trajectories (paths) as sequences of elements in which each element is a vector of start times and differs with its previous and next elements in the start time of only one activity. The author concluded that the proposed relinking method compared favorably to leveling heuristics from literature.

18.3 Minimum Moment and PACK Methods

As already noted, a number of heuristic methods have been developed over the years for addressing resource leveling. One of the most common methods is the *Minimum Moment Method* (Harris 1978) and a variation of it called *"Packing Method (PACK)"* (Harris 1990). The methods are based on the observation that a perfectly leveled project has a rectangularly shaped resource histogram and by extension on the premise that leveled projects result in a more efficient project execution. This, in turn, is based on the premise that the fluctuation in daily resource usage is linked to inefficiencies in their utilization and in their management and thus in inefficiencies in the project time and cost. In attempting to transform a project's original resource histogram into one of rectangular shape, both the Minimum Moment Method and the PACK method utilize a heuristic to assign activities to specific days and to build up the resource histogram, so that the final resource histogram approaches a rectangle and its moment about the x-axis converges to a minimum value.

The project's total resource moment (M_x) is obtained by summing up the individual resource moments about the time axis and it can be expressed as

$$M_x = \sum_{\chi=1}^{n_t} \left[\left(t_\chi r_\chi \right) \left(\frac{1}{2} r_\chi \right) \right] \tag{18.1}$$

where χ is the time interval index, n_t is the number of time intervals comprising the resource histogram, t_χ and r_χ are the time and resource values of the χ^{th} histogram interval respectively. Alternatively, t_χ is the time between two adjacent jump points in the resource profile and r_χ represents the total amount of the resource required for those activities in progress in the corresponding interval. The factor $1/2$ in the aforementioned equation stems from the fact that the moment of each histogram bar about the x-axis is calculated as the bar's area ($t_\chi r_\chi$) times the bar's center of gravity, or alternatively half the bar's height ($\frac{1}{2} r_\chi$). For a daily resource histogram, the time-step equals to 1 ($t = 1$), the number of time intervals equals the project duration and Eq. (18.1) becomes

$$M_x = \sum_{\chi=1}^{n_t} \left[0.5 r_\chi^2 \right] \tag{18.2}$$

If we consider projects specified by activity-on-node networks $N = (V, E, \delta)$ where V is the set of vertices, E is the set of arcs, and δ are the arc weights, we denote the start time of each activity $i \in V$ by S_i and we let $\mathscr{R}$ be the set of (discrete) renewable resources available at each point in time (independently of their utilization), then the amount of resource $k \in \mathscr{R}$ used constantly during execution of activity $i \in V$ represents the resource requirement and it is denoted by r_{ik}. Furthermore, given some schedule $S = (S_i)_{i \in V}$, the active set $\mathscr{A}(S,t)$ contains all real activities in progress at time t, and $r_k(S,t) := \sum_{i \in \mathscr{A}(S,t)} r_{ik}$ represents the total amount of resource $k \in \mathscr{R}$ required for the activities which are in progress at time t. With the aforementioned in mind, Eq. (18.2) becomes

$$M_{r_k} = \sum_{t=0}^{\overline{d}} \left[0.5 r_k^2(S,t)\right] \tag{18.3}$$

where, t denotes time; $r_k(S,t)$ is the amount of resource k used at time t given schedule S; M_{r_k} is the moment of resource k about the horizontal (time) axis; and $\overline{d}$ is the prescribed maximum project duration.

The method's objective is the reduction of daily fluctuations in resource demand by shifting activities in time and within each activity's free float whilst avoiding to impact successor activities. This activity shifting is typically time-constrained (the overall project duration is expected to remain unchanged) and not necessarily resource-constrained. The method's operating assumptions are the following:

- activities are time-continuous and, thus, once started they cannot be interrupted,
- resource assignments for each activity are assumed constant throughout the duration of the activity,
- the duration of each activity remains as originally planned,
- the project's logic (activity relationships) is fixed,
- the project's total duration is fixed (this may result in a project duration that is different than the one obtained from the critical path).

The priority rules used by the method in leveling resources are: (1) to place the activities in decreasing resource rate order; (2) if a tie on resource rates exists, the second priority is to place the tied activities in increasing order of total float; (3) if there is a tie on both the resource rate and the total float, the priority is to place the activities in decreasing order of sequence steps; and (4) if there still a tie, place these tied activities in an arbitrary order.

Computationally, the method requires two distinct cycles (a forward and a backward pass) that mimic the cycles of the traditional CPM. In the forward pass, a resource improvement factor (as defined by Harris 1978) is computed for all activities on the last sequence step of the network and the activity producing the largest positive improvement factor is shifted. This process is repeated for each sequence step until the first step is reached.

The improvement factor, which is computed for every possible shift of an activity within its free float, is used to compare activities within the same group regarding their capacity to reduce the moment of the histogram. This improvement factor is, in effect, the change in the resource moment, computed as the original moment minus the resulting (after an activity shift) resource moment.

The method, which was originally developed for single-type resource leveling, was computerized by Martinez and Ioannou (1993) and later extended by Hiyassat (2000, 2001) to account for multiple resource leveling (*Modified Minimum Moment Method*). A detailed description of the Minimum Moment Method can be found in the original work by Harris (1978), as well as in the work by Martinez and Ioannou (1993) and Hegazy et al. (2000). Extensions of the original method and further case studies can also be found in Harris (1990), Martinez and Ioannou (1993), and Hiyassat (2000). The Minimum Moment Method can also be modified to account for the resource utilization period by also considering the moments about the vertical axis (*Double Moments Method*). A higher resource moment about the vertical axis indicates that the resource remains employed in the project till a later date, thus its utilization period is higher (Hegazy 1999). The focus should usually be on minimizing both the daily resource fluctuations (minimum moment about the horizontal axis) and on reducing the resource utilization period (minimum moment about the vertical axis).

The PACK Method (Harris 1990) is based on the idea of "packing" activities one-by-one so that their daily resource requirements fill the largest gaps in the original daily resource histogram. Instead of shifting activities, the PACK method first builds a histogram considering only critical activities and the remaining activities are ordered in a processing queue based on the daily resource requirement (in decreasing order), the total float (in increasing order), and the sequence step (in decreasing order). Activities are then hierarchically selected from the processing queue and positioned in time between the originally scheduled early start and late start time. The activity shift is chosen so that it minimizes the sum of the daily resource requirement and it considers the impact on the successor activities.

18.4 Entropy Maximization Method

More recently, the Minimum Moment Method was revisited by Christodoulou et al. (2010) and was restated as an entropy maximization problem giving the method a different perspective and improving on its efficiency.

At the core of the method is the concept of entropy, its utilization as a metric of disorder (Christodoulou et al. 2009a,b) and two of entropy's principal properties. Entropy (H_x) in its classical definition is considered to be a metric of a system's order and stability and it can mathematically be evaluated as the product of the probability density function, $f_{\tilde{x}}(x)$, of a random variable $\tilde{x}$, times the natural logarithm of the inverse of this probability (Eq. 18.4).

$$H_x = f_{\tilde{x}}(x) \cdot ln\,(1/f_{\tilde{x}}(x)) \tag{18.4}$$

Similarly, in the case of a discrete distribution the total entropy, H_T, can be evaluated as the sum of the values of the product defined by Eq. (18.4) at each value of the variable (Eq. 18.5).

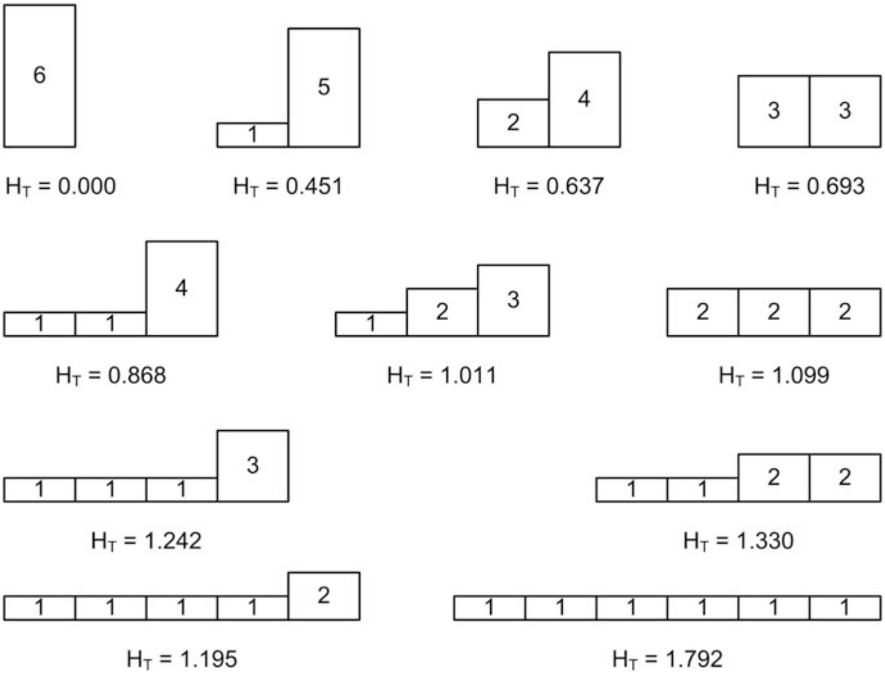


Fig. 18.1 Illustration of entropy's subadditivity and maximality properties

$$H_T = \sum_{x \in \mathbb{N}} \left\{ f_{\tilde{x}}(x) \cdot ln \left(\frac{1}{f_{\tilde{x}}(x)} \right) \right\} \tag{18.5}$$

Among the principal properties of entropy, two are of particular importance: subadditivity and maximality. The subadditivity property states that the entropy value for the sum of two elements is always less than or equal to the sum of the values at each element. The maximality property states that the entropy function takes its greatest value when all admissible outcomes have equal probabilities (i.e., the case of equiprobability). An illustrative example of these properties is given in Fig. 18.1, in which the possible allocations (in time) of six resources are depicted, along with the resulting project entropy values based on Eq. (18.7). As it can be seen, the project entropy increases as the number of splits increases and reaches a maximum when there is equiprobability.

Based on the aforementioned properties, the resource-leveling problem was restated as an entropy-maximization problem with its objective being the convergence to the equiprobability distribution in the daily resource histogram. The objective function was formulated as shown in Eq. (18.7), with the total project entropy denoted by H_T and the value of $f_{\tilde{x}}(x)$ in the entropy equation (Eq. 18.5) replaced with the ratio of assigned resource units, $r_k(S,t)$, over the total number of resource units needed to complete the project (w_k). If $r_k(S,t) = 0$ the value of the corresponding summand H_{tk} is zero (Eq. 18.6).

$$H_{tk} = \begin{cases} \frac{r_k(S,t)}{w_k} ln\left(\frac{1}{r_k(S,t)/w_k}\right) = -\frac{r_k(S,t)}{w_k} ln\left(\frac{r_k(S,t)}{w_k}\right) & \text{if } r_k(S,t) > 0 \\ 0 & \text{if } r_k(S,t) > 0 \end{cases} \tag{18.6}$$

$$H_T = \sum_{t=0}^{\overline{d}} H_{tk} \tag{18.7}$$

In the above equation, k is a single resource belonging to the set $\mathscr{R}$ of (discrete) renewable resources (e.g., workers) utilized in the project in examination; $r_k(S,t)$ is the amount of resource k used at time t given schedule S of prescribed duration $\overline{d}$; and w_k is the total number of units of resource k needed to complete the project. The value of w_k can be computed by summing up the resource units r_{ik} used by each activity i (of duration p_i) in the schedule ($w_k := \sum_{i \in V} p_i r_{ik}$). In the case of multiple resource types, the total project entropy was defined by Christodoulou et al. (2010) to be

$$H_T = \sum_{k=1}^{K} \sum_{t=0}^{\overline{d}} H_{tk} \tag{18.8}$$

where k is the resource-type index; K is the number of different resource types used in the project; $r_k(S,t)$ is the number of units of resource type k used on time unit t, and w_k is the total number of units of resource type k used in the project.

Thus, the entropy-maximization resource-leveling set of equations becomes,

$$\text{Max. } H_T = \text{Max. } \sum_{k=1}^{K} \sum_{t=0}^{\overline{d}} H_{tk} \tag{18.9}$$

$$\text{s. t.} \quad r_{ik} \in \mathbb{N} \quad (i \in V; k \in \mathscr{R}) \tag{18.10}$$

$$S_i + p_i \leq S_j \quad ((i,j) \in E) \tag{18.11}$$

$$S_{n+1} \leq \overline{d} \tag{18.12}$$

$$S_0 = 0 \tag{18.13}$$

$$\sum_{t=0}^{\overline{d}} x_{it} = 1 \quad (i \in V) \tag{18.14}$$

Furthermore, since w_k is the total number of resource units needed to complete the project, the following inequalities also hold true:

$$\sum_{t=0}^{\overline{d}} [r_k(S,t)] \leq w_k \quad (k \in \mathscr{R}) \tag{18.15}$$

$$r_k(S,t) \leq w_k \quad (k \in \mathscr{R}; t = 0, 1, 2, \ldots, \overline{d}) \tag{18.16}$$

The constraint of Eq. (18.10) in the above model formulation refers to the assumption that resource assignments are of integer value since fractional resource assignments are disallowed as non-physical. Equation (18.11) refers to the finish-start precedence relations with a time lag of zero, while Eqs. (18.12) and (18.13) impose, respectively, a prescribed project deadline and a project start at time instance zero. Equation (18.14) ensures that each activity $i \in V$ receives exactly one start time (i.e., activities are not allowed to be interrupted during their execution). The variable x_{it} in Eq. (18.14) is binary ($x_{it} = 1$ if activity i is started at time t; $x_{it} = 0$ otherwise), and $S_i = \sum_{t=0}^{\overline{d}} t x_{it}$ holds for all activities $i \in V$ (S_i being the start time of activity i). Then, the number of units of resource type k used on time unit t can be written as $r_k(S,t) = \sum_{i \in V} \sum_{\tau = \max(t - p_i + 1, 0)}^{t} x_{i\tau} r_{ik}$. Equation (18.15) satisfies the overall project resource-availability constraints, while Eq. (18.16) satisfies the renewable resource constraints.

It should be noted that the entropy method can also provide the planner with a measure of the obtained degree of optimization in terms of the theoretical lower and upper bounds. The theoretical lower bound is always $H_T = 0.0$, obtained when all resource units are employed on a single day, thus enabling the project at hand to finish in a single day. The upper bound can be obtained by invoking entropy's principle of maximality, according to which the entropy function takes the maximal value when all admissible outcomes have equal probabilities (i.e., an equiprobable distribution of resources). Thus, the upper bound of Eq. (18.7) can be obtained by dividing the total man-days required to complete the project by the total project duration ($\overline{d}$) and, for a project of a single-type resource (with w_k being the total number of units of this resource used in the project), it can be written as

$$0 \leq H_T \leq -\overline{d} \left(\frac{w_k/\overline{d}}{w_k} \right) ln \left(\frac{w_k/\overline{d}}{w_k} \right) = ln \left(\overline{d} \right) \tag{18.17}$$

In reality, both the lower and upper bounds are restricted by (1) the project relationships manifested in the network graphs and thus the time-sequence of resource requirements, (2) the total number of resources available per time-period, and (3) the exclusion from the solution space of the "non-physical" (fractional) resource assignments (Eq. 18.10).

The solution of Eqs. (18.9)–(18.16) can be obtained by either Monte Carlo simulation or by analytical methods.

18.5 RLP with Prescribed Fixed Activity Durations

As a demonstration of the proposed heuristic method for RLP, let us consider the case of a resource-leveling example from literature with prescribed fixed activity durations. Figure 18.2 presents a small network (Harris 1990) consisting of eleven activities and requiring a single resource type (e.g., "laborer"). A duration of 16 days and a total number of 108 man-days are required to complete the project, with the project network calculations for the unleveled schedule shown in Table 18.1.

If both the project duration and the individual activity durations are assumed fixed, then by use of Monte Carlo simulation the resource histogram's highest entropy value is obtained at $H_T = 2.714$, corresponding to daily resource assignments of $\{7, 9, 7, 7, 10, 10, 10, 6, 6, 6, 5, 7, 7, 7, 3, 2\}$. This is identical to the solution provided by Harris (1990) (Table 18.1).

The resource-based entropy metric (Eq. 18.8) for the network shown in Figs. 18.2 and 18.3 can be shown to be $H_T \approx 2.475$ for the unleveled state and $H'_T \approx 2.714$ for the leveled state. The values are in agreement with entropy's subadditivity and maximality properties, according to which higher entropy values indicate better leveling. Furthermore, since optimal leveling, as per Eq. (18.17), would have been achieved at an entropy value of $H'_{T_{max}} = ln(16) \approx 2.773$, the computed resource-leveled schedule exhibits a 89.25 % optimization. It should also be noted that 100 % optimization cannot always be reached due to time lags between the activities.

The aforementioned RLP case can be easily transformed into a generic RCSP case, by removing the constraint on fixed activity durations. In that case, and assuming that the project duration is still time-constrained (duration $\leq$ 16 days) then by use of simulation a number of feasible entropy-based and resource-leveled solutions can be obtained. The solutions are dependent on the level of flexibility

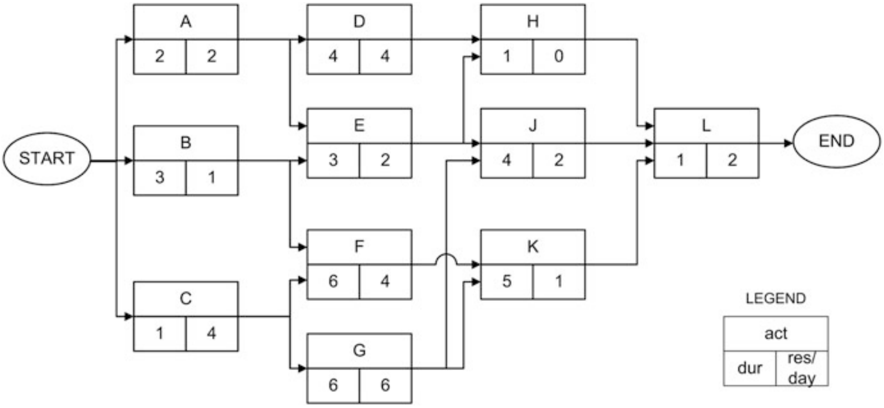


Fig. 18.2 Example 1—AoN network (from Harris 1990)

Table 18.1 Example 1—CPM calculations for unleveled and leveled networks (based on the *Minimum Moment Method* [from Harris 1990] and the *Entropy Maximization Method* [from Christodoulou et al. 2010])

			Unleveled		Leveled[a]		Leveled[b]	
Activity	Resources/Day	Duration (days)	ES	EF	ES	EF	ES	EF
START	0	0	0	0	0	0	0	0
A	2	2	0	2	0	2	0	2
B	1	4	0	4	0	4	0	4
C	4	1	0	1	0	1	0	1
D	4	4	2	6	10	14	10	14
E	2	3	4	7	7	10	7	10
F	4	6	4	10	4	10	4	10
G	6	6	1	7	1	7	1	7
H	0	1	7	8	14	15	14	15
J	2	4	7	11	11	15	11	15
K	1	5	10	15	10	15	10	15
L	2	1	15	16	15	16	15	16
END	0	0	16	16	16	16	16	16

[a] Schedule leveled with the *Minimum Moment Method*
[b] Schedule leveled with the *Entropy Maximization Method*

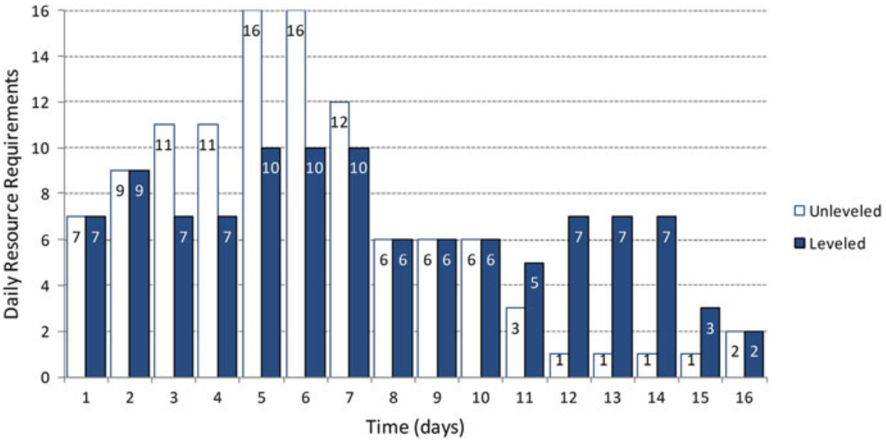


Fig. 18.3 Example 1—Resource histogram of unleveled and leveled schedules (based on the *Minimum Moment Method* [from Harris 1990] and the *Entropy Maximization Method*)

assumed for each activity duration (i.e., the activity ‘stretching’ and ‘compression’ factors). Detailed results pertaining to the aforementioned example study of Harris (1990) can be found in the work reported by Christodoulou et al. (2010), along with additional and more complex examples utilizing finish-to-start, start-to-start and finish-to-finish relationships.

18.6 Entropy-Maximization Resource-Leveling Algorithm

As in the case of other popular resource-leveling heuristics, the proposed entropy-maximization resource-leveling algorithm is a CPM-based heuristic relying on project characteristics and on decision-making rules for prioritizing the activities to be resource-leveled. The heuristic makes use of entropy-based decision criteria as well as the earliest late-start rule for evaluating the process at each network node and for ranking the activities to be leveled based on their contribution to the project's total entropy. The earliest late-start (ELS) rule is advantageous compared to the least total-float (LTF) rule because the values of the late-start dates are derived from the original CPM calculations, unlike the total float values which change every time an activity is rescheduled during the resource leveling process. Thus, the ELS rule can be applied with much less computational effort than the LTF rule (Hegazy et al. 2000).

The entropy-maximization algorithm listed below is an adaptation of the algorithm proposed by Martinez and Ioannou (1993) on the minimum moment method (Harris 1978) and makes use of the expected change of entropy ($\Delta H_T = H'_T - H^0_T$) from state H^0_T to state H'_T, as a result of a resource shift of "m" time units.

Figure 18.4 is used as an aid to the algorithm, with the ΔH value expressed analytically by Eq. (18.18). For the sake of simplicity in listing the algorithm, only one resource type is considered and the index k in all resource-related equations

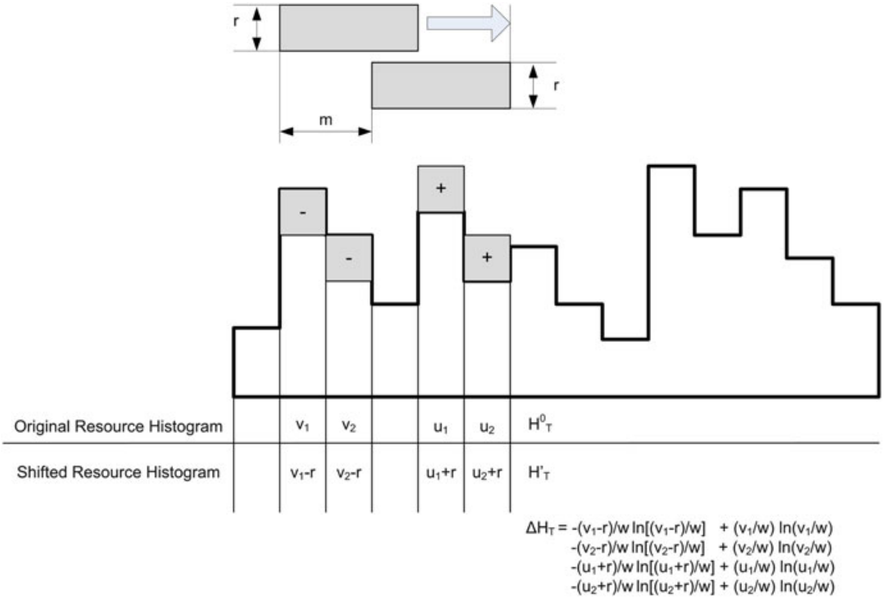

r
m
r
-
-
+
+
Original Resource Histogram
v1
v2
u1
u2
H0T
Shifted Resource Histogram
v1-r
v2-r
u1+r
u2+r
H'T
ΔHT = -(v1-r)/w ln[(v1-r)/w] + (v1/w) ln(v1/w)
-(v2-r)/w ln[(v2-r)/w] + (v2/w) ln(v2/w)
-(u1+r)/w ln[(u1+r)/w] + (u1/w) ln(u1/w)
-(u2+r)/w ln[(u2+r)/w] + (u2/w) ln(u2/w)

Fig. 18.4 Change in entropy of a project's resource histogram following an activity shift

in this chapter is dropped. Thus, $r_k(S,t)$ and w_k are expressed as $r(S,t)$ and w respectively, with $r(S,t)$ being the amount of resource in use at time t, and w being the total required resource effort for completing the project. The parameters v_q and u_q in Eq. (18.18) are used to describe the reduction or increase, respectively, in total resource usage at time q, as a result of a resource shift of r units during the leveling process (Fig. 18.4).

$$\Delta H_T = \sum_{q=1}^{m}\Bigg[-\frac{v_q - r}{w} ln\left(\frac{v_q - r}{w}\right) + \frac{v_q}{w} ln\left(\frac{v_q}{w}\right) - \frac{u_q + r}{w} ln\left(\frac{u_q + r}{w}\right) + \frac{u_q}{w} ln\left(\frac{u_q}{w}\right)\Bigg] \tag{18.18}$$

A description of the algorithm follows.

1. Preliminary calculations:
   1.1 Compute the network
   1.2 Schedule all activities i at their early start ES_i
   1.3 Assign sequence steps to each activity. These sequence steps are based on the early start date (ES_i) of the activities, and they are used as identifiers in grouping activities with the same ES_i
   1.4 Group the activities by sequence step (from [1.3])
   1.5 Compute the forward (early) free float (EFF_i) of each activity
   1.6 Calculate the daily resource histogram that corresponds to the early start dates
   1.7 Compute the project duration ($\overline{d}$) and the total number of resources required to complete the project (w)
2. Forward Pass: Starting with the last group of activities, the following steps are performed,
   2.1 Shift all activities j with zero daily resource requirements ($r_j = 0$), by the amount of their forward free float EFF_j
   2.2 Remove from the group all activities with no forward free float. If no activities remain in the group go to Step 5
   2.3 For the other activities j in the group:
      2.3.1 Define the "improvement factor (IF)" as
      $IF(j, \Delta_j) :=$
      $\sum_{q=1}^{m}\left[-\frac{v_q-r}{w} ln\left(\frac{v_q-r}{w}\right) + \frac{v_q}{w} ln\left(\frac{v_q}{w}\right)\right] +$
      $\sum_{q=1}^{m}\left[-\frac{v_q+r}{w} ln\left(\frac{v_q+r}{w}\right) + \frac{v_q}{w} ln\left(\frac{v_q}{w}\right)\right]$
      where m is the total number of time-steps the shift affects. Compute the improvement factor $IF(j, \Delta_j)$ for all possible forward shifts Δ_j, for $\Delta_j = 1, 2, \ldots, EFF_j$.
      2.3.2 Select the shift that produces the greatest improvement factor. If more than one shift produces the maximum improvement factor, select the largest shift

2.3.3 Call this shift Δ_j^* and the corresponding maximum improvement factor $IF_j = IF(j, \Delta_j^*)$

3. From within these activities:

   3.1 Select the one with the greatest IF_j
   3.2 If there is a tie, select the one with the largest r_j
   3.3 If still tied, select the one with the largest Δ_j
   3.4 If still tied, select the one with the latest ES_j
   3.5 The selected IF_j is defined as IF_j^*

4. If the IF_j^* for the selected activity j is negative go to Step 5, otherwise:

   4.1 Carry out the shift
   4.2 Update the forward and backward free float for all activities
   4.3 Update the resource histogram
   4.4 Go back to Step 2.2

5. Consider the group of activities in the earlier group and then go back to Step 2.1 until all the groups have been considered
6. Backward Pass: Repeat the forward pass steps, but this time (i) consider the groups in reverse order, (ii) shift activities backward to earlier times rather than forward to later times.

The aforementioned algorithm was kept rudimentary in nature, for the purpose of explaining the method. The efficiency of the algorithm can be improved by approximating Eq. (18.18) with a series of Taylor expansions. A computationally more efficient algorithm is currently in development, based on the cross-entropy algorithm (Rubinstein and Kroese 2004) and a greedy-search algorithm that simultaneously considers the resulting entropy values at times t and $t + 1$ in an attempt to reduce the computing steps/time by half.

18.7 Sample Implementation of Proposed Algorithm

As a demonstration of the aforementioned algorithm, consider the following simple project network of six activities and of a single resource type ("laborers") with a daily resource histogram as shown in Fig. 18.5a. The total project duration is 6 days (shaded bars in the Gantt chart indicate critical activities) and the total effort required to complete the project is 16 person-days.

By use of Eq. (18.7), the original project entropy is 1.630. If a daily resource constraint of three laborers is enforced, then the project shows a resource over-allocation in its first 3 days which calls for resource-leveling. Application of the entropy-based leveling algorithm results in a revised (leveled) Gantt chart and resource histogram (Fig. 18.5b) by shifting selected activities to later start dates. The activity shifts (in turn: A, C, and E twice) result in project entropy values of

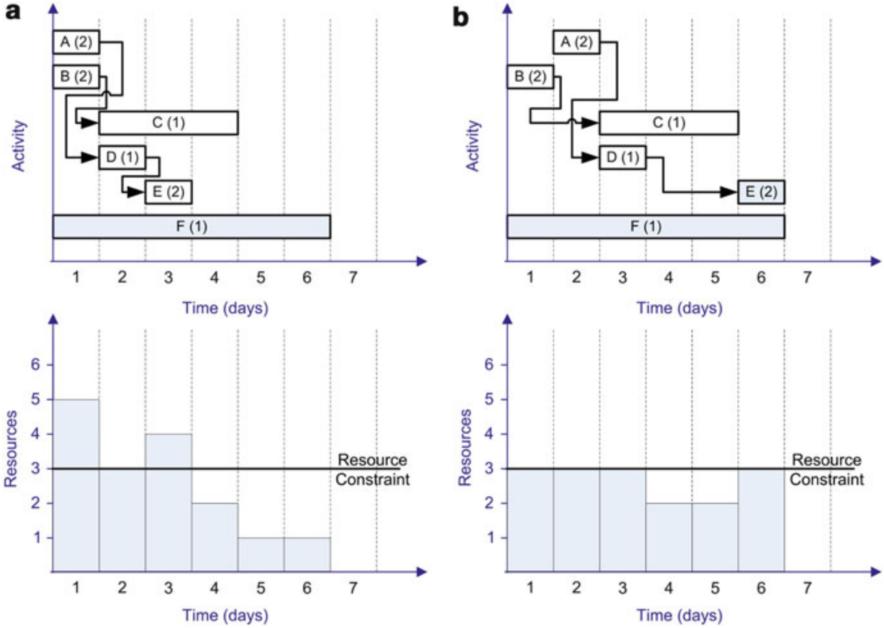


Fig. 18.5 Example 2—(**a**) Original (*unleveled*) Gantt chart and daily resource histogram; (**b**) Revised (*leveled*) Gantt chart and daily resource histogram

1.667, 1.721, 1.721, and 1.775 respectively. As expected, better resource-leveling is indeed accompanied by higher project entropy values.

18.8 Conclusions

This chapter discusses RLP and presents a newly-developed resource-leveling metaheuristic based on entropy and its two principal mathematical properties of subadditivity and maximality. The proposed metaheuristic and associated algorithm use an entropy-maximization approach to restate resource leveling as a process of maximizing the entropy found in a project's resource histogram. The entropy-maximization method performs very well both as a Monte Carlo simulation approach and as a metaheuristic algorithm. Furthermore, it can account for multiple resource types, for both duration-constrained and duration-unconstrained projects, as well as for activity stretching and activity crunching. Future work entails multi-resource leveling and improvements on the efficiency of the proposed algorithm.

References

Adeli H, Park HS (1995) Optimization of space structures by neural dynamics. Neural Netw 8(5):769–781

Ballestín F, Schwindt C, Zimmermann J (2007) Resource leveling in make-to-order production: modeling and heuristic solution method. Int J Oper Res 4(1):50–62

Brinkmann K, Neumann K (1996) Heuristic procedures for resource-constrained project scheduling with minimal and maximal time lags: the resource-levelling and minimum project-duration problems. J Decis Syst 5:129–156

Brucker P, Knust S, Schoo A, Thiel O (1998) A branch and bound algorithm for the resource-constrained project scheduling problem. Eur J Oper Res 107(2):272–288

Burgess AR, Killebrew JB (1962) Variation in activity level on a cyclic arrow diagram. J Ind Eng 13:76–83

Christodoulou SE, Ellinas G (2010) Scheduling resource-constrained projects with ant colony optimization artificial agents. J Comput Civil Eng 24(1):45–55

Christodoulou SE, Ellinas G, Aslani P (2009a) Disorder considerations in resource-constrained scheduling. Constr Manag Econ 27(3):229–240

Christodoulou SE, Ellinas G, Aslani P (2009b) Entropy-based scheduling of resource-constrained construction projects. Automat Constr 18(7):919–928

Christodoulou SE, Ellinas G, Michaelidou-Kamenou A (2010) Minimum moment method for resource leveling using entropy maximization. J Constr Eng M ASCE 136(5):518–527

Garmsiri M, Abassi MR (2012) Resource leveling scheduling by an ant colony-based model. J Ind Eng Int 8(7)

Harris RB (1978) Precedence and arrow networking techniques for construction. Wiley, New York

Harris RB (1990) Packing method for resource leveling (Pack). J Constr Eng M ASCE 116(2):331–350

Hegazy T (1999) Optimization of resource allocation and leveling using genetic algorithms. J Constr Eng M ASCE 125(3):167–175

Hegazy T, Shabeeb AK, Elbeltagi E, Cheema T (2000) Algorithm for scheduling with multiskilled constrained resources. J Constr Eng M ASCE 126(6):414–421

Hiyassat MAS (2000) Modification of minimum moment approach in resource leveling. J Constr Eng M ASCE 126(4):278–284

Hiyassat MAS (2001) Applying modified minimum moment method to multiple resource leveling. J Constr Eng M ASCE 127(3):192–198

Martinez J, Ioannou P (1993) Resource leveling based on the modified moment heuristic. In: Proceedings of 5th international conference on civil and building engineering, Anaheim, pp 287–294

Neumann K, Zimmermann J (1999) Resource leveling for projects with schedule-dependent time windows. Eur J Oper Res 117(3):714–729

Neumann K, Zimmermann J (2000) Procedures for resource leveling and net present value problems in project scheduling with general temporal and resource constraints. Eur J Oper Res 127(2):425–443

Ranjbar M (2013) A path-relinking metaheuristic for the resource levelling problem. J Oper Res Soc 64:1071–1078

Rubinstein RY, Kroese DP (2004) The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation, and machine learning. Springer, New York

Seibert JE, Evans GW (1991) Time-constrained resource leveling. J Constr Eng M ASCE 117(3):503–520

Senouci AB, Adeli H (2001) Resource scheduling using neural dynamics model of Adeli and Park. J Constr Eng M ASCE 127(1):28–34

Takamoto M, Yamada N, Kobayashi Y, Nonaka H (1995) Zero-one quadratic programming algorithm for resource leveling of manufacturing process schedules. Syst Comput Jpn 26(10):68–76

Zhang H, Li X, Li H, Huang F (2005) Particle swarm optimization-based schemes for resource-constrained project scheduling. Automat Constr 14(3):393–404

Part VI
Multi-Criteria Objectives in Project Scheduling

Chapter 19
Theoretical and Practical Fundamentals

Francisco Ballestín and Rosa Blanco

Abstract Project managers carry out a project with several objectives in mind. They want to finish the project as soon as possible, with the minimum cost, the maximum quality, etc. This chapter studies project scheduling problems when several goals are sought, that is, multi-objective project scheduling problems (MOPSPs) and multi-objective resource-constrained project scheduling problems (MORCPSPs). We will discuss some of the most important issues that have to be taken into account when dealing with these problems. We will also prove some useful results that can help researchers create algorithms for some of these problems.

Keywords Heuristic algorithms • Multi-criteria objective • Project scheduling • Resource constraints

19.1 Introduction

In most of the previous chapters the defined optimization problems included one objective function. The difficulty of these problems and the extensive number of interesting generalizations make this field almost endless. However, project scheduling is an inherently multi-objective problem. If academic research wants to get closer to practice we must be able to combine several goals in our project scheduling problems in one way or another. In this chapter we study multi-objective resource-constrained project scheduling problems (MORCPSPs), supposing we are working with a single mode for each activity and precedence relationships. We will make a distinction between problems where all the performance measures are regular and problems with at least one non-regular objective function. We recall that a regular objective function (ROF) is a non-decreasing function of the activity

F. Ballestín (✉)
Department of Mathematics for the Economy, University of Valencia, Valencia, Spain
e-mail: francisco.ballestin@uv.es

R. Blanco
Department of Statistics and Operations Research, Public University of Navarra, Pamplona, Spain
e-mail: rosa.blanco@navarra.es

C. Schwindt, J. Zimmermann (eds.), *Handbook on Project Management and Scheduling Vol. 1*, International Handbooks on Information Systems,
DOI 10.1007/978-3-319-05443-8_19

start times (in the case of a minimization problem), see Chap. 1 of this handbook. The best known case of a ROF is the project completion or makespan. Many papers in project scheduling are dedicated to this objective function: theoretical results, optimal and heuristic algorithms with different codifications, schedule-generation schemes, etc., have been developed throughout the years. It is consequently sound to start solving MORCPSP by solving MORCPSP with ROFs. However, the real power of MORCPSP lies in combining ROFs with non-regular objective functions (NROFs).

The chapter is organized as follows. The following section introduces key concepts used in the field of evolutionary multi-objective optimization, and establishes the problems that will be studied. Section 19.3 classifies relevant and recent papers of the literature and discovers an important weakness in most of them. Several important results in the fields of MOPSP and MORCPSP are presented in Sect. 19.4. Section 19.5 discusses some of the hot topics to cover when developing a heuristic algorithm for a MORCPSP. The final section is reserved for future lines of research.

19.2 Multi-Objective Optimization and MORCPSPs

A multi-objective optimization problem (MOP) can be defined as follows:

$$\begin{aligned}&\text{Min. } y = f(x) = [f_1(x), f_2(x), \ldots, f_\mu(x)]\\&\text{s. t. } \; x \in X^{feas},\end{aligned}$$

where $x = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$ is a vector of decision variables called decision vector, y is the objective vector, X^{feas} is the set of feasible solutions and $Y = f(X^{feas}) \subseteq \mathbb{R}^\mu$ is the image of the feasible set in the criterion space. A multi-objective combinatorial optimization problem (MOCO) is a MOP with a restriction of integrality on the decision variables.

While it is quite obvious what it means to solve a (single-objective) optimization problem, this is not so clear in the case of a MOP. There are several ways to tackle a MOP, depending on the importance we assign to the objective functions or the information we have about them. Hwang and Masud (1979) and Horn (1997) defined three categories, which depend on how the decision maker (DM) relates to the optimization process:

1. Decision making before search: Information given by the DM leads to a single objective optimization problem where the objective is an often linear function of the original objectives of the MOP.
2. Search before decision making: A set of candidate solutions (ideally the Pareto front or an approximation of it) is calculated and then the DM selects the solution among them.

3. Decision making during search: The optimization is divided into steps, after each of which a number of alternative trade-offs are presented to the DM, whose information guides the next step.

Some advantages and disadvantages of these approaches can be seen in Zitzler (1999). This chapter is devoted to the second class of this classification. However, many issues discussed in the text apply to the other two classes.

In the following paragraph we will follow Ehrgott and Gandibleux (2004) and Hansen (1979). A feasible solution $x \in X^{feas}$ is called efficient if there is no solution strictly better than x for at least one criterion and not worse in the remaining criteria. The image $y = f(x)$ of an efficient solution is called non-dominated. The set of efficient solutions is X_E^{feas}, the set of non-dominated vectors is $\mathscr{P}\mathscr{F} = f(X_E^{feas})$. The set of all non-dominated solutions is also called the Pareto set or Pareto front. Two efficient solutions x, x' are equivalent if $f(x) = f(x')$. A (minimal) complete set contains (only) one decision vector for every non-dominated objective vector. A maximal complete set contains all equivalent solutions for every non-dominated objective vector. When describing an exact or heuristic algorithm for a MOCO problem, it is important to clarify whether every non-dominated objective vector is attainable and whether a minimal or the maximal complete set is sought. In general, nevertheless, and as a generalization of single-objective optimization problems, a MOCO problem is considered as solved if a minimal complete set is calculated. That is, the Pareto front should be found, even if only one representative for each non-dominated objective vector is obtained. In the first category discussed above to solve a MOCO ("Decision making before search"), there are efficient solutions that are not optimal for any weighted sum of the objective functions. They are termed unsupported efficient solutions D_{NE}. The solutions which are optimal for some weighted sum problem are termed supported efficient solutions D_{SE}.

Regarding the difficulty of MOCOs, the problems are in general $\mathscr{N}\mathscr{P}$- and $\#\mathscr{P}$-complete, respectively, even if there are efficient algorithms in the single objective case. We refer to Ehrgott (2000), Emelichev and Perepelitsa (1991), and Serafini. (1986) for results in this respect; and to Garey and Johnson (1979) and Valiant and Vazirani (1986) for introductions to the theory of $\mathscr{N}\mathscr{P}$-completeness and $\#\mathscr{P}$completeness. As a consequence, metaheuristic algorithms are often developed to heuristically solve the problems, sometimes termed MOMHs (multi-objective metaheuristic algorithms).

MORCPSPs inherit many aspects from MOCOs, but there are also some relevant specific features to take into account. One of their most important characteristics is that a solution is always a schedule, a plan that explains when each activity is scheduled. This usually means defining a start time for each activity, supposing non-preemption. We may need though additional information to completely define a solution depending on the problem we are dealing with: the mode the activity is scheduled, how many resources are bought or rented, etc. The set of feasible solutions $\mathscr{S}$ of a MORCPSP is determined by precedence relationships between activities and resource restrictions. We can have therefore all the richness found in resource-constrained PSPs—minimum and maximum time lags, non-renewable resource constraints, multi-mode, etc.—or we can work with the basic restrictions of the RCPSP. Independently of $\mathscr{S}$, there are many objective functions of interest for managers. They usually have to do with time issues, managing the resources (usually how many to use or how to level their use), cost (associated to activities, to resources or to both), quality or stability. Here is a list with some of the most used ones (see the formal definitions in Ballestín and Blanco 2011). These objective functions should be minimised, unless stated otherwise: (1) makespan or project completion; (2) maximum tardiness; (3) total number of tardy activities; (4) sum of start times; (5) total weighted tardiness; (6) total weighted flow time; (7) net present value (maximise); (8) weighted earliness-tardiness; (9) resource levelling; (10) resource investment: RI (also known as resource availability cost RAC); (11) weighted sum of start times (maximise); (12) stability/robustness (maximise). In this last problem, the duration of an activity is a random variable instead of a known constant.

The average is sometimes considered instead of the total (weighted) tardiness and the total (weighted) flow time. On other occasions, the lateness is considered instead of the tardiness. Functions (9) and (10) are examples of resource levelling functions, other examples of these types of functions are described in Neumann et al. (2003). In these functions, the resource restrictions are sometimes not considered, because resources are already taken into account in the objective function. Functions (1)–(4) are ROFs, (5), (6), and—NPVare ROFs only if the weights (cash flows) are non-negative. We say that an ROF is strict, or that we work with a strict ROF, if the function is an increasing function of the activity start times. Function (4) is the classical example of a strict ROF. The remaining functions are usually non-regular.

We obtain a different MOPSP or MORCPSP for each combination of objective functions and restrictions. Obviously, there are thousands of them. Nonetheless, we think that some combinations of objective functions make more sense or are more interesting than others. Another crucial issue is whether each goal in the problem is included in the optimization problem as an objective function. Sometimes some goals should be included as restrictions, and sometimes as an objective function and as a restriction at the same time.

19.3 Classifications of Papers of the Literature

Once we have established some definitions, we can briefly classify and comment on the most important papers that have researched MORCPSPs or similar problems. Additional information can be obtained in Ballestín and Blanco (2011).

Slowinski (1981) was the first author to explicitly place the RCPSP in a multi-objective framework. His approach belongs to the first class of the classification, because he proposes applying parametric linear programming. Note that this procedure can only calculate supported efficient solutions. Slowinski also discusses the possibilities of using goal programming and fuzzy linear programming. This author, sometimes in collaboration with others, published several papers on

MORCPSPs (Slowinski 1989, Slowinski and Węglarz 1985, Slowinski et al. 1994). The procedures presented in these papers are interactive procedures that pertain to the third class of the classification. In Slowinski et al. (1994), a decision support system is presented. The fuzzy versions of MORCPSP have also been studied (see e.g. Hapke et al. 1997, Hapke and Slowinski 2000, and Pan and Yeh 2003). Pareto simulated annealing (PSA) is used in most of the papers for these problems. Hapke et al. (1998) presented a two-stage interactive search procedure. In the first stage an approximation of the efficient set is calculated and then an interactive search over the sample is organised. This second part belongs therefore to the third class of the classification discussed above. The definition of employed techniques seem to indicate that only active schedules can be calculated, something which is not enough for obtaining the Pareto front in the case of non-regular measures (see Sect. 19.4). Nabrzyski and Węglarz (1995, 1999) also developed procedures for a similar MORCPSP as in Hapke et al. (1998). In the first paper, a decision support system is described. From among the generated feasible schedules, non-dominated schedules are found and an interactive procedure with the DM is organised on this set. The definition of the algorithm, as stated in the paper, seems not to be able in general to find all the efficient solutions. In the second paper, an approach belonging to the third class described above is described. Viana and de Sousa (2000) implemented MOMHs for regular and non-regular objective functions. From the described neighbourhood, it seems very unlikely that the procedures can obtain solutions with good values in some of the used performance measures in medium or large instances.

There are also papers in the literature looking for trade-off curves between two objectives, which lie therefore between single- and multiobjective (see for example Schwindt and Zimmermann 2002).

Al-Fawzan and Haouari (2005) and Abbasi et al. (2006) have studied the MORCPSP with renewable resources and two objectives, makespan and robustness, modelled as the sum of the free slack of activities that should be maximised. A multi-objective Tabu Search is applied to calculate an approximation of the efficient set. Activity lists and the serial schedule-generation scheme (SGS) are used in the algorithm (see Chap. 1 of this handbook). A forward-backward recursion procedure helps to compute the latest completion times for the activities and hence their free slacks. The authors state that their MOTS algorithm consists in running a single-objective TS a certain number of times, each time with a different linear aggregated function. Due to the SGS used, the procedure can obtain good solutions in terms of makespan, but will probably miss all efficient solutions with (very) good robustness and bad makespan. Therefore, a whole part of the Pareto front cannot be reached with the proposed approach. Abbasi et al. (2006) work with only one renewable resource and aggregate the two objectives in a linear objective. Therefore, their procedure belongs to the first class of the classification described above. The authors apply a SA with a forward-backward recursion procedure. To generate different solutions, some virtual constraints are added to the problem. Valls et al. (2009) worked with the skilled workforce project scheduling problem, a complex problem of task scheduling and resource assignment that comes up in the daily management of many company's Service Centres. The solution methodology used was to lexicographically minimise firstly the infeasibility of the total violation of the temporal constraints, secondly a linear combination of maximum dates and worker availability violations, and finally three other secondary objectives. Odedario and Oladokun (2011) wrote about the relevance and applicability of MORCPSP, and concluded that single objective approaches in RCPSP are generally idealistic. Xiong et al. (2012) developed a MOMH for the RCPSP with perturbation on activity durations, with the makespan, robustness and stability as objectives. They also use activity lists and the serial SGS to calculate schedules. Yannibelli and Amandib (2013) developed a MOMH for a MORCPSP that minimises the makespan and optimises the effectiveness of human resources. The codification employed also contains activity lists, and the serial SGS decodes them, too. However, since the two functions are regular, these methods are searching in the right solution space (see Sect. 19.4). Nevertheless, the authors do not include any discussion about how or why the attainable schedules are appropriate for their problem.

As a conclusion of the review, there are many papers that seem not to be able to calculate a good approximation of the Pareto front. An essential aspect of an algorithm for a MORCPSP is that it is searching in the right solution space, i.e., there is no non-dominated decision vector which can never be found. A paper should prove that the described algorithms are indeed using the right type of schedules. This aspect is, however, almost never discussed in papers that cover MORCPSPs.

19.4 Theoretical Results for the MOPSP and MORCPSP

As stated in Sect. 19.2, the number of different MOPSPs and MORCPSPs is very big. Our experience with these problems says that a MORCPSP becomes easily intractable, and that in most cases heuristics algorithms will be needed to offer efficient solutions for instances with a moderate size. Nevertheless, exact algorithms should also be developed to allow us learn about MORCPSPs, and to evaluate heuristic algorithms. Moreover, theoretical results should be developed, too, so that heuristic and exact algorithms build the correct schedules (candidates for optimal schedules, see Neumann et al. 2003) that guarantee the obtaining of a minimal complete set. In the following subsections we present results for MOPSPs and MORCPSPs with ROFs, and for some MO(RC)PSPs with NROFs, especially bi-objective problems with the makespan as one of the objective functions. Proofs for those results can be found in Ballestín and Blanco (2011). We are going to suppose we work with the restrictions of the RCPSP, i.e., non-preemption, precedence relationships and renewable resource restrictions. This assumption implies for example that the start of an activity determines its finish time. Generalizations can be sometimes made to general precedence relationships and non-renewable resources.

One important aspect to take into account when solving MORCPSPs is that relationships between activities define time windows for activities. If we have a partial schedule where we have scheduled only some of the activities, the start time of the next activity to be scheduled is limited to an interval defined by its predecessors and successors. This fact is true for each objective value, and the time interval is the same for each objective function, since it is based on the restrictions. The objective function determines at which time inside that time window is (locally) optimal to schedule the activity. For instance, the best time for a ROF is the first instant of that time window. In a MORCPSP different objective functions will usually prefer different instants inside the time windows, and there lies the difficulty of the problems.

19.4.1 MOPSP with ROFs

As stated in Chap. 1 of this handbook, the ES schedule is the schedule where every activity is scheduled at its earliest start time according to the precedence relationships. The ES schedule forms a minimal complete set in any MOP with ROFs. Besides, the ES schedule is the only efficient solution in any MOP with ROFs with at least one strict ROF.

19.4.2 MORCPSP with ROFs

The most important problem studied in resource-constrained project scheduling is the RCPSP. Therefore it is natural to study MORCPSP with ROFs and to develop exact and metaheuristic algorithms for these problems. As commented in Chap. 1 of this handbook, the serial SGS generates so-called active schedules, where each activity is scheduled as early as possible. There is always an active schedule that is optimal in the single-objective RCPSP with any ROF (see Neumann et al. 2003). This fact leads to the result that the set of active schedules AS contains a complete set of efficient solutions in any MORCPSP with ROFs. Moreover, the set of active schedules AS contains a maximal complete set of efficient solutions in any MORCPSP with ROFs and at least one strict ROF. Obviously, as it happens in the classical RCPSP with the makespan, it is not enough in general to work with active schedules to obtain a maximal complete set of efficient solutions if the objective functions are regular but not strictly regular. For instance, let us consider a bi-objective RCPSP with the makespan and number of tardy activities, where the due date of each activity is greater than the optimal project length. In this problem, any non-active schedule with makespan equal to the optimal project length is efficient.

Based on these results we can devise procedures to calculate, optimally or heuristically, a minimal complete set for any MORCPSP with ROFS, or a maximal complete set in the case where at least one objective function is strict regular. Namely, any method that explicitly or implicitly explores all active schedules assures the exact solution of those problems. Besides, any heuristic algorithm that works with activity lists and the serial SGS is searching in the right solution space.

In principle, the algorithm is able to find any efficient solution. A good MOMH with these ingredients will therefore find a good approximation of a complete set in a MORCPSP with ROFs.

19.4.3 MORCPSP with at Least One NROFs

One of the most appealing aspects of working with MORCPSPs for a project manager is the possibility of combining the makespan with a non-regular objective function. All or almost all papers of the literature that work with MORCPSPs enter in that description. However, it seems that in some of those papers active schedules are used. The first fact to show is therefore that working with those schedules is not enough or advisable. Actually, it is easy to find a counterexample. Let us consider a project with two non-dummy activities 1 and 2, with a precedence relationship between them. They do not require resources and have a duration of 1. There is a due date of 3 for activity 2, with an associated earliness cost of 1. The only active schedule of this problem starts activities 1 and 2 at 0 and 1, respectively. This schedule has an optimal makespan of 2 and a cost of 1. There are two more efficient solutions in the problem. Both schedules starts activity 2 at 2, one solution starts activity 1 at 0 and the other starts activity 1 at 1. Both solutions have a makespan of 3 and a cost of 0. Summing up, if we work with NROFs active schedules are usually not going to be enough to find a representative for every non-dominated vector.

Hence, we have to find other types of schedules to use in those MORCPSPs. Combining ROFs and NROFs clearly makes more difficult the job to do. Nevertheless, there are several results that can be proven. Before stating them, we need to recall a result of Neumann et al. (2003). They introduced and gather together different types of functions, like antiregular, convex, binary-monotone, quasiconcave, locally regular and locally quasiconcave functions. For each of these functions they find the schedule type needed to solve the corresponding single-objective problem. They proved that to obtain an optimal solution for objective functions of the antiregular, convex, binary-monotone, quasiconcave, locally regular and locally quasiconcave types it is enough to work with antiactive, locally order-optimal, pseudostable, stable, quasiactive and quasistable schedules, respectively.

The objective functions of the WET, RI, RL and NPV problems are convex, locally regular, locally quasiconcave and binary-monotone, respectively. To ease the following discussion about all these measures, we will denote a certain type of objective function with χ. We will denote $\mathscr{S}_\chi$ the set of schedules always containing an optimal solution for an objective function of type χ. Depending on the function type, this set depends on the deadline $\overline{d}$ given for the project. For a given $\overline{d}$ we denote $\mathscr{S}_\chi(\overline{d})$ the set of schedules $\mathscr{S}_\chi$ corresponding to that deadline.

Results

1. The set of non-dominated solutions $\mathscr{PF}$ is finite if there is a deadline for the makespan $\overline{d}$.
2. The set of non-dominated solutions $\mathscr{PF}$ is finite if each of the involved objective functions is regular or belongs to the set {WET, RL, RI}.
3. We consider the bi-objective MORCPSP with the makespan and an objective function of the type χ and with a fixed global deadline $\overline{D}$. Then, the set $\cup_{\overline{d}=1}^{\overline{D}} \mathscr{S}_{\chi}(\overline{d})$ contains a minimal complete set for that problem.

The proofs can be seen in Ballestín and Blanco (2011).

The previous results are not true in the general case. Let us see counterexamples for the first two. Let us consider a project with two non-dummy activities of duration one, with a relationship between them and without resource requirements. Suppose the objective functions are the makespan and the NPV. The weight (cash flows) of activities 1 and 2 are 0 and -1, respectively, the parameter α is 1. So, the NPV function is $-e^{-C_2}$, with C_2 the finish time of activity 2, and we want to maximise it. That is, we want to minimise e^{-C_2}. The finish time of activity 2 marks the makespan and NPV of the schedule. If there is a deadline $\overline{d}$ there are $\overline{d} - 1$ non-dominated vectors $\mathscr{PF} = \{(2, e^{-2}), (3, e^{-3}), \ldots, (\overline{d}, e^{-\overline{d}})\}$. Therefore we have an infinite number of non-dominate vectors and efficient solutions if there is no deadline.

The third result offers a method to optimally solve the bi-objective MORCPSP with the makespan and objective functions of the type χ, to (explicitly or implicitly) enumerate all the schedules $\mathscr{S}_{\chi}(\overline{d})$, but with different deadlines. The result also helps in devising heuristic algorithms for these problems: using the SGS developed in each case to produce schedules of the right type. There are very efficient procedures capable of obtaining the optimal WET or NPV value for a given deadline (see e.g. Neumann et al. 2003). Just by applying these procedures to all deadlines between the makespan of the ES schedule (the critical path length) and the global deadline we obtain a set H containing the set of non-dominated solutions.

19.5 Topics to Study in a MOMH for MORCPSPs

Even though there are many possible MORCPSPs, there are only so many common topics that can be studied in MOMHs for those problems, apart from those related to the specific functions or restrictions defined in the problem. The following does not pretend to be an exhaustive list, but gives some ideas as to what to investigate in a MOMH for a MORCPS.

- Codification and SGS
  A very important decision in a heuristic algorithm, even in single-objective optimization problems, is the way to represent a solution (codification), and the way to build a solution out of a codification (SGS). A sensible decision when dealing with a ROF is to work with activity lists and the Serial SGS. In

a MORCPSP with ROFS, as stated above, it is also enough to work with that codification and SGS. However, each different problem with at least one NROF will call for a different codification and SGS. To give an example, let us consider the MORCPSP with precedence relationships and two objective functions, makespan and resource investment. In the resource investment problem, each resource has a cost associated with it, and it has to be decided how many units of each resource to have. Each unit of each resource implies a cost, and the total resource cost should be minimized. These two functions are negatively correlated, therefore we will obtain (the same or) a best solution with smaller makespan if the availability of each resource is higher. If there are n non-dummy activities and K resources, we can codify an individual with two lists, an activity list and a list of available resource capacities, $I = (\ell, R)$, with $R = (R_1, R_2, \ldots, R_K)$. To obtain a solution we schedule the activities according to the activity list ℓ and the serial SGS, but with the resource availabilities given by R. It can be proved (see Ballestín and Blanco 2011) that if we tried every duple (ℓ, R) we would find at least one efficient solution for each non-dominated vector.

Other codifications and SGSs will have to be found for other MORCPSPs, especially for bi-objective problems. We think that problems with several NROFS will not have easy codifications or SGS capable of calculating every efficient solution. Depending on the combination of objectives, a heuristic algorithm may have to work with a vector of start time for each activity, i.e., to work with the schedule itself.

- Metaheuristic algorithm to use
  Many papers propose general metaheuristic algorithms for MOCOs. Some of the most used are NSGA-II (Deb et al. 2000), SPEA2 (Zitzler et al. 2001), PSA (Czyzak and Jaszkiewicz 1998), or MOEA/D (Zhang and Li 2007). Many more procedures have been developed, based on Tabu Search, Ant Colony Optimization, Particle Swarm Optimization, Artificial Immune System, etc. One of the tasks of a researcher in MORCPSP will be to decide which one of the existing frameworks to use for a specific problem, or to develop an ad-hoc procedure. The suitability of these procedures, the performance differences among them, etc., have to be reassessed in the context of MORCPSPs. Even more than the metaheuristic themselves, researchers should study which tools or procedures from the vast multi-objective optimization literature have a great impact in MORCPSPs. Some of these tools are commented on in the following points.
- Improvement Operators for each objective function
  Our hypothesis is that MORCPSPs with several NROFS will not have easy codifications or SGS capable of calculating a representative for each non-dominated vector. In those cases, a MOMH can be built in the following manner. The defined codification and SGS provides (good-quality) solutions, which are further improved by improvement operators. The use of these operators applied to solutions obtained from the SGS could help reach the parts of the Pareto front

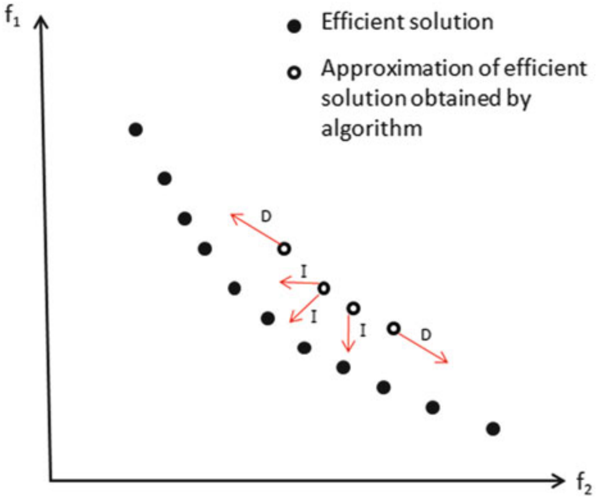


Fig. 19.1 Intensification and diversification operators

not reachable with the codification + SGS. Part of the research would then be to find the best combination codification + SGS + improvement operators.

- Diversification v. Intensification
  How to balance diversification and diversification has always been an issue in heuristic algorithms. Nevertheless, it is even more important in multi-objective optimization. Both concepts can be easily interpreted in terms of the non-dominated decision vectors. Figure 19.1 gives examples of the direction of typical intensification operators (I arrows) and diversification operators (D arrows). On the one hand, diversification comprises the techniques that try to calculate non-dominated decision vectors "quite different" from the (approximation of the) non-dominated decision vectors the algorithm has calculated until that moment. The most important goal of diversification in a MOMH should be that there is no part of the Pareto front without a representative in the outcome set, the set of solutions provided by the algorithm. An example of a diversification operator would seek to obtain new best values in one (or a couple of) objective function(s), even at the expense of significantly worsening the rest of the objectives. We consider again the bi-objective problem defined in the first point of this subsection, the MORCPSP with precedence relationships and two objective functions, the makespan and resource investment. At a given moment of a heuristic algorithm, the algorithm applies a diversification operator to the schedule S obtained from (ℓ, R) with the minimum cost we have obtained so far. The algorithm calculates a new list of available resource capacities R' by reducing one of the resource levels by one unit. If we calculate a random sampling centered in (ℓ, R'), all the solutions obtained will have a new best value in the cost function, although they may be worse in terms of the makespan.

On the other hand, intensification would try to carry the search closer to the Pareto front. Typically it could be applied to a non-efficient solution S obtained by the algorithm which is an approximation of one or several efficient solutions of the Pareto front—because of proximity of decision vectors. An intensification operator would start with S and try to obtain solutions closer to those efficient solutions. The goal would therefore be to obtain an improvement in one or several objectives, maintaining the remaining objectives without worsening or with a slight deterioration. In the same bi-objective problem defined above, we can fix the resource levels of a solution at hand and apply the forward backward technique. This technique may improve the makespan and will not worsen the cost function.

- Justification/forward-backward technique
  The basic forward-backward technique is defined in Chap. 4 of this handbook. However, the idea of justification is more general. Basically, justification consists in shifting activities within their time windows—i.e., fulfilling time constraints—and therefore without worsening the makespan. The forward-backward technique just shifts every activity as late as possible and then as soon as possible. The gaps created with those movements are used by other activities and sometimes the makespan is reduced. However, shifting one activity without worsening the makespan is a perfect movement for bi-objective problems with the makespan and a NROF like resource leveling, NPV or WET. We believe that this technique or modifications of it can be used in many MORCPSPs, also because justification can take advantage of (or be tailored to cope with) modes, due dates, non-renewable resources, etc.
- Including information in the initial population
  In some works for MOPs it is discussed whether to start a MOMH out of a population of random solutions or to use knowledge of the objective functions to start with better solutions. Our experience with RCPSPs and MORCPSPs clearly favors the use of information. However, in that case it has to be decided how to do it. For example, every objective function should be taken into account. The whole search might get biased if the population focuses on some objectives and disregards the remaining goals. A whole part of the Pareto front may even not be easily reached during the search.
- Management of the population: elitism, choosing among similar solutions, etc.
  The handling of a population is very different in a MOMH than in a heuristic for a single-objective problem. The reason for this is that we are looking for a set of solutions. One could store all currently efficient solutions, those solutions that are not improved by other so far calculated solutions. However, sometimes there are many of these solutions. Some researchers keep a separate set (sometimes called an archive) for those solutions, but it should be decided which of these currently efficient solutions to keep in the population and which ones to erase, even from the archive. In this decision, the similarity of solutions—either the similarity of schedules or of objective vectors—should also be taken into account. If too many similar solutions are kept, it is much more difficult to lead the search to different regions of the Pareto front. Independently of how many and

where to store the efficient solutions, the researcher should decide how these best individuals—perhaps not calculated in the last iterations—influence the creation of the next generation. This issue is termed elitism. We are convinced that a certain degree of elitism is beneficial: best solutions should be used to create the next population. A different thing is how much randomness or information from the last iteration should be included in that calculation. There are other issues concerning the management of the population to take into account, for instance, how to select among non-efficient solutions when there are very few efficient solutions, or when it is necessary to refresh the population, and in that case how to do it.

- Measures to express the quality of a heuristic algorithm
  The measurement of the performance of a heuristic algorithm in a single-objective optimization problem on a given instance is easy: algorithms can always be ranked according to their value in the objective function. To compare algorithms regarding their performance on a set of instances is more difficult. For instance, when the objective function is the total tardiness, depending on the tightness of the due dates there may be instances with an optimal value of 0 and others with an optimal value of thousands. It is not straightforward how to calculate an indicator of the algorithm quality using both instances. However, in most cases the commonly accepted measure is the average deviation with respect to a lower bound or with respect to the best obtained solutions—when enough different algorithms have been tested.
  When it comes to MOPs the problem of measuring the performance of an algorithm reaches a whole new level. In general, it is not even clear how to decide which of the two algorithms is the best, even with just one instance. Several metrics have been published to measure the quality of heuristics (see e.g. Zitzler et al. 2003). The hypervolume indicator has become a very popular metric for comparing approximation sets in the last years. The hypervolume indicator measures the area between the solutions of the Pareto approximation set and the 'reference point'. In Ballestín and Blanco (2011) some disadvantages for MORCPSPs of some measures are discussed: problems with scalability, sensitiveness to the size of the outcome sets, etc. There is no clear solution to determine which the best measure is, but some measures should not be used alone, for instance, the size of the approximation set or the maximum spread. The former just counts the number of solutions in the approximation set, while the latter measures the diversity of the approximation set. Both measures can be clearly misleading, assigning better values to algorithms clearly worse than others. A good measure in some cases for MORCPSPs is the distance from the reference set, cf. Czyzak and Jaszkiewicz (1998). The reference set *RefSet* is the set union of sets obtained by different algorithms. It is the generalisation to multi-objective problems of the best known solution for an instance. The distance between outcome set M and reference set *RefSet* calculates the distance between the outcome of an algorithm and this set. This distance can be defined as:

$$\Delta = \frac{1}{|RefSet|}\sum\nolimits_{y\in RefSet}\min_{x\in M}\{c\,(x,y)\},\ \text{where}\ c\,(x,y) = \max_{\mu=1,\ldots,v}\left|w_\mu\cdot\left(f_\mu\,(x) - f_\mu\,(y)\right)\right|,\ \text{with}\ x\in M,\ y\in RefSet,\ w_\mu = 1/\delta_\mu.$$

Due to the fact that the proximity to the Pareto optimal front is measured, smaller values are preferable. The weights and values δ_k are used to standardize all objective values. It is not easy to determine those δ_k. We propose to use the objective function of a feasible solution, for example one given by an established priority rule. We also propose to add one to the numerator of δ_k, since, for example, for the total tardiness the value of δ_k can be 0. This measure has also an important disadvantage, as it needs a good reference set. This means that, if a new MORCPSP is considered, several different methods should be implemented, so that a good reference set can be calculated. Random solutions, priority rules, random biased samplings and even other metaheuristic algorithm should be run to make sure that the developed MOMH has enough good competitors. The MOMH itself, but with much more time available, can also help build a better reference set.

- Computational tests

  Needless to say, computational tests are essential in the development of research. It is therefore necessary to develop benchmark instances for the most important MORCPSPs. These instances should help identify which MOMHs outperform better, and which components of a MOMH really contribute to the performance of the algorithm.

  In Ballestín and Blanco (2011) we worked with several MOMHs for different MORCPSPs with ROFS and a MORCPSP with one NROF. We worked with the standard j120 set for the RCPSP (Kolisch et al. 1995), which consists of 600 projects with 120 non-dummy activities. We introduced additional information for each activity and each resource to be able to work with functions such as the NPV (with positive cash flows), tardiness and RI. Different limits for the number of calculated solutions were imposed. We witnessed the following behaviour trends: the use of justification, the use of information in the initial population, and elitism, all of which helped find better solutions. Including the parallel SGS (see Chap. 1 of this handbook) did not improve the quality of algorithms. We run three metaheuristic algorithms, the NSGA-II, the SPEA2 and the PSA, and a sampling procedure to check the quality of the metaheuristic algorithms. Both SPEA2 and NSGA clearly outperformed the PSA and the sampling procedure. SPEA2 and NSGA2 offered a similar average quality. The size of the Pareto front was small or very small in the case of ROFs. This is a strange case in multi-objective optimization, caused by the positive correlation among the performance measures. This fact does not happen when NROFs are present. We observed in several cases how the Pareto fronts were more similar in those cases to what is expected in a multi-objective problem.

19.6 Conclusions

This chapter has established the general framework for working with MORCPSPs with ROFs. A very good method to develop MOMHs for these problems can be activity lists plus the serial SGS. However, many techniques and methods can still be evaluated in these problems, as the RCPSP has also been a testing bank for new metaheuristic procedures.

Nevertheless, the most important research field in MORCPSPs lies in combining ROFs with NROFs. The first steps should be to present quality algorithms for bi-objective problems that combine the makespan with important objective functions such as the WET, NPV, etc. Those algorithms should obviously search in the correct solution space, and proofs that any solution from the Pareto front can be attained must be offered. A second more difficult step could be to combine two negatively correlated NROFs, probably with the makespan. Here it would be interesting to see the possible development of new codifications and SGSs, intensification and diversification methods. In our opinion, MORCPSPs with many different ROFs and NROFs together will likely need algorithms that work with the schedules themselves. At the current stage of the research in MORCPSP, those problems may be more interesting if connected to a real problem rather than with pure academic research.

Finally, exact algorithms for the most important MORCPSPs cannot be forgotten. Their existence would help us learn the features of Pareto fronts in the different MORCPSPs, as well as compare the approximation sets obtained by heuristic algorithms.

Acknowledgements This research was partially supported by Ministerio de Ciencia e Innovación, MTM2011-23546.

References

Abbasi B, Shadrokh S, Arkat J (2006) Bi-objective resource-constrained project scheduling with robustness and makespan criteria. Appl Math Model 180:146–152

Al-Fawzan MA, Haouari M (2005) A bi-objective model for robust resource-constrained project scheduling. Int J Prod Econ 96:175–187

Ballestín F, Blanco R (2011) Theoretical and practical fundamentals for multi-objective optimisation in resource-constrained project scheduling problems. Comput Oper Res 38 (1):51–62

Czyzak P, Jaszkiewicz A (1998) Pareto simulated annealing: a metaheuristic technique for multiple-objective combinatorial optimization. J Multi-Criteria Decis Anal 7(1):34–47

Deb K, Agrawal S, Pratap A, Meyarivan T (2000) A fast elitist nondominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In: Schoenauer M et al. (eds) Parallel problem solving from nature (PPSN VI). Springer, London, pp 849–858

Ehrgott M (2000) Approximation algorithms for combinatorial multicriteria optimization problems. Int Trans Oper Res 7:5–31

Ehrgott M, Gandibleux X (2004) Approximative solution methods for multiobjective combinatorial optimization. TOP 12(1):1–88

Emelichev VA, Perepelitsa VA (1991) Complexity of vector optimization problems on graphs. Optimization 22:903–918

Garey MR, Johnson DS (1979) Computers and intractability – a guide to the theory of NP-completeness. Freeman, San Francisco

Hansen P (1979) Bicriterion path problems. In: Fandel G, Gal T (eds) Multiple criteria decision making theory and application. LNEMS, vol 177. Springer, Berlin, pp 109–127

Hapke M, Slowinski R (2000) Fuzzy set approach to multi-objective and multi-mode project scheduling under uncertainty. In: Slowinski R, Hapke M (eds) Scheduling under fuzziness. Physica, Heidelberg, pp 197–221

Hapke M, Jaszkiewicz A, Slowinski R (1997) Fuzzy project scheduling with multiple criteria. Fuzzy systems. In: Proceedings of the sixth IEEE international conference, Barcelona, pp 1277–1282

Hapke M, Jaszkiewicz A, Slowinski R (1998) Interactive analysis of multiple-criteria project scheduling problems. Eur J Oper Res 107:315–324

Horn J (1997) Multicriteria decision making. In: Back T, Fogel DB, Michalewicz Z (eds) Handbook of evolutionary computation, F1.9. IOP Publishing and Oxford University Press, Bristol, pp 1–15

Hwang CL, Masud AS (1979) Multi-objective decision making, methods and applications: a state of the art survey. LNEMS, vol 164. Springer, Berlin

Kolisch R, Sprecher A, Drexl A (1995) Characterization and generation of a general class of resource-constrained project scheduling problems. Manag Sci 41:1693–1703

Nabrzyski J, Węglarz J (1995) On an expert system with tabu search for multiobjective project scheduling. In: Proceedings of INRIA/IEEE symposium on emerging technologies and factory automation III, pp 87–94

Nabrzyski J, Węglarz J (1999) Knowledge-based multiobjective project scheduling problems. In: Węglarz J (ed) Project scheduling: recent models, algorithms and applications. Kluwer, Boston, pp 383–413

Neumann K, Schwindt C, Zimmermann J (2003) Project scheduling with time windows and scarce resources. Springer, Berlin

Odedario BO, Oladokun V (2011) Relevance and applicability of multi-objective resource constrained project scheduling problem. Eng Technol Appl Sci Res 1(6):144–150

Pan H, Yeh CH (2003) Ametaheuristic approach to fuzzy project scheduling. In: Palade V, Howlett RJ, Jain L (eds) KES 2003. LNCS, vol 2773. Springer, Heidelberg, pp 1081–1087

Schwindt C, Zimmermann J (2002) Parametrische Optimierung als Instrument zur Bewertung von Investitionsprojekten. Z Betriebswirt 72:593–617

Serafini P (1986) Some considerations about computational complexity for multi objective combinatorial problems. In: Jahn J, Krabs W (eds) Recent advances and historical development of vector optimization. LNEMS, vol 294. Springer, Berlin, pp 222–232

Słowiński R (1981) Multiobjective network scheduling with efficient use of renewable and nonrenewable resources. Eur J Oper Res 7:265–273

Słowiński R (1989) Multiobjective network scheduling under multiple-category resource constraints. In: Slowinski R, Węglarz J (eds) Advances in project scheduling. Elsevier, Amsterdam, pp 151–167

Słowiński R, Węglarz J (1985). An interactive algorithm for multiobjective precedence and resource constrained scheduling problems. In: Proceedings of the 8 World congress on project management INTERNET85. North-Holland, Amsterdam, pp 866–873

Slowiński R, Soniewiclu B, Węglarz J (1994) DSS for multiobjective project scheduling. Eur J Oper Res 79(2):220–229

Valiant LG, Vazirani VV (1986) NP is as easy as detecting unique solutions. Theor Comput Sci 47:85–93

Valls V, Pérez A, Quintanilla S (2009) Skilled workforce scheduling in service centres. Eur J Oper Res 193:791–804

Viana A, de Sousa JP (2000) Using metaheuristics in multiobjective resource constraints project scheduling. Eur J Oper Res 120:359–374

Xiong J, Chen Y-W, Yang K-W, Zhao Q-S, Xing, L-N (2012) A hybrid multiobjective genetic algorithm for robust resource-constrained project scheduling with stochastic durations. Math Probl Eng 2012. Article ID 786923, 24 pp.

Yannibellia V, Amandib A (2013) Project scheduling: a multi-objective evolutionary algorithm that optimizes the effectiveness of human resources and the project makespan. Eng Optim 45(1): 45–65

Zhang Q, Li H (2007) MOEA/D: a multi-objective evolutionary algorithm based on decomposition. IEEE Trans Evol Comput 11(6):712–731

Zitzler E (1999) Evolutionary algorithms for multiobjective optimization: methods and applications. Ph.D. dissertation, Swiss Federal Institute of Technology (ETH), Zurich

Zitzler E, Laumanns M, Thiele L (2001) Sped: improving the strength Pareto evolutionary algorithm. Technical Report 103, Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Switzerland

Zitzler E, Thiele L, Laumanns M, Fonseca C, Grunert da Fonseca V (2003) Performance assessment of multiobjective optimizers: an analysis and review. IEEE Trans Evol Comput 7(2):117–132

Chapter 20
Goal Programming for Multi-Objective Resource-Constrained Project Scheduling

Belaïd Aouni, Gilles d'Avignon, and Michel Gagnon

Abstract The aim of this chapter is to present a simplified formulation of multi-objective resource-constrained project scheduling problem based on a goal programming model. Three objectives will be explained and formulated in the context of project management, namely: the project duration, the project cost, and the quantity of the allocated resources. These objectives are incommensurable and conflicting. The proposed model will provide the baseline schedule of the best compromise based on the project manager's preference structure.

Keywords Baseline schedule • Goal programming • Manager's preferences • Multi-criteria • Project scheduling • Resource constraints

20.1 Introduction

The multi-criteria decision aid (MCDA) paradigm is driven by the need to deal with decision-making situations where several incommensurable and conflicting dimensions (attributes, criteria or objectives) are simultaneously optimized. The aggregation of different objectives requires from the decision-maker (DM) or the manager some compromises or tradeoffs. Hence the obtained solution is the one of

B. Aouni (✉)
Department of Management and Marketing, College of Business and Economics, Qatar University, Doha, Qatar
e-mail: belaid.aouni@qu.edu.qa

G. d'Avignon
Faculty of Business Administration, Laval University, Quebec, QC, Canada
e-mail: Gilles.DAvignon@fsa.ulaval.ca

M. Gagnon
AGL & Associates, 213 du Grand-Hunier, Saint-Augustin-de-Desmaures, QC, Canada
e-mail: Gagnon.Michel.J@gmail.com

C. Schwindt, J. Zimmermann (eds.), *Handbook on Project Management and Scheduling Vol. 1*, International Handbooks on Information Systems,
DOI 10.1007/978-3-319-05443-8_20

the best compromise. In fact, within the MCDA paradigm the DM pure rationality and the optimal solution concepts have been evolved to the notion of satisficing concept where the outcome of the decision-making process can be seen as a recommendation (Simon 1956, 1976; Lee 1973; Ignizio 1976; Aouni et al. 2009).

Some of the MCDA aggregation procedures are based on the distance function model (DFM) that minimizes the distance between the achievement and the aspiration levels of ν objectives as follows: Min. $\left(\sum_{\mu=1}^{\nu}\left(w_{\mu}\left|g_{\mu}-f_{\mu}\left(x\right)\right|\right)^{l}\right)^{1/l}$, subject to $x \in X \subset \mathbb{R}^{n}$, where $f_{\mu}\left(x\right)$ represent the μ-th achievement level of the alternatives, X designates the set of the feasible solutions, g_{μ} are the fixed goals for objectives μ, l is a parameter that defines the family of distance functions and w_{μ} are the relative importance of the objectives μ. The Goal Programming (GP) model is a special case of the DFM where the parameter $l = 1$ and all objective functions $f_{\mu}(x)$ are linear. The linear representation of the FDM minimizes the unwanted deviations between the aspirations levels g_{μ} and the achievement levels $f_{\mu}(x)$. The GP model has been introduced first by Charnes et al. (1955) as well as Charnes and Cooper (1961). This model has been widely applied in several fields (Tamiz et al. 1995, 1998, Aouni and Kettani 2001, Jones and Tamiz 2002, Caballero et al. 2006, Caballero and Hernandez 2006, Glover and Sueyoshi 2009).

One of the fields of the GP application is the resource-constrained project scheduling problem. Recently, Gagnon et al. (2012) developed a GP model that integrates explicitly, through the concept of satisfaction functions, the project manager's preferences for generating the baseline schedule of the best compromise among a set of conflicting project objectives, such as: the project duration, the project cost of different resource types, and the fluctuation of the required resources. The aim of this chapter is to present a framework of the use of GP model as a tool for the multi-objective resource-constrained project scheduling problem. Section 20.2 presents the GP model. Section 20.3 deals with the modeling of the scheduling problem under resource constraints. Section 20.4 defines the multi-objective resource-constrained project scheduling problem. A simplified GP formulation for the resource-constrained project scheduling problem is provided in Sect. 20.5. Section 20.6 is devoted to some concluding remarks.

20.2 Goal Programming Model

Linear programming (LP) was initially developed by George Dantzig in the 1940s and it has been applied widely in several domains and economic sectors. Within the mono-criteria paradigm, the LP models optimize only one objective function such as the maximization of the profit or the minimization of the cost. In the context of a multi-criteria decision aid paradigm, the GP model can be seen as an extension of the LP models where several incommensurable and conflicting objectives are simultaneously aggregated and optimized. This model is a DFM where the deviations between the achievement and the aspiration levels are to be minimized. Through this model, the DM looks for a solution (alternative) with an achievement level closer to the target values g_μ for each objective (for $\mu = 1, 2, \ldots, \nu$). This can be expressed by the following mathematical relation: $f_\mu(x_1, x_2, \ldots, x_n) \cong g_\mu$ (for $\mu = 1, 2, \ldots, \nu$). The left side of this relation denotes the achievement level or the performance of the different alternatives in contributing to the objective μ. The right side is the DM's aspiration levels (goals) associated with each objective. The difference between the achievement and the aspiration levels is expressed by the unwanted positive δ_μ^+ and negative δ_μ^- deviations. In fact, the DM will be satisfied with an alternative that minimizes the following total sum of deviations $\sum_{\mu=1}^{\nu}\left(\delta_\mu^+ + \delta_\mu^-\right)$.

For each objective, the DM may face the three following situations:

1. The achievement and the aspiration levels are equal ($f_\mu(x_1, x_2, \ldots, x_n) \cong g_\mu$, for some $\mu = 1, 2, \ldots, \nu$). This means that the obtained solution fits well the target value fixed by the DM for specific objectives. In such case, there are no deviations from the goals ($\delta_\mu^+ = \delta_\mu^- = 0$).
2. The achievement level is lower than the aspiration level ($f_\mu(x_1, x_2, \ldots, x_n) < g_\mu$, for some $\mu = 1, 2, \ldots, \nu$). In such case, the target value of the objective has not been met and there are negative deviations for some $\mu(\delta_\mu^- > 0$ and $\delta_\mu^+ = 0)$.
3. The achievement level is greater than the aspiration level ($f_\mu(x_1, x_2, \ldots, x_n) > g_\mu$, for some $\mu = 1, 2, \ldots, \nu$). In such case, the target value of the objective has been met and there are positive deviations for some $\mu(\delta_\mu^+ > 0$ and $\delta_\mu^- = 0)$.

From these three situations, the unwanted deviational variables δ_μ^+ and δ_μ^- are mutually exclusive and the solution of the best compromise will satisfy the relation $\delta_\mu^+ \cdot \delta_\mu^- = 0$, (for $\mu = 1, 2, \ldots, \nu$) (Martel and Aouni 1990). Hence, there is no need to add this relation in the constraints of the GP model.

The GP aims to minimize the total sum of deviations under the two following types of conditions: (a) constraints related to the objectives, and (b) constraints of the project or the decision-making situation. The alternative that optimizes the objectives must respect the model constraints. Charnes and Cooper (1961) developed the first standard formulation of the GP model as follows:

$$\begin{array}{lll}
\text{Min.} & Z = \sum_{\mu=1}^{\nu}\left(\delta_\mu^+ + \delta_\mu^-\right) & \\
\text{s.t.} & f_\mu(x) + \delta_\mu^- - \delta_\mu^+ = g_\mu & (\mu = 1, 2, \ldots, \nu) \\
 & x \in X & \\
 & \delta_\mu^- \text{ and } \delta_\mu^+ \geq 0 & (\mu = 1, 2, \ldots, \nu)
\end{array}$$

This formulation is known as the standard GP where the objectives have the same relative importance. However, the DM may appreciate the objectives differently. Moreover, for the same objective, the DM can provide different weights for the positive and the negative deviations depending on his/her preferences. In order to integrate the relative importance of the objectives within the GP model, Charnes and Cooper (1977) have proposed the weighted GP as follows:

$$\begin{array}{lll}
\text{Min.} & Z = \sum_{\mu=1}^{\nu} \left(w_{\mu}^{+} \delta_{\mu}^{+} + w_{\mu}^{-} \delta_{\mu}^{-} \right) & \\
\text{s.t.} & f_{\mu}(x) + \delta_{\mu}^{-} - \delta_{\mu}^{+} = g_{\mu} & (\mu = 1, 2, \ldots, \nu) \\
 & x \in X & \\
 & \delta_{\mu}^{-} \text{ and } \delta_{\mu}^{+} \geq 0 & (\mu = 1, 2, \ldots, \nu)
\end{array}$$

where w_{μ}^{+} and w_{μ}^{-} are the relative importance (weights) of the positive and the negative deviations for objective μ, respectively. According to Kettani et al. (2004), the weights play the following two roles: (a) the normalization of different measurement scales of the objectives, and (b) the valuation of each objective. Zeleny (1982) highlighted that there are two types of weight estimates namely subjective and objective estimates. The subjective estimates depend on social, cultural, traditional and environmental influences. The objective estimates are related to the intrinsic information contained in the alternatives and the objectives. Generally, the DM provides the relative importance of the objectives at an early stage of the decision-making process and the determination of the weights is usually done in interactive and iterative manners (Lee and Olson 1999). This can be seen as a learning process where the DM can learn more about his/her preferences. The DM can modify the values of the weights while he/she is evolving towards the best recommendation or the solution of the best compromise. In fact, there is no optimal solution or alternative since the objectives are, in general, conflicting and incommensurable. Therefore, the DM has to make some tradeoffs based on his/her preferences. The different variants of GP integrate differently of DM's preferences during the decision-making process. In their paper, Aouni et al. (2009) present the main variants of the GP model and they propose a typology based on the moment that the DM's preferences are integrated in the GP variant. Aouni et al. (2009) present and discuss the formulations of the most known variant of the GP model. Within this section, we will provide the standard and the weighted GP variants. We will also present the mathematical formulation of the Lexicographic GP (LGP) or the pre-emptive GP. The LGP is one of the well and wide applied variant because it is easier for the DM to provide his/her preferences through a lexicographic order. The most important objectives will be at the higher level and so on. Those are less important for the DM will be at the lower priority levels. In fact, the objectives in the lower priority levels will play a marginal role in the decision-making process. This can be seen as a limit of the LGP.

The mathematical formulation of the LGP model is as follows:

$$\begin{aligned}
&\text{Lex. Min.} && Z = \left[l_1\left(\delta^-, \delta^+\right), l_2\left(\delta^-, \delta^+\right), \ldots, l_h\left(\delta^-, \delta^+\right)\right] \\
&\text{s.t.} && f_\mu(x) + \delta_\mu^- - \delta_\mu^+ = g_\mu \qquad (\mu = 1, 2, \ldots, \nu) \\
& && x \in X \\
& && \delta_\mu^- \text{ and } \delta_\mu^+ \geq 0 \qquad (\mu = 1, 2, \ldots, \nu)
\end{aligned}$$

where Z represents an ordered vector of the deviation variables, h indicates the priority levels. Obtaining a solution through the LGP model requires solving h subprograms in sequential manner. The objectives at the higher priority levels play an important role and the obtained values of their deviations will be introduced successively as constraints within the mathematical programs related to the objectives placed in lower levels of priority.

The GP has been applied in several domains and in the literature it is considered as the most known model of the multi-objectives programming tools. This model is continually fed with theoretical developments and new applications with resounding success (Aouni and Kettani 2001). In Sect. 20.5 of this chapter we will be presenting how the GP model has been applied to the scheduling problem under resource constraints. In the Sect. 20.3, one will provide a brief review of the resource-constrained project scheduling problem.

20.3 Modeling of the Project Scheduling Problem Under Resource Constraints

The resource-constrained project scheduling problem (RCPSP) consists of a set $V = \{1, \ldots, n\}$ of n activities. A project may be represented by an activity-on-node network. Two additional dummy activities of null duration, noted by 0 and $n + 1$ are introduced to set the beginning and the end nodes of the network and they provide, respectively, the start and the completion time of the project. Logical and technological execution constraints between two activities result in precedence relations, denoted by (i, j), between the activities concerned. Hence, the precedence relations define a partial order relation. The set $Pred(j)$ denotes all immediate predecessors of the activity j so that $i \in Pred(j)$ if and only if i immediately precedes j. Here, V denotes the set of all the nodes representing the activities of the project and E denotes the set of precedence relations between the activities. The corresponding graph, denoted by $G = (V, E)$, has to be without circuit. Each activity j has a fixed time duration, noted $p_j \in \mathbb{N}$. Furthermore, all the project activities cannot be interrupted during their execution. To execute the project, each activity may require renewable resources taken out of a set of resource types given by $\mathscr{R} = \{1, \ldots, K\}$. For each resource type k, $k \in \mathscr{R}$, a determined amount of renewable resources, noted $r_{jk} \in \mathbb{N}$, is required for each period (time unit) t of the activity duration p_j to execute the activity j. The renewable resources are allocated for the activity duration only and they become again available at the end of the activity, hence the name renewable resources. The maximum amount of renewable resources of type k available at each period t is given by R_k. In this traditional RCPSP model, the rate of use of renewable resources of type k is constant during the execution of each activity j.

In the same way, the objective function of the traditional RCPSP optimization problem is to minimize the longest path length of the project activity network, often designated by the project makespan. A feasible solution $\mathscr{S}$ to the scheduling problem gives the start times of each activity j, satisfying the following two conditions:

1. the immediate predecessor relationships,
2. the availability constraints of resources for each time period.

A regular objective function is a non-decreasing function of the activity start times, e.g., the makespan minimization in the RCPS problem is a regular objective function.

In practice, the search for an optimal solution to this problem is only applicable to projects of small size up to 60 tasks. Błażewicz et al. (1983) has demonstrated that the RCPSP belongs to the class of $\mathscr{NP}$-hard problems. To solve large scale RCPSPs in reasonable time, efficient heuristics or meta-heuristics are needed.

The best known heuristic method being used belongs to one of two classes: the class of priority rule-based heuristic (Boctor 1993) and the class of local search approaches often called meta-heuristics. A priority rule-based heuristic builds a schedule by selecting activities from a range of activities available successively so that all activities are sequenced (Boctor 1993). There are still two types of priority rule-based heuristics, the serial heuristics where the priority of the activities is predetermined and remains fixed for the duration of the scheduling process, and the parallel heuristics where the activity priorities of the current active set at period t are updated each time an activity is inserted in the partial schedule under construction.

The class of local search approaches available in the literature groups among them, the tabu search, the simulated annealing and genetic algorithms (Hartmann 1999). These meta-heuristics start with an initial solution, often obtained by a priority rule-based heuristic method, and try to improve it according to varying search strategies. The improvement of current solutions at a given stage of the search is obtained by transforming one or several solutions into new ones. These meta-heuristics stop with the best solution found when given conditions are satisfied.

Over the years, to better reflect the needs of project management and to implement diverse constraints to reflect practical situations, many variants and extensions of the basic RCPSP problem model have been introduced. Variants may account, for example, for the variability of the amount of resources available during the project execution, lags and delays between the activities to accelerate the project, activity due dates and a project deadline. We refer the reader to the papers by Brucker et al. (1999); Kolisch and Hartmann (1999) and Artigues et al. (2008) for a survey of many of the best known RCPSP problem variants.

Other objective functions with appropriate solution methods have been develop to satisfy Project Manager (PM) decision making requirements. Depending on the management needs of particular projects, the PM may be looking to satisfy other objectives such as the minimization of the total cost of the project, the maximization of the net present value, the minimization of the maximum workload of renewable resources over a given period and other resource related objective functions such as resource leveling. Considering and aggregating simultaneously several conflicting and incommensurable objectives requires trade-offs that the PM should made according to his/her preferences. In the Sect. 20.4, one will be discussing the Multi-Objective Resource-Constrained Project Scheduling Problem (MORCPSP).

20.4 Multi-Objective Resource-Constrained Project Scheduling Problem

In fact, the RCPSP is an important academic problem in project management, since many real-life applications in engineering, forestry, mining operations, software implementations and many other areas make more and more use of such mathematical modeling. During the last two decades, the RCPSP problem has been largely studied in the Operations Research literature. In solving the RCPSP problem, the minimization of the project duration, noted C_{max}, is the usual objective (Kolisch and Hartmann 2006). That objective is defined as the duration between the start time and the end time of the project. In some cases, the objective could be the maximization of the net present value of the project, the minimization of the project cost or the minimization of the quantity of resources of given types, allocated in a time period during the project planning. For each of these objectives, extensive research and progress has been reported, mainly through the use of heuristic procedures for practical size project seeking solutions in efficient time as reported in the literature.

Moreover, in real applications, the PM, as a decision maker, has to consider several objectives at the same time in deciding on a particular baseline schedule for the project. Here, the PM must usually consider trade-offs (compromises) between the project duration, the project cost of different resource types and other performance measures. Nowadays, for the implementation of project management in organizations, one observes that PMs rely to A Guide to the Project Management Body of Knowledge—PMBOK Guide, Fifth edition (PMI 2013). That Guide suggests producing, as an output, a baseline schedule after considering activity durations and its resource requirements, among others. Therefore, the project scheduling problems could be linked to multi-objective decision-making problems. In this context, one faces to the MORCPSP. In this multi-objective paradigm, this chapter focuses on methodologies for explicitly considering several objectives or criteria used for the choice by the PM of the “best” baseline schedule. The need for further research to improve these multi-objective scheduling tools was raised by Pollack-Johnson (1998). It should be noted here that the literature on project scheduling based on several objectives is sparse when compared to the mono-criterion one. One direction of improvement would benefit from incorporating methodological developments from multi-objective decision-aid literature. Recent results in multiple objectives literature for project management are promising in this context (Gagnon and d'Avignon 2006; Gagnon et al. 2007, 2012).

In order to explore the generation of baseline schedules, one needs to recall the basic formulation of MORCPSP. Conceptually, a MORCPSP has the following form:

$$\begin{aligned} &\text{Optimize} \quad f\,(S) = \{f_1\,(S)\,, f_2\,(S)\,, \ldots, f_\nu\,(S)\} \\ &\text{such that} \quad S \in \mathscr{S} \end{aligned}$$

where $S = (S_0, \ldots, S_{n+1}) \in \mathbb{R}^{n+2}_{\geq 0}$ denotes a vector of decision variables corresponding to a schedule, $\mathscr{S}$ is the set of feasible schedules and $f\,(\mathscr{S})$ is the image of the feasible set in the criterion space representing the criteria to be optimized. Set $\mathscr{S}$ is given implicitly by the various scheduling constraints and the selected criteria by restraining the size of the set of feasible schedules. The solution of a MORCPSP consists in choosing the "best values" of the vector $f\,(S)$ calculated on $\mathscr{S}$ from the set $\mathscr{S}$. In general, a MORCPSP yields to a large set of Pareto optimal solutions. A feasible solution $\mathscr{S}$ is considered as a Pareto optimal or efficient if there is no better feasible solution $S' \in \mathscr{S}$ such that $f\,(S') \leq (S)$ and $f_\mu\,(S') < f_\mu\,(S)$ for some μ, where μ indexes the multi-objective function vector, and assuming the minimization of all objective functions. In this case, the multi-objective vector $f\,(S)$ is said to dominate $f\,(S')$. For this and since the objectives are conflicting, it is important to be aware of trade-offs among the solutions because the improvement of one objective function value is generally gained at the expense of at least another objective function value. The Pareto optimal set, denoted by $\mathscr{PF}$, is a subset of all the possible solutions in $\mathscr{S}$. A class of solution algorithms aims to determine the Pareto front which is the set that contains the evaluated objective vectors of $\mathscr{PF}$.

Among the many attempts in solving the MORCPSP, one considers the use of the goal programming (GP) which has the advantage of explicitly making use of the PM's preferences in choosing a baseline schedule for the project. Section 20.5 presents the methodological formulation of how the GP incorporates explicitly the PM's preferences into a trade-offs analysis for the generation of a baseline schedule. The idea of trade-off analysis between quality, time, and cost for the project has recently been proposed by Pollack-Johnson and Liberatore (2006). In fact, one utilizes the weighted GP model introduced in Sect. 20.2 to solve the MORCPSP discussed in Sect. 20.3 and described in this section.

20.5 Resource-Constrained Project Scheduling Through the Goal Programming Model

In order to present the GP formulation for the MORCPSP, one used the notation defined in Sect. 20.3 as well as the following ones. The project planning horizon, denoted by T, is a number of time periods, noted by $t = 1, \ldots, T$, large enough to complete all project activities. Given the upper bound T on the project duration, the precedence relations can be utilized to calculate the earliest start time ES_j and latest completion time LC_j, containing the precedence feasible completion time of activity $j(j \in V)$, by forward and backward recursion. Moreover, the maximal availability R_k of resources of type k is given by the PM.

The literature offers different formulations for the project cost. In this paper, we are interested to minimize the resource availability cost over the project duration. Considering a per-period resource availability cost c_{kr} for every named resource r of type $k(r \in \{1, \ldots, R_k\})$, the PM can derive an upper bound on the project cost C, which is usually linked to the project budget. For a given schedule and having the required amount of resources for each activity, one wants to find the appropriate number of resources of each type to be assigned over the project duration. Now, having a per-period availability cost for each named resource, one determines the project availability cost by multiplying the project duration by the availability cost of resources assigned to the project according to a project schedule.

Consider a PM pursuing the following project scheduling objectives: (a) the minimization of the project duration (makespan); (b) the minimization of the project availability cost ; and (c) the minimization of the quantity of allocated resources of each type.

One uses the couple (x, y) to denote a schedule S to the MORCPSP. Hence, the first entry of the couple, x, is a vector indicating the finished periods where:

$$x_{jt} = \begin{cases} 1, & \text{if the activity } j \text{ finishes in period } t, \\ 0, & \text{otherwise.} \end{cases}$$

The second entry of the couple, y, is a vector indicating the resource assignments where:

$$y_{kr} = \begin{cases} 1, & \text{if the resource } r \text{ of type } k \text{ is assigned to project,} \\ 0, & \text{otherwise.} \end{cases}$$

With the above notation, one observes that a solution or a schedule to the MORCPSP is given by the couple (x, y). We will be using also the following notations:

ES_j, LC_j Earliest start and latest completion times of activity j

p_j Duration, in consecutive periods, of activity j

$Pred\,(j)$ Set of immediate predecessors of activity j

T Feasible project planning horizon

g_T Goal for the project duration specified by the PM, $g_T \leq T$

c_{kr} Per-period availability cost of named resource r of type k when allocated to the project

C Upper bound on project cost

g_c Goal for the project cost specified by the PM, $g_c \leq C$

r_{jk} Required amount of resources of type k per period to execute activity j

R_k Maximal availability of resources of type k, $k \in \mathscr{R}$, at each point in time

g_k Goal for the amount of resources of type k allocated during the project, $(k \in K)$

x_{jt} Zero-one decision variable equal to 1 if the activity j ends in the time period t and equal to 0 otherwise

y_{kr} Zero-one decision variable equal to 1 if the named resource r of type k is allocated to the project and 0 otherwise

$f_T\,(x, y)$ Project duration for the schedule $S = (x, y)$, $f_T\,(x, y) = \sum_{t=ES_{n+1}}^{LC_{n+1}} tx_{n+1,t}$

$f_c\,(x, y)$ Project cost for the schedule $S = (x, y)$, $f_c\,(x, y) = \left(\sum_{t=ES_{n+1}}^{LC_{n+1}} tx_{n+1,t}\right)\left(\sum_{k=1}^{K}\sum_{r=1}^{R_k} c_{kr}y_{kr}\right)$

$f_k\,(x, y)$ Amount of resources of type k allocated in a period during the project for the schedule $S = (x, y)$, $f_k\,(x, y) = \sum_{r=1}^{R_k} y_{kr}(k = 1, 2, \ldots, K)$

δ_μ Deviation between the objective function value $f_\mu\,(x, y)$ and the goal g_μ, $\mu = T, 1, 2, \ldots, K, c$

For a schedule $S = (x, y)$, the resource assignments, $f_k\,(x, y)$, gives precisely the maximum amount of resources of type k allocated in at least one period during the project. This amount will then be used to determine the availability cost for the whole project duration.

The weighted GP variant will be utilized to formulate the MORCPSP, as follows:

$$\text{Min. } Z = w_T\delta_T + w_c\delta_c + \sum_{k=1}^{K} w_k\delta_k$$

$$\text{s.t. } \sum_{t=ES_j+p_j}^{LC_j} x_{jt} = 1 \quad (j = 0, 1, \ldots, n+1) \tag{20.1}$$

$$\sum_{t=ES_i+p_i}^{LC} tx_{i,t} \leq \sum_{t=ES_j+p_j}^{LC_j} \left(t - d_j\right) x_{j,t} \quad (i \in Pred\,(j)\,;\, j = 1, 2, \ldots, n+1) \tag{20.2}$$

$$\sum_{j=1}^{n} \sum_{\tau=\max\{t,ES_j+p_j\}}^{\min\{t+p_j-1,LC_j\}} r_{jk} x_{j\tau} \le \sum_{r=1}^{R_k} y_{kr}$$
$$(k = 1, \ldots, K; t = 1, 2, \ldots, T) \quad (20.3)$$

$$\sum_{r=1}^{R_k} y_{kr} - \delta_k = g_k \quad (k = 1, \ldots, K) \quad (20.4)$$

$$\left(\sum_{t=ES_{n+1}}^{LC_{n+1}} t x_{n+1,t}\right) \cdot \left(\sum_{k=1}^{K} \sum_{r=1}^{R_k} c_{kr} y_{kr}\right) - \delta_c = g_c \quad (20.5)$$

$$\sum_{t=ES_{n+1}}^{LC_{n+1}} t x_{n+1,t} - \delta_T = g_T \quad (20.6)$$

$$x_{jt} = \{0, 1\} \qquad \left(j = 0, 1, \ldots, n+1; t = ES_j + p_j, \ldots, LC_j\right) \quad (20.7)$$

$$y_{kr} = \{0, 1\} \qquad (k = 1, \ldots, K; r = 1, \ldots, R_k) \quad (20.8)$$

$$\delta_T, \delta_c \text{ and } \delta_k \ge 0$$

Since the three considered objectives are to be minimized, the target values (goals) for the objectives can be seen as an ideal point that can be computed by minimizing separately each objective subject to the same set of constraints. The target values, included in the previous weighted GP model, g_k $(k = 1, \ldots, K)$, g_c, and g_T are obtained by solving the following three mathematical programs:

1. Target value for the amount of resources of type k to be allocated during the project (g_k):

   Min. $\sum_{r=1}^{R_k} y_{kr}$
   Subject to the constraints (20.1), (20.2), (20.3), (20.7) and (20.8).

2. Target value for the resource availability cost of the project (g_c):

   Min. $\left(\sum_{t=ES_{n+1}}^{LC_{n+1}} t x_{n+1,t}\right) \cdot \left(\sum_{k=1}^{K} \sum_{r=1}^{R_k} c_{kr} y_{kr}\right)$
   Subject to the constraints (20.1), (20.2), (20.3), (20.7) and (20.8).

3. Target value for the project duration (g_T):

   Min. $\sum_{t=ES_{n+1}}^{LC_{n+1}} t x_{n+1,t}$
   Subject to the constraints (20.1), (20.2), (20.3), (20.7) and (20.8).

The parameters w_T, w_c and w_k are the relative importance of the objectives: (a) the project duration, (b) the project cost, and (c) the quantity of the allocated resources of type k, $(k = 1, \ldots, K)$, respectively. These parameters are provided by the PM according to his/her preferences. This problem can be also formulated through the Lexicographic GP where the PM will be asked to provide the priority level of the three objectives. For simplicity, we will keep in this chapter only the formulation of the weighted GP. In their paper, Gagnon et al. (2012) provide a GP formulation for the MORCPSP where the PM's preferences are explicitly integrated through the satisfaction functions.

A Tabu search algorithm has been designed to solve the MORCPS. The algorithm proceeds in two levels. At the first level, the algorithm generates a new resource configuration. Starting with this resource configuration at the second level, the algorithm iterates and tries to improve the current schedule by local search. For each schedule, the algorithm verifies the number of resources allocated and evaluates the objective functions simultaneously. It also keeps a current list of approximate Pareto optimal solutions. The list is updated each time the algorithm finds a new non-dominated solution and removes.

The proposed weighted GP formulation in this context will allow the PM to obtain the best baseline schedule that integrates simultaneously several conflicting objectives and his/her preferences through the relative importance of the objectives. This formulation can be extended to other variants of the GP.

20.6 Conclusions

The aim of this chapter was to present a simple formulation of the MORCPSP. The provided formulation is based on the GP model and it aggregates simultaneously the following three incommensurable and conflicting objectives: (a) the project duration, (b) the quantity of resources of each type to be allocated in a period during the project, and (c) the project cost. The proposed model generates the best baseline schedule that will allow monitoring and controlling the realization of a project. The PM's preferences are partially integrated through the relative importance coefficients (the weights) associated with the project scheduling objectives. In fact, these weights will allow the PM to make compromises among the project duration, the project cost, and the amount of resources to be allocated. This process of elucidating the PM's preferences will be done through an iterative manner that allows him/her to learn more about the decision-making situation. It is a learning process that fits well with the paradigm of decision aid axioms where the PM is involved along the decision-making process.

References

Aouni B, Kettani O (2001) Goal programming model: a glorious history and a promising future. Eur J Oper Res 133:225–231

Aouni B, Hassaine A, Martel J-M (2009) Decision-maker's preferences modelling within the goal programming model: a new typology. J Decis Anal 16:163–178

Artigues C, Dempsey S, Néron E (eds) (2008) Resource-constrained project scheduling: models, algorithms, extensions and applications. Wiley, Hoboken

Błażewicz J, Lenstra JK, RinnooyKan AHG (1983) Scheduling subject to resource constraints: classification and complexity. Discrete Appl Math 5:11–24

Boctor FF (1993) Heuristics for scheduling projects with resource restrictions and several resource-duration modes. Int J Prod Res 31:2547–2558

Brucker P, Drexl A, Möhring R, Neumann K, Pesch E (1999) Resource-constrained project scheduling: notation, classification, models, and methods. Eur J Oper Res 112:3–41

Caballero R, Hernandez M (2006) Restoration of efficiency in a goal programming problem with linear fractional criteria. Eur J Oper Res 172:31–39

Caballero R, Gomez T, Ruiz F (2006) Goal programming: realistic targets for the near future. In: Proceedings of the VII international conference on multi-objective and goal programming, Toulouse

Charnes A, Cooper WW (1961) Management models and industrial applications of linear programming. Wiley, New York

Charnes A, Cooper WW (1977) Goal programming and multiple objectives optimisations. Eur J Oper Res 1:39–54

Charnes A, Cooper WW, Ferguson R (1955) Optimal estimation of executive compensation by linear programming. Manage Sci 1:138–151

Gagnon M, d'Avignon GR (2006) Project planning and scheduling using goal programming models. In: Proceedings of ASAC 2006, Banff, Canada

Gagnon M, d'Avignon GR, Aouni B (2007) Project planning and scheduling using goal programming models. In: Proceedings of ASAC 2007, Ottawa

Gagnon M, d'Avignon G, Aouni B (2012) Resource-constrained project scheduling through the goal programming model: integration of the manager's preferences. Int Trans Oper Res 19:547–565

Glover F, Sueyoshi T (2009) Contributions of Professor William Cooper in operations research and management science. Eur J Oper Res 197:1–16

Hartmann S (1999) Project scheduling under limited resources: models, methods, and applications. LNEMS, vol 478. Springer, Berlin

Ignizio JP (1976) Goal programming and extensions. Lexington Books, Lexington

Jones DF, Tamiz M (2002) Goal programming in the period 1990–2000. In: Ehrgott M, Ganibleux X (eds) Multicriteria optimization: state of the art annotated bibliographic surveys. Kluwer, Boston, pp 129–170

Kettani O, Aouni B, Martel J-M (2004) The double role of the weight factor in the goal programming model. Comput Oper Res 31:1833–1845

Kolisch R, Hartmann S (1999) Heuristic algorithms for solving the resource-constrained project scheduling problem: classification and computational analysis. In: Wellers J (ed) Handbook on recent advances in project scheduling. Kluwer, Boston, pp 147–178

Kolisch R, Hartmann S (2006) Experimental evaluation of heuristics for the resource-constrained project scheduling problem: an update. Eur J Oper Res 174:23–37

Lee SM (1973) Goal programming for decision analysis of multiple objectives. Sloan Manage Rev 14:11–24

Lee SM, Olson DL (1999) Goal programming. In: Gal T, Stewart TJ, Hanne T (eds) Multicriteria decision making: advances in MCDM models, algorithms, theory and applications. Kluwer, Boston, pp 203–235

Martel J-M, Aouni B (1990) Incorporating the decision-maker's preferences in the goal programming model. J Oper Res Soc 41:1121–1132

PMI (2013). A guide to the project management body of knowledge (PMBOK®Guide). Project Management Institute, Newtown Square

Pollack-Johnson B, Liberatore MJ (1998) Project management software usage patterns and suggested research directions for future developments. Proj Manage J, 29(2): 19–28

Pollack-Johnson B, Liberatore MJ (2006) Incorporating quality considerations into project time/cost tradeoff analysis and decision making. IEEE T Eng Manage 53(4):534–542

Simon HA (1956) Rational choice and the structure of the environment. Psychol Rev 63:129–138

Simon HA (1976) From substantive to procedural rationality. In: Latsis S (ed) Method and appraisal in economics. Cambridge University Press, Cambridge

Tamiz M, Jones DF, El-Darzi E (1995) A review of goal programming and its applications. Ann Oper Res 58:39–53

Tamiz M, Jones DF, Romero C (1998) Goal programming for decision-making: an overview of the current state-of-the-art. Eur J Oper Res 111:569–581

Zeleny M (1982) Multiple criteria decision making. McGraw-Hill, New York

Part VII
Multi-Mode Project Scheduling Problems

Chapter 21
Overview and State of the Art

Marek Mika, Grzegorz Waligóra, and Jan Węglarz

Abstract In this chapter we present a state-of-the-art in the area of multi-mode project scheduling problems. These problems are characterized by the fact that each activity of a project can be executed in one of several modes, representing a relation between the resource requirements of the activity and its duration. In the overview we present the models and solution approaches that have been proposed in the literature across the class of multi-mode project scheduling problems up to now. Firstly we deal with the basic multi-mode resource-constrained project scheduling problems with the objective to minimize the project duration. We present the mixed-integer linear programming formulations of the problem, describe the exact approaches, the existing methods for lower bounds calculation, as well as heuristic approaches to solve the problem. Secondly, we also discuss special cases and extensions of the basic problem. Finally, we analyze multi-mode problems with other objectives, distinguishing between financial and resource-based objectives.

Keywords Exact approaches • Heuristics • Lower bounds • Multi mode • Project scheduling • Resource constraints

21.1 Introduction

The multi-mode resource-constrained project scheduling problem (MRCPSP), denoted in the three-field classification $\alpha\,|\beta|\,\gamma$ for project scheduling as $MPS\,|\,prec\,|\,C_{max}$, is a generalization of the single-mode RCPSP ($PS\,|\,prec\,|\,C_{max}$) (see Chap. 1 of this handbook). There are two main differences between these problems. In the MRCPSP each activity can be performed in one of several execution modes, and two other basic categories of discrete resources are considered in addition to renewable ones: nonrenewable and doubly constrained. A mode is a pair whose first element is a vector of resource requirements of an activity, and the

M. Mika (✉) • G. Waligóra • J. Węglarz
Institute of Computing Science, Poznań University of Technology, Poznań, Poland
e-mail: Marek.Mika@cs.put.poznan.pl; Grzegorz.Waligora@cs.put.poznan.pl; Jan.Weglarz@cs.put.poznan.pl

C. Schwindt, J. Zimmermann (eds.), *Handbook on Project Management and Scheduling Vol. 1*, International Handbooks on Information Systems,
DOI 10.1007/978-3-319-05443-8_21

second element is the activity processing time. Doubly constrained resources are usually not considered explicitly, since each such resource can be replaced by a pair of one renewable and one nonrenewable resource, according to the transformation proposed by Talbot (1982).

In the MRCPSP a set of n nonpreemptable activities have to be performed using and consuming resource units from a given set of resources. The structure of the project is represented by an Activity-on-Node network $G = (V, E)$ where set of nodes V represents activities, and set of arcs E represents direct precedence constraints between pairs of activities. Usually, two additional dummy activities are members of the set V: activity 0 which is the unique initial activity without predecessors, and activity $n + 1$ which is the unique terminal activity without successors. Both dummy activities have durations and all resource requirements equal to 0. The planning horizon is divided into time periods which means that time is also a discrete variable. All activities and resources are available at the start of the project. The objective is to find a vector of mode assignments as well as precedence- and resource-feasible start times for all activities such that the project duration (or makespan) is minimized.

The MRCPSP is $\mathscr{NP}$-hard in the strong sense being a generalization of the RCPSP. Moreover, for more than one nonrenewable resource the problem of finding a feasible solution is already $\mathscr{NP}$-complete (Kolisch 1995).

In this chapter we present a survey of recent developments concerning the deterministic multi-mode resource-constrained project scheduling problem. Within this class of problems it is an updated version of the survey presented by Węglarz et al. (2011).

The chapter is organized as follows. In Sect. 21.2 some mixed-integer linear programming formulations are presented. In Sects. 21.3–21.6 we survey the results concerning the multi-mode problems with the makespan minimization objective. Section 21.3 deals with the exact approaches. In Sect. 21.4 several methods of the lower bounds calculations are presented. Section 21.5 is devoted to the heuristic approaches and is divided into three parts concerning priority-rule based algorithms, metaheuristics and local search, and other heuristic approaches. Special cases and extensions of the MRCPSP with the makespan minimization objective are described in Sect. 21.6. Section 21.7 deals with multi-mode problems with objectives other than the makespan, from among which the maximization of the net present value (NPV) is discussed most extensively.

21.2 MILP Formulations

The most commonly used mathematical model of the MRCPSP was introduced by Talbot (1982), and based on the model of the RCPSP by Pritsker et al. (1969). It is formulated as follows:

$$\text{Min.} \sum_{t=EC_{n+1}}^{LC_{n+1}} t \cdot z_{n+1,1,t} \tag{21.1}$$

$$\text{s.t.} \sum_{m \in \mathscr{M}_j} \sum_{t=EC_j}^{LC_j} z_{jmt} = 1 \qquad (j \in V) \tag{21.2}$$

$$\sum_{m \in \mathscr{M}_j} \sum_{t=EC_i}^{LC_i} t \cdot z_{imt} \le \sum_{m \in \mathscr{M}_j} \sum_{t=EC_j}^{LC_j} t \cdot z_{jmt} - p_{jm} \qquad \big((i,j) \in E\big) \tag{21.3}$$

$$\sum_{j \in V^a} \sum_{m \in \mathscr{M}_j} \sum_{q=\max\{t,EC_j\}}^{\min\{t+p_{jm}-1,LC_j\}} r_{jkm} \cdot z_{jmq} \le R_k \qquad \begin{pmatrix} k \in \mathscr{R}; \\ t = 1,\ldots,\overline{d} \end{pmatrix} \tag{21.4}$$

$$\sum_{j \in V^a} \sum_{m \in \mathscr{M}_j} \sum_{t=EC_j}^{LC_j} r_{jkm} \cdot z_{jmt} \le R_k \qquad (k \in \mathscr{R}^n) \tag{21.5}$$

$$z_{jmt} \in \{0,1\} \qquad \big(j \in V; m \in \mathscr{M}_j; t = EC_j,\ldots,LC_j\big) \tag{21.6}$$

where:

- binary variable $z_{jmt} = 1$ if and only if activity j executed in mode $m \in \mathscr{M}_j$ is completed at the end of time period t
- R_k is the number of available units of renewable (nonrenewable) resource $k \in \mathscr{R}$ $(k \in \mathscr{R}^n)$
- r_{jkm} is the number of units of the k-th renewable (nonrenewable) resource required by activity j executed in mode $m \in \mathscr{M}_j$
- p_{jm} is the duration of activity j executed in mode $m \in \mathscr{M}_j$
- EC_j, LC_j are the earliest and the latest completion time of activity j calculated assuming that the shortest duration mode is assigned to each activity, resource constraints are not taken into account, and the latest possible completion time for the entire project $\overline{d}$ is calculated for the modes with the longest durations.

Constraints (21.2) ensure that each nonpreemptable activity is performed exactly once in exactly one mode. Precedence constraints are guaranteed by inequalities (21.3). Constraints (21.4) and (21.5) ensure that the renewable and nonrenewable resource limits are not exceeded, respectively. Finally, constraints (21.5) define the binary status of the decision variables. The objective is to minimize the makespan, which is defined as completion time of the final dummy activity $n+1$ and is expressed by objective function (21.1). The solution of the problem (21.1)–(21.6) defines an optimal schedule as a pair of lists: (i) list of assignments of modes to activities, (ii) list of activity completion times.

Another mathematical formulation of the MRCPSP, based on the *feasible subsets* concept, is presented by Maniezzo and Mingozzi (1999). A feasible subset A_λ of the set of all activity-mode combinations is a set of pairs (j, m), where $m \in \mathscr{M}_j$ denotes the processing mode assigned to activity $j \in V^a$ that satisfy the following conditions:

- each activity $j \in V^a$ occurs in at most one pair (j, m)
- $\sum_{(j,m)\in A_\lambda} r_{jkm} \leq R_k$ for $k \in \mathscr{R}$
- $\sum_{(j,m)\in A_\lambda} r_{jkm} \leq R_k$ for $k \in \mathscr{R}^n$
- there are no precedence constrains between any activities from subset A_λ

Three types of binary variables are used in this formulation:

- $z_{jt} = 1$ if and only if activity j is completed at the end of period t
- $y_{\lambda t} = 1$ if and only if all activities of feasible subset A_λ are executed in period t
- $x_{jm} = 1$ if and only if activity j is performed in mode $m \in \mathscr{M}_j$

The corresponding mathematical model of the MRCPSP is formulated as follows:

$$\text{Min.} \sum_{t=EC_{n+1}}^{LC_{n+1}} t \cdot z_{n+1,t} \tag{21.7}$$

$$\text{s.t.} \sum_{\lambda\in\mathscr{A}_{jm}} \sum_{t=ES_j}^{LC_j} y_{\lambda t} = p_{jm} \cdot x_{jm} \qquad \left(j \in V^a; m \in \mathscr{M}_j\right) \tag{21.8}$$

$$\sum_{\lambda=1}^{\Lambda} y_{\lambda t} \leq 1 \qquad \left(t = 1, \ldots, \overline{d}\right) \tag{21.9}$$

$$\sum_{m\in\mathscr{M}_j} x_{jm} = 1 \qquad (j \in V) \tag{21.10}$$

$$z_{jt} \geq \sum_{\lambda\in\mathscr{A}_j} y_{\lambda t} - \sum_{\lambda\in\mathscr{A}_j} y_{\lambda,t+1} \qquad \left(j \in V^a; t = EC_j, \ldots, LC_j\right) \tag{21.11}$$

$$\sum_{t=EC_j}^{LC_j} t \cdot z_{jt} - \sum_{t=EC_i}^{LC_i} t \cdot z_{it} \geq \sum_{m\in\mathscr{M}_j} p_{jm} \cdot x_{jm} \qquad ((i, j) \in E) \tag{21.12}$$

$$\sum_{j\in V} \sum_{m\in\mathscr{M}_j} r_{jkm} \cdot x_{jm} \leq R_k \qquad (k \in \mathscr{R}^n) \tag{21.13}$$

$$z_{jt} \in \{0, 1\} \qquad \left(j \in V; t = 1, \ldots, \overline{d}\right) \tag{21.14}$$

$$y_{\lambda t} \in \{0, 1\} \qquad \left(\lambda = 1, \ldots, \Lambda; t = 1, \ldots, \overline{d}\right) \tag{21.15}$$

$$x_{jm} \in \{0, 1\} \qquad \left(j \in V; m \in \mathscr{M}_j\right) \tag{21.16}$$

where: Λ is the number of feasible subsets, set $\mathscr{A}_{jm} = \{\lambda : (j, m) \in A_\lambda\}$, and $\mathscr{A}_j = \cup_{m \in \mathscr{M}_j} \mathscr{A}_{jm}$.

Constraints (21.8) guarantee that feasible subsets containing activity-resource combination for activity j executed in mode $m \in \mathscr{M}_j$ are in progress for exactly p_{jm} time periods. Constraints (21.9) ensure that in each time period t at most one feasible subset is in progress, whereas constraints (21.10) ensure that each activity is performed in exactly one mode. Constraints (21.11) guarantee that the processing of activity j terminates at time period t if this activity belongs to a feasible subset being in progress at time period t and does not belong to a feasible subset being in progress at time period $t + 1$. Precedence constraints are guaranteed by (21.12). Constraints (21.13) ensure that the nonrenewable resource limitations are not exceeded. Finally, constraints (21.14)–(21.16) define the binary status of the decision variables. Notice, that the renewable resource limitations do not occur explicitly in this formulation, because feasibility of a solution with respect to renewable resources is guaranteed by definition by the feasible subsets.

Recently, two papers concerning new mathematical formulations of the extensions of the MRCPSP have been published. Sabzehparvar and Seyed-Hosseini (2008) propose a mathematical formulation for the MRCPSP with generalized precedence constraints and mode-dependent time lags. Zapata et al. (2008) propose three alternative formulations of the multi-project version of the MRCPSP, in two of them time is a continuous variable. Obviously, under some assumptions these models may be used to formulate mathematically the MRCPSP.

Another MILP formulation for the MRCPSP based on the Resource-Task Network (RTN) representation is proposed by Kyriakidis et al. (2012). The formulation consists of five types of constraints: timing, slot, excess resources balances, excess resource capacities, and task operations constraints. The timing constraints ensure that the total duration of all time slots is equal to the planning horizon, and that activity i may extend over one or more consecutive time slots and the sum of the durations of these slots is equal to duration p_{im} of activity i executed in mode m. Slot constraints guarantee that each activity is performed exactly once in exactly one mode. Excess resource balances preserve the balance of resource k at slot boundary t. Excess resource capacity constraints ensure that the actual excess amount of any resource k lies between a lower and upper bound of its capacity. Finally, task operation constraints secure that surpluses of resource k consumed and produced by activity i lies between corresponding bounds.

21.3 Exact Approaches

Exact approaches are proposed by Talbot (1982), Patterson et al. (1989), Speranza and Vercellis (1993), Sprecher (1994), Sprecher et al. (1997), Hartmann and Drexl (1998), Sprecher and Drexl (1998), and Zhu et al. (2006). All approaches, except the one proposed by Zhu et al. (2006), are based on the branch-and-bound (B&B)

method and the idea to enumerate partial schedules. A detailed comparison of these methods is provided by Hartmann and Drexl (1998).

Talbot (1982) proposes a two-stage algorithm. In stage one, activities, resources, and modes are sorted according to some selected rules. In stage two, a priority rule heuristic is used to calculate an upper bound and then a B&B with backtracking is used. At each level μ of the search tree the first unscheduled activity j from the sorted list is assigned its first resource-feasible mode, and it is added to the partial schedule with the earliest precedence- and resource-feasible completion time C_j. If none of the modes of activity j is resource-feasible, then the algorithm backtracks to the previous level $\mu - 1$, and an attempt is made to reassign activity $j - 1$ previously scheduled at this level to the earliest feasible completion time greater than the previously assigned completion time C_{j-1}. If it is impossible, the next mode is selected for activity $j - 1$. This process continues until an attempt is made to backtrack below activity 0, or until activity $n + 1$ is scheduled. In the former case the best solution found so far is the optimal one. In the latter case a new improved solution with makespan C'_{max} is found. If C'_{max} is equal to the known solution lower bound then the obtained schedule is optimal. Otherwise, a new upper bound is set to C'_{max} and the process starts anew with activity 0 executed in its first mode.

A depth-first B&B with backtracking is also used by Patterson et al. (1989) where the solution is represented by two resource- and precedence-feasible lists: a list of selected modes and a list of activity completion times. The solution method consists of two phases: a problem initialization phase, and an enumeration phase. During the initialization phase, similar to the stage one of the Talbot's algorithm (1982), activities and their modes are sorted using some rules in order to heuristically generate a good initial schedule, that is used to calculate an initial upper bound. In the enumeration phase the *precedence tree* is used to guide the search in the set of all precedence-feasible sequences of activities. At each level of the search tree only one activity from the set of eligible activities is chosen together with an assigned mode. An eligible activity is defined as an activity, the predecessors of which are already scheduled. Next, the precedence- and resource-feasible start time of this activity is calculated that is not less than the start time assigned at the previous level of the search tree and the procedure pass to the next level where another activity is chosen. If we obtain the sink dummy activity it means that a complete schedule is found. In this case the upper bound on completion time of each activity is computed, and the backtracking to the previous level occurs, where the next untested mode for this activity is chosen. If all modes are already tested then the next eligible activity is chosen. If all activities from the set of eligible activities at the given level of the tree are already tested, the backtracking to previous level occurs once more. Optimality is guaranteed when a solution equal to a known lower bound is found, or when backtracking proceeds to the dummy activity 0. The bounding rules include the consideration of continuously updated upper bounds on the completion time of each activity, as well as procedures ensuring precedence and resource feasibility. The performance of this approach was examined for 91 problems with the number of activities equal to 10, 20, 30, 50, 100, and 500 (Patterson et al. 1990).

Speranza and Vercellis (1993) propose a depth-first B&B procedure which enumerates the set of *tight schedules*. This algorithm is used for a single project being a part of the multi-project model. It is shown that the set of tight schedules dominates the set of schedules, and therefore an enumeration procedure can be reduced to the set of tight schedules only. The presented algorithm uses so-called maximal extensions of partial schedules in order to generate tight schedules. However, Hartmann and Sprecher (1996) showed that the method might fail to find optimal solution if at least two renewable resources are used during the execution of the project. Moreover, for some problems with limited availability of nonrenewable resources the algorithm does not even find an existing feasible solution.

Another depth-first B&B algorithm where the precedence tree is used as an enumeration scheme is presented by Sprecher (1994). Moreover, the author proposes the following bounding rules: (i) Basic Time Window Rule – if an activity should be scheduled but the assigned completion time is greater than the current latest finish time, then the current partial schedule does not lead to a solution better than the best currently known; (ii) Non Delay Ability Rule – if an eligible activity cannot be feasibly scheduled in any mode in the current partial schedule without exceeding its latest completion time, then no other eligible activity needs to be examined at this level; (iii) Nonrenewable Resource Rule – if scheduling each currently unscheduled activity in the mode with the lowest requirement for a nonrenewable resource would exceed the capacity of this nonrenewable resource, then the current partial schedule cannot be feasibly completed; (iv) Local Left Shift Rule – if an activity that has been started at the current level of the B&B tree can be locally left shifted without changing its mode, then the current partial schedule needs not be completed; (v) Single Enumeration Rule (further improved by Hartmann and Drexl (1998) and called Precedence Tree Rule) – if for two activities i and j scheduled at the previous and at the current level of the B&B tree, respectively, $S_i = S_j$ and $i > j$, then the current partial schedule needs not be completed.

Sprecher et al. (1997) propose a B&B algorithm in which another enumeration scheme named mode and delay alternatives is used, which is a multi-mode extension of the delay alternative concept proposed by Christofides et al. (1987) and used by Demeulemeester and Herroelen (1992) for the RCPSP. In the mode and delay alternatives each node g of the search tree is associated with a fixed time t_g at which activities may start processing. An activity j becomes eligible at time t_g when all its predecessors have completion times smaller than t_g, and it is in progress at time t_g if $C_j - p_{jm} \leq t_g < C_j$, assuming that activity j is processed in mode $m \in \mathscr{M}_j$. At each level g of the search tree firstly a new decision point t_g is calculated as the earliest completion time of activities currently in progress. Then a set of eligible activities is computed. All eligible activities with modes assigned at the previous level are temporally started at time t_g. For the set of all new eligible activities (i.e., those becoming eligible at time t_g) a set of mode alternatives is calculated, where a mode alternative represents one of the possible choices of modes for eligible activities. Selecting one mode alternative the new eligible activities are temporally started at time t_g as well. Now, when all eligible activities are added to the set of activities in progress, some resource conflicts may occur. In order to resolve these conflicts, a set of minimal delay alternatives is computed, where a delay alternative is the set of activities the removal of which from the set of activities in progress at time t_g makes all resource constraints satisfied. A delay alternative is minimal if no proper subset of the delay alternative is also a delay alternative. One minimal delay alternative is chosen, and the activities from this set are delayed in the partial schedule considered at this level of the search tree. Next, another decision point is calculated for the next level of the search tree. If the complete schedule is obtained, a backtracking to the previous level occurs and the next minimal delay alternative or the next mode alternative is tested. The main differences between this approach and the precedence tree are that at each level more than one activity can be scheduled, and that at the current level one may withdraw decisions made at the previous level.

Sprecher et al. (1997) propose another enumeration scheme, named *mode and extension alternatives*, which is very similar to the mode and delay alternatives, and it is almost the same except that extension alternatives are employed instead of the delay alternatives. An extension alternative is a subset of activities in progress for which all resource constraints are satisfied and they are delayed but scheduled at time t_g. The bounding rules introduced or redefined in this implementation of B&B include: (i) Nondelayability Rule modified for the presented enumerating schemes is defined in the following way – if an eligible activity, the mode of which has not yet been fixed, cannot be started in the mode with the shortest duration at the current level of the search tree without exceeding its latest completion time, then no mode alternative needs to be examined at the current level; (ii) Data Reduction (so-called preprocessing) which looks as follows – the project data can be adapted by executing the following steps: (a) removing all nonexecutable modes (modes which always violate the resource constraints), (b) deleting the redundant nonrenewable resources (resources with availabilities greater than the maximal total resource requirements), and (c) eliminating all inefficient modes (modes with duration not shorter and resource requirements not less than those of another mode for the same activity). Steps b and c are executed iteratively until further elimination is no more possible. This preprocessing rule is also used in some metaheuristic approaches described in Sect. 21.5.2; (iii) Multi-mode Rule defined in the following way – assume that no currently unscheduled activity will be started before the completion time of a scheduled activity j when the current partial schedule is completed, if a multi-mode left shift or a mode reduction of activity j with resulting mode $m' \in \mathscr{M}_j$, $1 \leq m' \leq M_j$, can be performed on the current partial schedule and, moreover, $r_{jkm'} \leq r_{jkm}$ if holds for each nonrenewable resource $k \in \mathscr{R}^n$, then the current partial schedule needs not be completed; (iv) and finally Immediate Selection that looks as follows – the following situation is assumed: all activities that start before the current decision point t_g complete at or before t_g. After selecting a mode alternative, there is an eligible activity j with assigned mode $m \in \mathscr{M}_j$ which cannot be simultaneously processed with any other activity i in its assigned mode $m' \in \mathscr{M}_i$. Moreover, activity j cannot be simultaneously processed with any unscheduled activity h in any mode $m'' \in \mathscr{M}_h$. Then the delay alternative obtained from the set of activities in progress at time t_g after removing activity j from this set is the only minimal delay alternative that has to be examined, and the extension set containing only activity j is the only extension alternative that has to be examined.

Sprecher and Drexl (1998) use in their B&B algorithm the precedence tree enumeration scheme and many bounding rules proposed by Sprecher (1994). The newly introduced bounding rule is the Cutset Rule which is defined as follows. Let $\overline{PS}$ denote a previously evaluated partial schedule with cutset $CS\left(\overline{PS}\right)$, maximal completion time $C_{max}(\overline{PS})$, and leftover capacities $R_k(\overline{PS})$ of the nonrenewable resources $k \in \mathscr{R}^n$. Let PS be the current partial schedule considered to be extended by scheduling some activity j with start time S_j. If we have $CS\,(PS) = CS\left(\overline{PS}\right)$, $S_j \geq C_{max}\left(\overline{PS}\right)$ and $R_k\,(PS) \leq R_k\left(\overline{PS}\right)$ for all $k \in \mathscr{R}^n$, then PS needs not be completed, where a cutset of a partial schedule PS is defined as the set of activities scheduled in PS.

Hartmann and Drexl (1998) extend the approach proposed by Sprecher and Drexl (1998) by two new bounding rules, namely the Order Swap Rule and the Immediate Selection Rule for Precedence Tree. The Order Swap Rule is defined as follows. Consider a scheduled activity the finish time of which is less than or equal to any start time that may be assigned when completing the current partial schedule. If an order swap on this activity together with any of those activities that finish at its start time can be performed, then the current partial schedule needs not be completed. The Immediate Selection Rule for Precedence Tree looks as follows. Consider an eligible activity j no mode of which is simultaneously performable with any currently unscheduled activity in any mode. If the earliest feasible start time of each other eligible activity in any mode is equal to the maximal completion time of the currently scheduled activities, then j is the only eligible activity that needs to be selected for being scheduled at the current level of the B&B tree.

Another exact approach was proposed by Zhu et al. (2006). Although it is primarily developed for the multi-mode version of the RCPSP with partially renewable resources, it can be successfully used for the classical MRCPSP because both renewable and nonrenewable resources can be easily modeled using partially renewable resources. Instead of using the B&B method, which explicitly enumerates all (partial) schedules, they use a branch-and-cut (B&C) approach. In this approach the linear programming relaxation of the integer linear programming model is used to obtain a lower bound of the project duration at each node of the search tree. If the node of the search tree cannot be fathomed and has fractional value of the objective function, then the algorithm tries to find cuts, i.e., valid inequalities that are violated by the fractional solution but are satisfied by all feasible integer solutions represented by this node in the search tree. If no cut is found for a given node, then the branching is performed to create new nodes in the search tree. The rules used in the considered B&C algorithm can be divided into four categories: reduction of the number of variables, branching and bound tightening, cuts, and high-level search strategy. The number of variables is reduced using: (i) time windows derived from the improved distance matrix, (ii) lower bound based on the usage of renewable resources, (iii) lower bound obtained by a truncated version of B&C, (iv) upper bound obtained by the authors' implementation of a genetic algorithm. The considered B&C algorithm uses the cut-generating features built into the mixed integer programming (MIP) solver that is a part of CPLEX (ILOG 2002), as well as two other problem-specific cuts: cuts from make the branching more effective, a procedure operating on the so-called special ordered set (SOS) is used for bound tightening.

Another bound tightening procedure, which takes into account the current value of the makespan, is proposed for the problem with makespan minimization criterion. Finally, local branching (Fischetti and Lodi 2003) procedure is used as high-level search strategy. The performance of the proposed B&C algorithm is examined on the basis of computational experiments where datasets containing instances of the MRCPSP with 20 and 30 activities from PSPLIB – a library of data for project scheduling (Kolisch and Sprecher 1997) – are used. Optimal solutions are found for all instances with 20 activities and for 506 out of 552 instances with 30 activities. Moreover, for 5 instances with 30 activities the considered B&C algorithm found solutions with the makespan better than the best one known, and for 23 instances it found solutions worse than those reported in PSPLIB. Unfortunately, these results were not recorded in the PSPLIB files containing the best known solutions.

The computational times of the B&C reported by the authors are smaller than those reported by Sprecher and Drexl (1998) for their B&B, however, taking into account the processor clocks, the B&B performs faster than the B&C. A comparison of the presented exact approaches, except the last one by Zhu et al. (2006), is reported by Hartmann and Drexl (1998). The obtained theoretical and experimental results show that for both precedence tree and mode and delay alternatives enumerating schemes there exist instances for which the sets of schedules generated by these two procedures may differ between themselves, but the mode and extension alternatives enumerating scheme is able to generate the same schedules as the two other approaches. Moreover, the algorithm by Sprecher and Drexl (1998) outperforms the other approaches. And finally, finding optimal solutions for a number of activities greater than 20 is computationally intractable by any of the analyzed algorithms. The last conclusion is still true. The newest exact approach – B&C by Zhu et al. (2006) – was tested on instances from PSPLIB. The algorithm found optimal solutions for all instances with 20 activities, but only 506 out of 552 instances with 30 activities were solved optimally (notice that 307 of those instances can be solved optimally by CPM assuming the shortest duration mode for each activity). Thus, there still exist instances with 30 activities for which optimal solutions have not been found.

21.4 Lower Bounds

The first lower bound for the MRCPSP was proposed by Talbot (1982). It was calculated as the length of the critical path for the project, where the shortest duration mode is assigned to each activity and resource constraints are neglected.

Maniezzo and Mingozzi (1999) propose four other rules for computing the lower bounds. The first one is computed as a solution of a weighted node packing problem on graph $\tilde{G}$, which consists in finding an independent set of $\tilde{G}$ of maximum weight. $\tilde{G} = (V^a, E')$ is an intersection graph, where V^a is a set of non-dummy activities of the project, and $(i, j) \in E'$ if and only if the precedence network does not contain any path from vertex i to vertex j (or from j to i), and for each renewable resource the sum of minimal renewable resource requirements for both activities i and j is smaller than the availability of this resource. The second lower bound, giving results not worse than the previous one, is based on the mathematical formulation (21.7)–(21.16) of the MRCPSP, and is calculated as a solution of the LP-relaxation of the problem where precedence and nonpreemptibility constraints (21.9), (21.11), (21.12) are removed. The resulting LP problem solved to find the lower bound looks as follows: (i) the objective is to minimize $\sum_{\lambda=1}^{\Lambda} z_\lambda$, where $z_\lambda = \sum_{t=1}^{\bar{d}} y_{\lambda t}$ is the total processing time of feasible subset A_λ; (ii) constraints (21.10), (21.13), and (21.16)are the same as in formulation (21.7)–(21.16); (iii) constraints (21.8) are changed to $\sum_{\lambda \in \mathscr{A}_{jm}} \sum_{t=ES_j}^{LC_j} y_{\lambda t} = p_{jm} \cdot x_{jm}$ for $j \in V; m \in \mathscr{M}_j$, and constraints $z_\lambda \geq 0$ for $\lambda = 1, \ldots, \Lambda$ are added; (iv) all other constraints, i.e., (21.9), (21.11), (21.12), (21.14), and (21.15), are removed from the model.

The third and the fourth lower bounds, which give results not worse than the first one and not better than the second one, are based on the same mathematical formulation of the LP-relaxed problem where for the third lower bound equations are replaced by inequalities, and for the fourth lower bound additionally nonrenewable resource constraints are removed. The formulation of the third lower bound looks as follows: (i) the objective is to minimize $\sum_{\lambda=1}^{|\mathscr{A}|} z_\lambda$, where $\mathscr{A} = \{\lambda : \lambda \in \{1, \ldots, \Lambda\}, A_{\lambda'} \not\subseteq A_\lambda, \lambda' \in \{1, \ldots, \Lambda\} \setminus \{\lambda\}\}$ is the set of maximal feasible subsets; (ii) constraints (21.10), (21.13), and (21.16) are the same as in formulation (21.7)–(21.16); (iii) constraints (21.8) are changed to $\sum_{\lambda \in \mathscr{A}_{jm}} z_\lambda \geq p_{jm} \cdot x_{jm}$ for $j \in V; m \in \mathscr{M}_j$; where $\mathscr{A}_{jm}$ is the set of indices of all maximal feasible subsets containing activity j performed in mode m, and constraints $z_\lambda \geq 0$ for $\lambda = 1, \ldots, |\mathscr{A}|$ are added; (iv) all other constraints, i.e., (21.9), (21.11), (21.12), (21.14), and (21.15), are removed from the model. The formulation of the fourth lower bound is the same except for constraints (21.13) which are also removed.

Pesch (1999) uses the Talbot's B&B (1982) as a base algorithm to test various strategies in order to analyze the influence of different settings of the problem parameters on the quality of the enumeration approaches by defining different classes of problem instances. The main conclusion from this paper is that probably no procedure exists that is able to efficiently solve problem instances of different classes. Therefore different implementations are necessary, and they are to be applied to different classes of problem instances.

A destructive lower bound for the MRCPSP with minimal and maximal time lags is presented in Brucker and Knust (2003). The lower bound calculations are based on two methods for proving infeasibility of a given threshold value T for the makespan. The first one uses constraint propagation techniques, while the second one is based on the linear programming formulation by Maniezzo and Mingozzi (1999) which is solved by a column generation procedure. Computational results are reported for several test instances of the multi-mode problem with and without time lags and the single-mode version with time lags.

21.5 Heuristics

Most of the papers on the MRCPSP are devoted to heuristic approaches, which are discussed in this section. We divide the section into three subsections, similarly to the classification made by Kolisch and Hartmann (1999). First, we present some priority rule-based algorithms, next metaheuristics and other heuristics based on local search procedures, and finally some other approaches are discussed. Most of the algorithms presented in this section, especially list algorithms, use one of the two basic procedures as a decoding rule to build a schedule. These are well-known serial and parallel schedule generation schemes (SGS) (see, e.g., Bedworth 1973; Kelley 1963; Kolisch 1996; Chap. 1 of this handbook).

21.5.1 Priority Rule-Based Heuristics

This group of algorithms is dominated by algorithms, where the order in which activities are executed is determined by their priorities calculated on the basis of a given rule. A schedule is generated either once, when the only one priority rule is used (single-pass methods), or at most a few times, when a set of rules is applied (multi-pass methods). Therefore, the priority-rule based heuristics are the fastest algorithms developed for the MRCPSP, but the obtained results are usually worse than those obtained by metaheuristics and other algorithms which visit a larger number of solutions. It should be noted that in the case of the MRCPSP the order of activities is not a sufficient information to build schedule. Essential is also the second part of the solution, which is used to determine an execution mode for each activity. Priority rules are also used in other algorithms in order to build a good initial solution. In the literature, there are several papers concerning such an approach.

Talbot (1982) uses a priority rule in the stage one of his B&B to reorder the list of activities and to provide a good initial solution used to obtain initial upper bounds for completion times of activities. The following eight rules are proposed: maximum average activity duration (MAX ADUR), maximum activity duration (MAX DUR), minimum late finish time (MIN LFT), maximum average resource demand (MAX RD), minimum early finish time (MIN EFT), minimum late finish time reduced by smallest duration (MIN (L-D)), random (RAND), and minimum late finish time reduced by average duration (MIN (L-ADUR)). A small computational experiment (100 problem instances with 10 activities, 3 renewable resources, and 1–3 modes for each activity) shows that random rule is outperformed by all other heuristics, and moreover the best results are obtained using priority rules based on the minimum late finish time.

Another set of priority rules was proposed by Boctor (1993) where the following priority were introduced: minimum total slack (MIN SLK), shortest processing time (SPT), maximum number of immediate successors (MAX NIS), maximum remaining work (MAX RWK), longest processing time (LPT), and maximum number of subsequent candidates (MAX CAN). More information about this approach is presented in Sect. 21.6, because it is used for the special case of the MRCPSP where only renewable resources occur.

Özdamar (1999) uses several priority rules to construct a solution in genetic algorithms. The following rules were not considered in previous approaches: weighted resource utilization and precedence (WRUP), minimum late start time (MIN LST), minimum early start time (MIN EST), and most total successors (MTS).

Lova et al. (2006) consider five new priority rules used to determine the order of activities: minimum activity number (MIN AN), greatest rank positional weight (GRPW), greatest resource demand (GRD), minimum latest start and finish time (MIN LSTLFT), and minimum free slack (MIN FREE).

In the case where at least two activities have the same priority according to a given priority rule another rule have to be applied to resolve the draws. The simplest approach is a rule that uses the activity indices to determine which activity has a higher priority. Other approaches require additional priority rules to be applied in cases where the master rule would make a draw.

Another set of rules is used to determine the execution modes of activities. It includes three rules proposed by Boctor (1993): shortest feasible mode (SFM), the least criticality ratio (LCR), and the least resource proportion (LRP); as well as one rule introduced by Lova et al. (2006) – earliest feasible finish time (EFFT).

Drexl and Grünewald (1993) present a simple stochastic scheduling method named STOCOM. In this method a schedule is generated by selecting activities from the set of eligible activities together with modes assigned to them in a random way. The probability of a choice is either proportional to the weights that are calculated for each activity-mode combination regarding the longest duration mode from all modes of all eligible activities, or proportional to the weights calculated regarding the latest completion times of eligible activities. Computational results show that this method is superior to other deterministic scheduling rules existing when the experiment was performed. Extensions to problems in which resource requirements of activities vary with time, in addition to time-varying supply resource profiles, are discussed as well. Boctor (1996a,b) uses this approach in his computational experiment, but unfortunately it completely failed being unable to find a feasible solution to any of the 240 test instances of the problem. The reason for such a result is that this heuristic may generate feasible or infeasible schedules, scheduling all activities at their earliest start times which are calculated taking into account precedence constraints only. Feasibility with respect to resource constraints is checked solely for the final schedule.

Another simple heuristic approach is proposed by Boctor (1996a) but it is developed for the special case of the MRCPSP with renewable resources only, and is discussed in Sect. 21.6.

The next simple heuristic approach named local constraint based analysis (LCBA) is proposed by Özdamar and Ulusoy (1994) for the MRCPSP with one nonrenewable resource only. In this algorithm the parallel SGS is used as decoding rule. The selection of activities and their assigned modes is made locally at every decision point as long as a complete sequence of schedulable activities is found. The procedure is also adapted to various model extensions such as flexible resource requirement levels. The computational experiment including a set of 95 instances with 20–57 activities, 1–6 renewable and one nonrenewable resource was performed for LCBA and three dispatching rules. The obtained results show that the proposed procedure yields an average increase over the precedence-based lower bound of 59 % which is a better result than this obtained by priority rules which were applied for comparative purposes and yielded an average deviation of 65 %. Unfortunately, the constraint-based approach has two disadvantages. First, the worst-case time complexity of the procedure is exponential. Secondly, it is not suited for solving the MRCPSP with multiple scarce nonrenewable resources.

Finally, the polynomial activity insertion algorithm developed by Artigues and Roubellat (2000) is a simple heuristic that can be used jointly with an activity removal algorithm in the neighbourhood generation mechanism of the local search algorithms applied to the MRCPSP with renewable resources only, and therefore it is discussed in Sect. 21.6.

21.5.2 Metaheuristics and Local Search

The first local search strategy applied to the MRCPSP is an algorithm proposed by Kolisch and Sprecher (1997), where a solution is represented by both a mode assignment list and a list of activity completion times. It consists of three phases. Firstly, in the construction phase, an initial mode assignment is generated and then, if it satisfies nonrenewable resource constraints, a fast heuristic for the resulting RCPSP is used. In the second phase, a local search, that performs a single neighbourhood search on the set of feasible mode assignments, is performed for prescribed number of iterations. In the final intensification phase a schedule with an improved objective function is searched on the basis of the best mode assignment obtained during the previous phase. The computational experiment, where two sets of benchmark instances of the problem with 10 and 30 activities were generated using the ProGen project generator (Kolisch et al. 1995), is used to check the performance of the proposed approach. Both sets contain 640 instances, where in each instance two renewable and two nonrenewable resources, as well as three modes per activity, are considered. The results obtained by the proposed algorithm are compared with the results obtained by a truncated B&B (Talbot 1982) and STOCOM (Drexl and Grünewald 1993). The proposed heuristic is the only one from the three tested approaches that finds feasible solutions for all 10-activity instances and for most 30-activities instances.

From among many various metaheuristics applied to the MRCPSP, genetic algorithm (GA) is the most popular one. The first adaptation of this algorithm to the MRCPSP is presented by Mori and Tseng (1997), but it is developed for a special case of the problem with renewable resources only, and therefore it is discussed in Sect. 21.6.

The next two versions of GA, pure (PGA) and hybrid (HGA), are proposed by Özdamar (1999). A solution in the HGA is represented by two lists: a mode assignment list and a list of priority rules. The position i on the second list denotes a priority rule used for the i-th scheduling decision. The set of the applied priority rules includes: MIN SLK, MIN LFT, SPT, RAND, WRUP, MIN LST, MIN EST, and MTS. As a decoding rule, a forward–backward scheduling (see, e.g., Li and Willis 1992) employing the parallel SGS is used. Solutions of the next population are generated using two-point crossover and uniform crossover operators as well as a mutation operator, which randomly changes one position in the mode assignment list and one position in the list of priority rules. In the PGA a solution is represented by both a mode assignment list and an activity list. The serial SGS is employed as a decoding rule. Offspring solutions are generated using two-point linear crossover, and repair procedure is run when the obtained offspring is not precedence-feasible. A mutation operator for the PGA randomly changes one position in the mode assignment list and pairwise swaps two activities from the activity list, if it is precedence-feasible. In both algorithms, if the offspring is not feasible with respect to any nonrenewable resource, then it is not evaluated. In other words, only solutions with nonrenewable resource-feasible assignments of modes are permitted. The performance of the proposed approaches is evaluated using 536 problem instances with 10 activities, 3 modes per activity, two renewable and two nonrenewable resources as well as 32 instances with 90 activities, two modes per activity, two renewable and two nonrenewable resources. All instances are generated by ProGen and available via Internet at PSPLIB. The obtained results are compared with those reported by Kolisch and Drexl (1997) and show that the HGA outperforms all other algorithms tested in the experiment.

Another genetic algorithm is proposed by Hartmann (2001). For a long time it was considered the best adaptation of GA to the MRCPSP. Before executing the main procedure, the preprocessing rule (described in Sect. 21.3) is applied in order to reduce the search space by adapting the project data. A solution is represented by two lists: a precedence-feasible list of activities and a mode assignment list. Such a representation allows to generate schedules infeasible with respect to nonrenewable resource constraints. In such a case, a penalty function is used to calculate the fitness of an infeasible solution. Offspring solutions are generated using one-point crossover where two different cut points are used, one for the activity list and another one for the mode assignment list, as well as a mutation operator that swaps two adjacent activities on the activity list if it is precedence feasible and randomly changes one mode on the mode assignment list. The ranking method is used as a selection operator. A schedule is constructed applying the serial SGS, and a single-pass or multi-pass local search based on the multi-mode left shift operation is performed on this schedule. Next, if the schedule is improved, an encoding rule is applied to reversely transform the schedule into activity and mode assignment lists. This operation, called inheritance, unfortunately does not significantly improve the obtained results. The author also shows that repetition approach and island model of GA, as well as other selection operators, do not improve the performance of the proposed approach. During the computational experiment where data sets from PSPLIB with 10, 12, 14, 16, 18, 20, and 30 activities are used the best settings for the proposed GA are experimentally chosen, and the performance of the final fine-tuned version of GA is compared with performance of other approaches, namely local search by Kolisch and Drexl (1997), genetic algorithm by Özdamar (1999), simulated annealing by Bouleimen and Lecocq (2003), and the truncated B&B by Hartmann and Drexl (1998). The obtained results show that the new genetic algorithm clearly performs better than other heuristics. The only exception is the truncated B&B which is slightly better for instances with small numbers of activities.

In the next adaptation of genetic algorithm proposed by Alcaraz et al. (2003) a solution is also represented by an activity list and a mode assignment list as well as one additional bit (forward/backward gene) representing the scheduling generation scheme used to build the schedule (serial forward or backward). As in other approaches, preprocessing precedes the main genetic algorithm. The fitness function is calculated either simply as the makespan, if the obtained schedule is feasible, or otherwise as $C_{max} + MFC_{max} - LB_0^{min} + PF$, where C_{max} is the makespan of the current solution, MFC_{max} is the maximum feasible makespan from the current population, LB_0^{min} is the critical path length calculated for the shortest duration modes, and $PF = \sum_{l=1}^{n} \left(\min \left\{ 0, R_k - \sum_{j \in V^a} r_{jkm} \right\} \right)$ for $k \in \mathscr{R}^n$ is a penalty for violating nonrenewable resource constraints. It is shown that this fitness function is better than the one proposed by Hartmann (2001). A so-called two-point forward–backward crossover is proposed in which an offspring is build either from the head or from the tail depending on the value of the parent's forward/backward gene. The mutation operator consists of two-phases: in the first one, activities are reordered in the activity list with a given probability in the way that the activity list after this operation is still precedence-feasible, and in the second phase, modes from the mode assignment list are changed with a given probability. The forward/backward gene may be changed in the first phase. Finally, a random replacement procedure is applied that replaces with a given probability solutions from the current population with other solutions generated randomly. The computational experiment is carried out using PSPLIB data sets containing instances with 10, 12, 14, 16, 18, 20, and 30 activities. The comparison with the local search by Kolisch and Drexl (1997), the genetic algorithm by Hartmann (2001), and the genetic algorithm by Özdamar (1999) is made only for the data set with 10 activities after generating 6,000 solutions, and show that the proposed GA outperforms the local search and GA by Özdamar, but performs slightly worse than Hartmann's GA. More results are presented for the comparison with simulated annealing (SA) by Józefowska et al. (2001). In this case all data sets except the one with 30 activities are considered, and a stop criterion for both the analyzed algorithms is fixed at 5,000 generated solutions. The presented results show that the proposed GA outperforms SA.

Lova et al. (2009) propose a hybrid genetic algorithm (MM-HGA). As in many other algorithms, preprocessing is performed before the start of the main genetic algorithm procedure. A solution is represented in the same way as in GA proposed by Alcaraz et al. (2003) except that one additional bit named serial/parallel gene is used to denote the variant of SGS used to build the schedule. A new fitness function is proposed. It is shown that applying this function in the proposed approach improves its performance compared with two other fitness functions proposed by Hartmann (2001) and by Alcaraz et al. (2003), respectively. The next generation is obtained using two-point crossover and mutation operators. Mutation is applied to both the activity list and the mode assignment list, as well as to the two additional genes. On the activity list a random shift is performed. Mutation applied on the mode assignment list depends on the feasibility of the solution with respect to nonrenewable resources. For a feasible solution modes are changed randomly with a given probability that is the same for all activities. If a solution is infeasible then a so-called massive mutation is applied: a mode is randomly chosen for a randomly chosen activity until either a feasible mode assignment is found or modes are already chosen for all activities. The forward/backward and serial/parallel genes are randomly changed with a given mutation probability. A 2-tournament selection operator with elitism is applied to generate the next population. Moreover, two additional mechanisms are used: a random replacement of some solutions from the current population, and a multi-mode forward–backward improvement. The data sets from PSPLIB with instances containing 10, 12, 14, 16, 18, 20, and 30 activities are used in the computational experiment. The performance of MM-HGA is compared with local search (Kolisch and Sprecher 1997), genetic algorithms (Alcaraz et al. 2003; Hartmann 2001; Özdamar 1999), and simulated annealing (Bouleimen and Lecocq 2003) for the instances with 10 activities. Moreover, for all data sets, except the one with 30 activities, MM-HGA is compared with simulated annealing by Józefowska et al. (2001) and genetic algorithm by Alcaraz et al. (2003). The obtained results show that MM-HGA outperforms the other heuristics.

Yet another implementation of a genetic algorithm is presented by Van Peteghem and Vanhoucke (2010), where, similarly to other approaches, preprocessing runs before the start of the main algorithm. They use a so-called bipopulation version of GA in which the idea of justifying the schedule to the right or to the left is adapted from Valls et al. (2005) resulting in two different populations of the same size: a population POP_R that contains right-justified schedules only, and a population POP_L that contains left-justified schedules only. A solution is represented by two lists: a topological ordering random key list and a mode assignment list. A random key representation is a list of priority values, and when topological ordering (see, e.g., Valls et al. 2003) is employed, priorities preserve precedence constraints, i.e., if i precedes j then the priority value for i is smaller than the one j. The forward-backward scheduling with the serial SGS is used to construct a schedule.

The forward procedure is applied to the left-justified population, and is used to build a right-justified schedule. Next, the completion times of activities are used as the priority values for the random key representation, and each activity is scheduled as late as possible. Genetic operators are then applied to the right-justified schedules, and the backward procedure runs for the obtained population of right-justified schedules. A mode improvement procedure runs together with the serial SGS. This procedure is applied to activities of the project with a given probability, and checks for a chosen activity if the change of the assigned mode leads to an earlier completion time of this activity without increasing the penalty function value. The penalty function is calculated for the nonrenewable resources consumption that exceeds resource availability limitations. Two fitness functions are investigated: the one proposed by Hartmann (2001) and the other one proposed by Alcaraz et al. (2003). The offspring solutions are generated using 2-tournament selection, one-point crossover, and two mutation operators (one acting on the mode assignment list, and the other acting on a random key vector). In the computational experiment the data sets from PSPLIB containing instances with 10, 12, 14, 16, 18, 20, and 30 activities are used. The performance of the proposed algorithm is compared with the performance of other approaches including the genetic algorithms by Özdamar (1999), by Hartmann (2001), and by Alcaraz et al. (2003), as well as the local search by Kolisch and Drexl (1997), and the simulated annealing by Józefowska et al. (2001). The obtained results show that the considered genetic algorithm approach is one of the most powerful heuristic developed up to now. The average relative deviation from optimum is less than 0.5 % for all data sets, and the percentage of optimal solutions found is from about 88 % for the data set with 20 activities up to almost 100 % for the data set with 10 activities.

The second metaheuristic commonly used for the MRCPSP is simulated annealing. Up to now, the best adaptation of this approach to the considered problem is proposed by Józefowska et al. (2001). A solution is represented by a precedence-feasible activity list and a mode assignment list. A schedule is constructed using the serial SGS. Similarly to other algorithms, preprocessing is performed before the start of the main algorithm in order to reduce the search space. Two versions of SA are considered: with and without penalty function. A penalty for the violation of nonrenewable resource constraints is added to the upper bound of the makespan in the version with penalty function. Neighbour solutions are generated by a random shift of a chosen activity and/or by changing randomly the chosen mode. In the version without penalty function a change of mode has to result in the mode assignment feasible with respect to the nonrenewable resource constraints. The search process is controlled by the adaptive cooling scheme. Performance of both the proposed approaches is examined during a computational experiment where data sets from PSPLIB with 10, 12, 14, 16, 18, 20, and 30 activities are used. The obtained results indicate that the version with penalty function is the better one and comparable with the GA proposed by Hartmann (2001).

Another implementation of simulated annealing is proposed by Bouleimen and Lecocq (2003) who use the same decoding rule and solution representation (except that the mode assignment should be resource-feasible). Neighbour solutions are generated using a two-phase procedure. In phase one, a resource-feasible change of a randomly chosen activity is made. Then for a fixed resource-feasible mode assignment, in the second phase, nonrenewable resource constraints are removed, and an improved schedule is searched by generating a neighbour activity list using a random shift operation. The search process is controlled by a cooling scheme where the control parameter is changed according to the geometrical progression, and "reheating" is allowed for instances where the search process prematurely traps in local optimum. A computational experiment is performed for the same data sets as in Józefowska et al. (2001), but no comparison with any other approach is made.

In the first adaptation of tabu search (TS) approach to the MRCPSP (Nonobe and Ibaraki 2002) a solution is represented by an activity list and a mode assignment list. Neighbour solutions are generated either by a change on the mode assignment list, or by shifting some activities on the activity list. The schedule is generated by applying a scheme proposed by the authors, named CONSTRUCT, but further observations show that in some cases an optimal solution cannot be obtained by this procedure. Computational results are only reported for data set with 30 activities from PSPLIB, but no comparison to any other approach is presented.

Van Peteghem and Vanhoucke (2011) propose a scatter search (SS) in which they incorporate the resource scarceness characteristics in different improvement methods, each tailored to the specific characteristics of different renewable and nonrenewable resource scarceness values methods. A solution is encoded using random key representation for ordering the activities, and a mode assignment list. A schedule is constructed using the serial SGS. The next solutions are generated using two-point crossover operator. Several methods are used to improve the obtained schedules including feasibility improvement method, critical path improvement method, and work content improvement method, as well as local search. Some other mechanisms are used in different phases of the proposed algorithm. The computational experiment is divided into two parts. The first one is used to check the influence of different improvement methods on the algorithmic parameter settings and is performed using new data sets especially generated for this purpose by RanGen project scheduling instances generator (Vanhoucke et al. 2008). In the part of computational experiment carried out in order to compare the proposed approach with other existing approaches the instances with 10, 12, 14, 16, 18, 20, and 30 activities from PSPLIB are used. A set of algorithms used in the computational experiment include: simulated annealing by Józefowska et al. (2001), scatter search by Ranjbar et al. (2009), and four different genetic algorithms (Alcaraz et al. 2003; Lova et al. 2009; Tseng and Chen 2009; Van Peteghem and Vanhoucke 2010). The presented results show that the proposed algorithm is the most powerful heuristic developed up to now. The average relative deviation from optimum is less than 0.35 % for data sets with up to 20 activities, and the percentage of optimal solutions found is from about 88 % for the data set with 20 activities up to 100 % for the data set with 10 activities. These results show that a good algorithm should adapt itself to the instance of the considered problem.

Another group of metaheuristics used to solve the MRCPSP are evolutionary algorithms. They include a population learning algorithm by Jędrzejowicz and

Ratajczak (2006), evolutionary algorithm by Elloumi et al. (2006) and Elloumi and Fortemps (2010), differential evolution algorithm by Damak et al. (2009), genetic local search algorithm by Tseng and Chen (2009), shuffled frog-leaping algorithm by Wang and Fang (2011), and estimation of distribution algorithm by Wang and Fang (2012).

Jędrzejowicz and Ratajczak (2006) propose a population learning algorithm (PLA) where a solution is represented by a list of objects. Each object refers to one activity and includes some important information about it. At the beginning of PLA the preprocessing procedure runs. The main PLA consists of three learning stages. At the first learning stage three procedures are used. They are: the crossover operator, simple local search algorithm (LSA), and the RHIA procedure that is based on the idea of improving the resource utilization in homogeneous intervals proposed in Valls et al. (2004). At the second learning stage, two evolutionary operators, crossover and mutation, as well as two heuristics procedures LSA and EPTA (exact precedence tree algorithm – based on precedence tree branching rule) are used. And finally, at the third learning stage two heuristics LSA and EPTA are applied. If the new generated solution is infeasible regarding nonrenewable resource constraints then a procedure trying to improve the usage of nonrenewable resources is run. The performance of PLA is validated for standard data sets from PSPLIB. The results obtained during the computational experiment are compared with those presented in Józefowska et al. (2001), and show that these two approaches are very similar in terms of performance.

Elloumi et al. (2006) develop an evolutionary algorithm which, after preprocessing, starts with generating an initial population of individuals represented by an activity list and a mode assignment list. For each individual the fitness function is calculated using rank-based assignment method, clustering heuristics for density computation, and penalty for violation of the nonrenewable resource constraints. Neighbourhood is generated using one-point crossover, mutation operating on both lists independently, ranking, and the roulette wheel selection methods, as well as the left shift procedure. The computational experiment, where instances from PSPLIB with 10, 12, 14, 16, 18, 20, and 30 activities are used, shows that the proposed approach is comparable to the genetic algorithm by Alcaraz et al. (2003), and performs better than the genetic algorithms by Hartmann (2001) and by Özdamar (1999), the truncated B&B by Sprecher and Drexl (1998), simulated annealing algorithms by Józefowska et al. (2001) and by Bouleimen and Lecocq (2003), and local search by Kolisch and Drexl (1997).

Elloumi and Fortemps (2010) propose another version of the evolutionary algorithm (EA) where a solution is represented by an activity list and a mode assignment list. Solutions infeasible with respect to nonrenewable resources are penalized. A penalty function is treated as the second criterion which has to be minimized. Thus, the MRCPSP becomes a bi-objective problem. The preprocessing procedure is executed before the beginning of the EA. In each iteration of the algorithm a local search procedure is applied on the infeasible solutions in order to improve them by trying to reduce their penalties. Then a half of the population is chosen for one-point crossover operation adapted from the GA by Hartmann (2001), and the obtained offspring solutions undergo the mutation with a given probability. For the obtained solutions the fitness function is calculated for both makespan and penalty. Next the rank-based fitness assignment method and clustering heuristics are used. Next a selection operator is applied to choose individuals for the next population. In the computational experiment instances from data sets for problems with 10, 12, 14, 16, 18, 20, and 30 activities from PSPLIB are used. The obtained results are compared with those reported for simulated annealing (Bouleimen and Lecocq 2003; Józefowska et al. 2001), genetic algorithms (Alcaraz et al. 2003; Hartmann 2001; Özdamar 1999), local search (Kolisch and Drexl 1997), scatter search (Ranjbar et al. 2009), particle swarm optimization (Jarboui et al. 2008), differential evolution (Damak et al. 2009) and truncated B&B (Sprecher and Drexl 1998). The proposed EA with K-means based clustering heuristic and different ranks clusters numbers outperforms all other algorithms except GA by Hartmann (2001).

Damak et al. (2009) propose a differential evolution algorithm, where a solution is represented by a mode assignment list and an activity list which needs not be precedence-feasible. Neighbour solutions are generated using two operators: mutation and crossover. Selection operator uses the values of the objective function which is penalized for solutions infeasible with respect to the nonrenewable resources. The performance of this algorithm is evaluated on the basis of a computational experiment where instances from PSPLIB with 10, 12, 14, 16, 18, and 20 activities are used. The obtained results are compared with the results obtained by two other approaches only: simulated annealing by Bouleimen and Lecocq (2003) and particle swarm optimization by Jarboui et al. (2008). Unfortunately, they are not compared with the results obtained by other more efficient algorithms.

A two-phase genetic local search algorithm where the same genetic local search algorithm runs with different initial populations for both phases for different search purposes is proposed by Tseng and Chen (2009). In the first phase, the initial population is generated randomly, and the set of good solutions (so-called elite set) is searched. In the second phase, the initial population contains mainly solutions from the elite set and the purpose of this phase is to search more thoroughly within the regions located by the solutions from the elite set. Similarly to other approaches, preprocessing is executed before the start of the main procedure. A single solution is represented as in many other algorithms by an activity list and a mode assignment list. The fitness function is calculated in the same way as in Alcaraz et al. (2003), and the proposed forward–backward local search method is used to transform a given solution to the standard representation. After this transformation each schedule has exactly one solution representation. Neighbour solutions are generated using two-point crossover proposed by Alcaraz et al. (2003) and its slightly modified version, as well as two mutation operators which allow to diversify the population lightly or heavily, respectively. The first mutation operator is taken from Alcaraz et al. (2003), whereas the second one is a new concept developed by the authors. Selection is made using ranking and 2-tournament methods. The computational experiment carried out using instances from PSPLIB with 10, 12, 14, 16, 18, 20, and 30 nondummy activities was executed to compare the performance of the proposed approach with five other algorithms: local search by Kolisch and Drexl (1997), simulated annealing by Józefowska et al. (2001), and three versions of genetic algorithms – by Özdamar (1999), by Hartmann (2001), and by Alcaraz et al. (2003). The presented results show that the proposed two-phase genetic local search algorithm outperforms the other approaches.

Wang and Fang (2011) propose one of the newest metaheuristics named the shuffled frog-leaping algorithm (SFLA). The SFLA combines the benefits of memetic algorithm and particle swarm optimization. In the SFLA each individual is called a frog, and the whole population is divided into a number of memeplexes. The different memeplexes are considered as different cultures of frogs, and each memeplex performs a local search by moving around in the search space with the help of neighbourhood frogs' social experiences in the same memeplex. In the presented adaptation of the SFLA to the MRCPSP a solution is represented by a so called extended multi-mode activity list (EMAL) which consists of four lists: an activity list, a mode assignment list, a list of activity start times, and a list of activity finish time. The serial SGS is used as the decoding rule. Similarly to the other approaches, the preprocessing procedure is executed before the start of the metaheuristic. At the beginning of the main procedure the regret-based biased random sample with LFT priority rule is used to generate an initial population which is next improved by the forward-backward scheduling. For each generation the population is partitioned into a given number of memeplexes, and for each memeplex a submemeplex is generated using the two-point crossover. If the obtained frog is better than the worst one in the memeplex it replaces this worst one, otherwise a new frog is generated using two-point crossover operator for another pair of parents. If for a given number of such crossover operations a generated frog is still worse than the worst frog in the memeplex, then the worst frog is replaced by the randomly generated one. A combined local search including permutation based local search and forward-backward improvement is used to explore the neighbourhood of the frog. The instances with 10, 12, 14, 16, 18, 20, and 30 activities are used in the computational experiment where the results obtained for the considered algorithm are compared with those reported for other approaches including simulated annealing (Bouleimen and Lecocq 2003; Józefowska et al. 2001), genetic algorithms (Alcaraz et al. 2003; Hartmann 2001), scatter search (Ranjbar et al. 2009), evolutionary algorithm (Elloumi and Fortemps 2010), truncated B&B (Sprecher and Drexl 1998), and particle swarm optimization (Jarboui et al. 2008). The obtained results show that the proposed approach outperforms other approaches used in the experiment.

Estimation and distribution algorithm (EDA) is applied by Wang and Fang (2012). It is a kind of stochastic optimization algorithm based on statistical learning, where new individuals are generated by predicting the most promising area based on the distribution of elite individuals of former generations in the search space. In this algorithm a solution is encoded using a mode assignment and an activity list. A novel probability model and updating mechanism are developed to help identifying the most promising area. The forward-backward iteration and the permutation based local search method are applied to the best individuals to exploit their neighbourhood. Some other mechanisms known from other algorithms are implemented as well. They include: preprocessing, the serial SGS, and a penalty function for violating nonrenewable resource constraints. The results of the computational experiment where instances from PSPLIB with 10–30 activities are used show that the proposed algorithm performs better than the simulated annealing by Józefowska et al. (2001), genetic local search by Tseng and Chen (2009), genetic algorithm by Alcaraz et al. (2003), scatter search by Ranjbar et al. (2009), and evolutionary algorithm by Elloumi and Fortemps (2010). It is outperformed by genetic algorithms by Lova et al. (2009) and Van Peteghem and Vanhoucke (2010), as well as by artificial immune system by Van Peteghem and Vanhoucke (2009).

Another quiet large group of metaheuristics frequently used for the MRCPSP are algorithms commonly named swarm intelligence algorithms. This group includes ant colony optimization (Chiang et al. 2008; Chyu et al. 2005; Li and Zhang 2013; Zhang 2012), and particle swarm optimization (Zhang et al. 2006; Jarboui et al. 2008) approaches.

Ant colony optimization (ACO) is a class of algorithms modeled on the behaviour of an ant colony. The 'artificial ants' similarly to the natural ants locate the best solutions by moving through the search space and record their positions and the quality of their solution, which may be used by other 'artificial ants' in the next iterations of the algorithm. The first adaptation of this method to the MRCPSP is a hybrid ant colony optimization (H-ACO) by Chyu et al. (2005) in which the solution construction mechanism of B&B and ant colony optimization (ACO) algorithms are hybridized. First, B&B is used to find a set of feasible (i.e., satisfying nonrenewable resource constraints) mode assignments. For each feasible mode assignment the following procedure is applied. Firstly, CPM is used to calculate the schedule length with neglected resource constraints. Secondly, a disjunctive rule is applied to add some arcs to the AoN network in order to remove resource conflicts. Thirdly, a new project makespan is calculated for the new AoN network representing the structure of the project. Next, a given number of mode assignments with the best potential values of the makespan obtained during the third step of the above mentioned procedure are chosen for the second phase of H-ACO where the ACO algorithm with the forward–backward improvement is applied to each chosen mode assignment. Finally, the best schedules obtained by ACO for each chosen mode assignment are compared, and the best one is chosen as a solution of the considered instance of the problem. H-ACO is compared only with SA by Józefowska et al. (2001) using the standard data sets from PSPLIB. The obtained results show that H-ACO outperforms SA.

Another ACO algorithm called ACO-MRCPSP is proposed by Chiang et al. (2008). In this approach a solution is represented by two lists: a mode assignment list and a random key list. The proposed algorithm consists of four phases: (1) preprocessing; (2) initialization where some control parameters are calculated for a given instance of the problem; (3) construction where artificial ants are guided to construct feasible solutions; and (4) feedback where the best schedule found so far is taken as the guide for properly reinforcing the pheromone concentration on the links of specific paths in the construction graphs. The performance of the ACO-MRCPSP is checked on the basis of a computational experiment where instances from

PSPLIB with 10, 20, and 30 nondummy activities are used. The obtained results are compared with the results provided for four other state-of-the-art algorithms: simulated annealing by Józefowska et al. (2001), genetic algorithms by Özdamar (1999) and by Alcaraz et al. (2003), and evolutionary algorithm by Elloumi et al. (2006). The presented results show that the ACO-MRCPSP outperforms the other approaches.

The newest ACO approach is presented by Li and Zhang (2013), in which a solution is represented by an activity list and a mode assignments list, and therefore two levels of pheromones are considered with regard to the solution (one for each list). Elitist-rank strategy and nonrenewable resource-constraint are incorporated into the updating procedure of the pheromones. The serial SGS is used as the decoding rule. In the computational experiment data for instances with 10, 12, 14, 16, 18, 20, and 30 activities from PSPLIB are used. The proposed algorithm is compared with simulated annealing by Bouleimen and Lecocq (2003), genetic algorithm by Hartmann (2001), particle swarm optimization by Zhang et al. (2006) and Jarboui et al. (2008). The obtained results show that the proposed approach is better than those proposed by Zhang et al. (2006) and Bouleimen and Lecocq (2003), but are worse than algorithms proposed by Hartmann (2001) and Jarboui et al. (2008). The same algorithm but without the results of the computational experiment is presented by Zhang (2012).

In the particle swarm optimization (PSO) a population called swarm consists of candidate solutions called particles. Each particle is moved over the search space according to a few simple rules taking into account the position of the swarm. The particle is characterized by its position and its velocity, and the best known positions of both the particle and the swarm are memorized for the next iterations. The first PSO adaptation applied to the MRCPSP is proposed by Zhang et al. (2006), where a solution is represented by two particles: the first one in the form of a random key representation of activity priorities, and the second one in the form of a mode assignment list. A new solution is generated using for each particle two formulas calculating both a new position, and a new velocity of a particle taking into account the position and the velocity of the best particle of the swarm. A repairing procedure is applied for solutions with mode assignments infeasible according to nonrenewable resource constraints. The serial SGS is used as a decoding rule. A comparison with other heuristics including the truncated B&B (Sprecher and Drexl 1998), simulated annealing (Bouleimen and Lecocq 2003), and genetic algorithm (Hartmann 2001) is based on a computational experiment where instances of projects with 10, 12, 14, 16, 18 and 20 activities available in PSPLIB are used. The results show that the proposed PSO performs a little bit worse than GA, and better than SA. Moreover, it is outperformed by truncated B&B for small number of activities 10 and 12.

Another PSO approach named combinatorial particle swarm optimization (CPSO) is used by Jarboui et al. (2008) to generate solutions of the mode assignment subproblem of the MRCPSP. Next, for a fixed mode assignment, a local search algorithm is used to find suboptimal solutions of the resulting RCPSP. A computational experiment is carried out for seven data sets from PSPLIB containing instances with 10–30 activities. The results obtained by the proposed approach are compared with the results generated by two other approaches, namely the simulated annealing by Bouleimen and Lecocq (2003) and the particle swarm optimization by Zhang et al. (2006), and show that the proposed approach performs better than the two other approaches.

Van Peteghem and Vanhoucke (2009) propose artificial immune system (AIS), where in order to generate a good and diverse initial population a controlled search procedure is used, which is based on an observed link between predefined profit values and the makespan of the mode assignment, and which leads the search process more quickly to more interesting search regions. The performance of this approach is verified on the basis of a computational experiment, and its high effectiveness is proved by the obtained results.

Coelho and Vanhoucke (2011) propose a new approach which splits the MRCPSP into two interrelated subproblems: the mode assignment and single mode resource-constrained project scheduling. The mode assignment subproblem is solved using a satisfiability (SAT) problem solver which finds a feasible mode assignment which is then passed to the scheduling subproblem where, for a fixed assignment of modes to activities, the resulting RCPSP is solved using the best metaheuristics developed for this problem. It is shown that if SAT solver is used for the mode assignment subproblem, then the size of the memory needed to run this algorithm grows exponentially with the size of the problem (number of activities and/or number of nonrenewable resource constraints). Several small improvements are implemented to solve this subproblem in less memory. In the second subproblem the decomposition based genetic algorithm of Debels and Vanhoucke (2007) is used. In the computational experiment data sets with 10, 12, 14, 16, 18, 20, and 30 activities from PSPLIB are used. The obtained results are compared with results obtained for genetic algorithms by Alcaraz et al. (2003), Lova et al. (2009), and Van Peteghem and Vanhoucke (2010). The analysis of these results show that the proposed approach is better than GA by Alcaraz et al. (2003), but is outperformed by both the other approaches.

21.5.3 Other Approaches

Apart from the methods described in the two previous subsections, also other approaches have been developed for the MRCPSP.

Setting up a limit on computational time of the B&B, it can be treated as a heuristic. Such an approach is proposed by Talbot (1982), Patterson et al. (1989), as well as Sprecher and Drexl (1998).

Maniezzo and Mingozzi (1999) use their mathematical formulation (21.7)–(21.16) of the MRCPSP to develop a new heuristic based on the Benders decomposition and named it HBEND. The LP-relaxed problem in the form used to calculate the third lower bound proposed by them is decomposed into a master problem and a subproblem, which run iteratively. At each iteration the master problem constructs an RCPSP instance assigning modes to activities in such a way that nonrenewable resource constraints hold. Next, the subproblem heuristically finds a valid lower bound for this instance of the RCPSP. This lower bound is then used to obtain a valid Benders cut that is added to the master problem in the next iteration. A number of the best RCPSP instances obtained from the master problem are memorized and solved optimally using the B&B algorithm for the RCPSP developed by Mingozzi et al. (1998). It is done at the end of HBEND to improve the quality of the MRCPSP upper bound. This approach is compared with the local search approach by Kolisch and Drexl (1997), and the truncated B&B by Sprecher and Drexl (1998) on the basis of results obtained during the computational experiment where test instances with 10, 20, and 30 activities from PSPLIB are used. The obtained results show that HBEND requires a number of cuts before achieving good results. Moreover, it outperforms both the considered algorithms (except results for the set with ten activities where HBEND is outperformed by the B&B).

HBEND is once more presented in Boschetti and Maniezzo (2009), where it is compared with other well performing algorithms, namely the truncated B&B by Sprecher and Drexl (1998), the genetic algorithm by Hartmann (2001), two versions of simulated annealing proposed respectively by Józefowska et al. (2001) and by Bouleimen and Lecocq (2003), and the particle swarm optimization (PSO) by Zhang et al. (2006). The results of the computational experiment obtained for instances from data sets with 10 and 20 activities available in PSPLIB show that HBEND is outperformed by the other heuristics except the truncated version of B&B.

21.6 Special Cases and Extensions

Some special cases as well as extensions of the basic MRCPSP model are presented in several papers. In this section we shortly describe these models together with algorithms proposed to solve them. A comprehensive survey of different variants and extensions of the RCPSP, including variants and extensions of the MRCPSP, is presented by Hartmann and Briskorn (2010).

21.6.1 Special Cases

The most commonly considered special case of the MRCPSP is its version with renewable resources only. Algorithms developed for this model of the MRCPSP cannot be used for the model with nonrenewable and doubly constrained resources, since they do not guarantee that the obtained solution is feasible with respect to the nonrenewable resource constraints. Such a problem is studied by Elmaghraby (1977), Boctor (1993, 1996a,b), Knotts et al. (2000), Artigues and Roubellat (2000), Gagnon et al. (2005), and Lova et al. (2006). Other special cases are the discrete time-cost tradeoff problem (DTCTP) (see Chap. 29 of this handbook) and the discrete time-resource tradeoff problem (DTRTP), as well as the resource-constrained project scheduling problem (RCPSP).

Elmaghraby (1977) was the first one who considered multiple operating modes of activities in the project scheduling problem. The objective of this problem is to minimize both costs and the makespan. The set of constraints is formulated, and an example of such a problem is presented.

Boctor (1993) compares 21 priority rule based heuristics on the basis of a computational experiment consisting of 240 problems with 50 and 100 activities, and 1, 2, and 4 renewable resources (nonrenewable and doubly constrained resources are not considered). The following priority rules are used for ordering activities: MIN SLK, MIN LFT, SPT, MAX NIS, MAX RWK, LPT, and MAX CAN. Draws, if occur, are broken by choosing the activity with the smallest number. Modes are selected using SFM, LCR, and LRP heuristics. The resulting heuristics that are examined in a computational experiment appear as possible combinations of any priority rule for activities and any mode assignment rule. The results of the computational experiment show that the best combination of heuristic rules are the following: MIN SLK-SFM, MAX RWK-SFM, MAX CAN-SFM, and MIN LFT-SFM. Another heuristic approach for the same problem is also proposed by Boctor (1996a). This heuristic uses forward and backward parallel scheduling of schedulable and nondominated activity-mode combinations. An activity-mode combination is a subset of the set of eligible activities where for each activity a mode is assigned. Such a combination is called schedulable at a given time period if the resources available (nonallocated) during this period and in succeeding periods allow to execute all activities from the corresponding subset in the assigned modes. An activity-mode combination is dominated by another combination if at least one of the following conditions holds: (i) the first combination is a subset of the second one; (ii) both combinations are identical except that the mode assigned to one activity in the second combination is shorter than the mode for the same activity in the first combination. The results of the computational experiment for the same set of 240 problems show that the proposed approach is better than the previously examined priority rules. The same problem is considered once more by Boctor (1996b) who uses a simulated annealing algorithm to find suboptimal schedules. A solution is represented by an activity list only. Modes are assigned to activities during the execution of the serial SGS procedure that is used as a decoding rule. For a given activity, a mode that guarantees the earliest precedence- and resource-feasible completion time of this activity in the generated schedule is selected. Neighbour solutions are generated by shifting a randomly chosen activity to a new precedence-feasible position in the activity list, chosen in a random way. The search process is controlled by a cooling scheme where the control parameter is changed according to the geometrical progression. The computational experiment is once more carried out for all 240 problems from the Boctor's benchmark set. The obtained results show that the proposed approach outperforms all heuristics proposed previously by Boctor (1993, 1996a).

Mori and Tseng (1997) propose a genetic algorithm for the MRCPSP without nonrenewable or doubly constrained resources. A direct representation of a schedule is used. It contains information about the activity, the assigned mode, the priority, and the calculated start and completion times of this activity. The initial population is built by setting activities in an ascending order and randomly choosing a mode for each activity. The priority is determined randomly for the activity order interval, and start and completion times of each activity are calculated. A crossover operator where one parent is always the best solution in the current population is used to generate the offspring solutions together with two mutation operators. In the first one, a set of activities is chosen and then a mode is randomly chosen for each selected activity. In the second one, a new solution is constructed using the same method as in the initialization phase. A new generation is built of: the 20 best solutions from the previous generation, 10 solutions generated using a crossover operator, 7 solutions generated using a mutation operator, and 3 solutions generated randomly. In the computational experiment 700 instances of the problem with 10, 20, 30, 40, 50, 60, or 70 activities, 2 to 4 modes for each activity, and 4 renewable resources are generated using a Random Activity Network Generator proposed by Demeulemeester et al. (1993). The proposed GA is compared with STOCOM (Drexl and Grünewald 1993) that generates 100 feasible solutions for each test problem. The obtained results show that the proposed GA is significantly better than STOCOM, especially for larger problems.

The next approach to the MRCPSP without nonrenewable resources is presented by Knotts et al. (2000) who propose to use the agent technology that is known from artificial intelligence. For each activity of the project one agent is created that is responsible for acquiring the resources required by this activity. Agents try to find the best solution during the simulation process in which they act according to some rules which determine their behavior. The performance of the approach is validated on the basis of a computational experiment in which an extended data set from (Maroto and Tormos 1994) is used, and the obtained results are compared with the results obtained by commercial software. These results show that the considered approach performs better than some commercial applications but is outperformed by others.

Artigues and Roubellat (2000) develop a polynomial activity insertion algorithm, which is used to insert an activity into an existing schedule for problems with minimization of the maximum lateness. This algorithm can also be used to generate neighbour solutions in local search methods applied to the MRCPSP with renewable resources only and minimization of the makespan. In such an application this insertion algorithm must be used jointly with an activity removal algorithm. Unfortunately, no computational results are reported for the application of this method to the MRCPSP.

Another approach proposed by Gagnon et al. (2005) for the MRCPSP without nonrenewable resources is based on tabu search. A solution is represented by an activity list and a mode assignment list, and seven operators are proposed to generate neighbour solutions. A computational experiment is performed using data sets by Boctor (1993). The proposed approach is compared with simulated annealing (Boctor 1996b) only.

Lova et al. (2006) consider several single-pass and multi-pass heuristics based on priority rules for the MRCPSP with renewable resources only. They analyze three components of such heuristics: schedule generation scheme, priority rule, and mode selection rule on the basis of a computational experiment in which Boctor's (1993) data sets are used. The analyzed schedule generation schemes are serial and parallel; priority rules include: MIN EFT, MIN EST, MAX DUR, MIN LFT, MIN LST, MIN SLK, MAX NIS, MAX RWK, SPT, MIN AN, GRPW, GRD, MIN LSTLFT, and MIN FREE; and earliest feasible finish time (EFFT) is used as a mode selection rule. The results show that the serial SGS greatly outperforms the parallel SGS, and the multi-pass heuristic combining eight priority rules (LSTLFT, LFT, LST, and RWK with both versions of SGS) gives the best results.

Ranjbar et al. (2009) present a new hybrid metaheuristic algorithm based on scatter search (SS) and path relinking methods for the DTRTP with multiple resources. The resulting problem denoted as MDTRTP is the DTRTP with multiple renewable resources, each with time-resource tradeoffs. In fact, the MDTRTP becomes a special case of the MRCPSP with the absence of nonrenewable resources. In the proposed SS algorithm for the MDTRTP, they use path relinking concepts to generate children from parent solutions, in the form of a new combination method. They also incorporate new strategies for diversification and intensification to enhance the search, in the form of local search and forward-backward scheduling, based on so-called reverse schedules, with the activity dependencies reversed. The proposed algorithm is also modified to tackle the RCPSP and MRCPSP. The performance of the algorithm is tested on four datasets, which show that in most cases it outperforms the other heuristic approaches presented in the literature.

21.6.2 Extensions

Apart from the special cases discussed above, also some extensions of the MRCPSP have been considered in several papers.

An extension of the MRCPSP to its version with generalized precedence relations (also called time windows), denoted in the three-field classification $\alpha\,|\beta|\,\gamma$ for project scheduling problems as $MPS\,|\,temp|\,C_{max}$ or using the acronyms as the MRCPSP-GPR (or MRCPSP/max), is studied in several publications. Exact approaches based on the B&B method are proposed in De Reyck and Herroelen (1998), Dornorf (2002), and Heilmann (2003). Some heuristic algorithms are proposed by De Reyck and Herroelen (1999) who use a hybrid of tabu search and a truncated version of their B&B, by Heilmann (2001) who proposes a multi-pass priority rule approach with back planning which is based on an integration approach and embedded in random sampling, and by Calhoun et al. (2002) who implement tabu search. Moreover, Van Hove and Deckro (1998) propose a B&B approach for the MRCPSP with minimal time lags only. An exhaustive review of some project scheduling problems with time windows may be found in Neumann et al. (2002). Barrios et al. (2011) propose for the MRCPSP-GPR a so-called double genetic algorithm which outperforms other state-of-the-art approaches in medium and large instances. This version of a genetic algorithm consists of two-phases. In the first phase algorithm searches for the best modes of the activities, and in the second phase the makespan is minimized. For each phase different parameters and mechanisms are defined including representation, fitness, operators, etc. Ballestín et al. (2013) propose a heuristic for this problem called SA-EVA that is a combination of simulated annealing and evolutionary algorithm. The problem is divided into two parts, which are solved in two successive phases: a mode assignment phase, and a RCPSP with minimum and maximum time lags phase. First, the best mode vector is searched using a simulated annealing algorithm. Then the evolutionary algorithm EVA designed in Ballestín et al. (2011) is executed in order to search for the best start time vector. The computational results show that SA-EVA outperforms the state-of-the-art algorithms for medium and large instances. It is also proved that looking for the best modes can be very useful in the optimization of the MRCPSP/max.

An extended version of the MRCPSP with so-called mode identity constraints is considered by Salewski et al. (1997). The resulting problem is called the mode identity resource-constrained project scheduling problem (MIRCPSP), and is motivated by real-world situations where several activities should be performed in the same way, i.e., by allocating them the same resources. Practical examples of such a problem occur in an audit staff scheduling, timetabling, course scheduling, etc. Formally, in this problem the set of all project activities is partitioned into several disjoint subsets, and all activities belonging to the same subset have to be performed by the same resources. The time and cost of executing activities from such a subset depend on the resources assigned. Moreover, for each activity a deadline, a ready time, and a set of mode-dependent finish-to-start time lags with direct predecessors are defined. A mathematical model of the problem is formulated, and the $\mathcal{NP}$-hardness in the strong sense is proved. A two-phase heuristic is used to find a good feasible schedule. In phase one, for each subset of activities a mode is selected randomly. In phase two, a solution is built by scheduling randomly chosen activities from the eligible set. Afshar-Nadjafi et al. (2013) propose a genetic algorithm for this problem. As in many other metaheuristic approaches, the preprocessing procedure runs before the main procedure. A solution is represented by an activity list and a reduced mode assignment list (with size equal to the number of activities' subsets). An initial population is generated randomly and offspring solutions are generated using a one-point crossover operation, which acts independently on both lists, and a mutation operator which is a shift operation for the activity list and a mode change for the list of mode assignments. A local search is performed for each offspring solution using a structure called Memory Vector. A computational experiment is carried out using instances of the problem with 20, 25, 30, 35, 60, and 90 activities, 2 renewable resources, and 3 modes, generated by ProGen/πx (Drexl et al. 2000). Rahimi et al. (2013) propose for the considered problem adaptations of three metaheuristics: imperialist competitive algorithm, simulated annealing and differential evolution. In order to improve the quality of the obtained results a local search and learning module are combined with the metaheuristics. The performance of the algorithms is evaluated on 180 test problems by comparing solutions found by metaheuristics with results of a B&B algorithm. The results show that the integration of the learning module and the employed evolutionary algorithms statistically gives better performance regarding the objective function and/or convergence time.

Ahn and Erenguc (1998) and Erenguc et al. (2001) propose heuristic and exact procedures, respectively, for the RCPSP with multiple crashable modes (RCPSPMCM). In this problem each activity can be executed in one of several modes, duration of which may be shortened (crashed) by additional cost. Thus, this problem can be viewed as a combination of the MRCPSP with the time-cost tradeoff problem (TCTP). Indeed, in the absence of the resource constraints the RCPSPMCM reduces to the TCTP, and in the absence of crashing within a mode it becomes the MRCPSP.

The multi-skill project scheduling problem (MSPSP) (see, e.g., Bellenguez and Néron 2005; Bellenguez-Morineau and Néron 2008; Bellenguez-Morineau 2008) is another project scheduling problem where activities may be executed in one of several modes. In this problem multi-skill resources are used during activity processing. Each unit of a multi-skill resource treated as unary resource is able to perform activities requiring different skills. Thus, if a given resource unit is allotted to an activity requiring one skill, it cannot be allotted at the same time to any other activity requiring another skill managed by this resource unit. In other words, each resource unit can be assigned to at most one skill at a time. In consequence, the MSPSP can be treated as multi-mode version of the project scheduling problem with resource/resource trade-off. Li and Womer (2009) extend this model by employing general precedence relations, and replacing the project makespan minimization by a cost-related objective function. Another extension of the MSPSP is presented in Valls et al. (2009) where there are general precedence relations considered, activity processing times depending on the skill level of the resource unit assigned to this activity, and an objective function which takes into account: criticality of activities, assignment of the best skilled resources to each activity, and well-balanced resource workload levels. Santos and Tereso (2011) consider a version of the problem where the objective is to minimize the total cost of the project including cost of resources used, penalty for tardiness, and bonus for earliness.

The MRCPSP with renewable and nonrenewable resources replaced by partially renewable ones is considered by Zhu et al. (2006) who propose a B&C approach to solve this problem optimally. The authors show on the basis of a computational experiment that the proposed approach appears very promising, and can be successfully applied to the MRCPSP as well.

Mika et al. (2008) present an extension of the MRCPSP where schedule-dependent setup times are considered. Three algorithms are proposed for this problem and compared on the basis of a computational experiment. The obtained results show that the proposed tabu search algorithm outperforms the other two presented approaches, namely the multi-start iterative improvement and random sampling. In all the considered approaches, a solution is represented in the same way, and the number of generated solutions is identical.

Li and Womer (2008) model the supply chain configuration problem with resource constraints as the MRCPSP with due dates and additional quality level constraints, and show that this model can be easily extended by applying minimal and maximal time lags, variable resource availability, and various objectives.

A problem of scheduling tests in automotive research and development projects is considered by Bartels and Zimmermann (2009), who model this problem as the MRCPSP with renewable and cumulative resources, as well as minimal and maximal time lags. Some resources used during the course of the project have to be created by certain activities of the project in order to enable the execution of further activities, but it is unknown in advance how many units of these resources are required. Moreover, partially ordered destructive relation between pairs of activities is introduced, because some tests (activities) may destroy a resource unit being used, and therefore it cannot be used in other tests anymore. A mixed integer linear programming (MILP) model of this problem is formulated and used in the computational experiment for small instances solved optimally by CPLEX. For large instances both single-pass and multi-pass priority rule-based heuristics are proposed.

A sports league scheduling problem which occurs in planning nonprofessional table tennis leagues is considered by Knust (2010). This problem consists in finding a schedule for a time-relaxed double round robin tournament with many different hard and soft constraints. One of the two presented approaches is to model this problem as the MRCPSP with partially renewable resources and time-dependent resource profiles. All hard constraints are covered by introducing appropriate additional renewable, nonrenewable, or partially renewable resources, whereas soft constraints are incorporated into the objective function in the form of penalties. All activities have unit processing times. Although a large number of resources have to be introduced to cover all hard constraints, each activity requires only a few of them and, in consequence, the solution algorithm is quite fast. A two-stage local search algorithm is proposed, where in the first stage some theoretical results on so-called balanced home-away assignments (Knust and von Thaden 2006) are implemented, and in the second stage algorithms known from the RCPSP are used (in this implementation – the genetic algorithm by Hartmann 1998).

In the classical models of project scheduling problems it is assumed that activities are nonpreemptable. However, in some project environments it is possible that the nonpreemption assumption might be relaxed. To the best of our knowledge, the multi-mode version of the preemptive RCPSP has only been considered in five papers. Nudtasomboon and Randhawa (1997) were the first to include activity preemption into the MRCPSP. They assume that activities may be preempted an arbitrary number of times at integer time instants. The main contribution of the paper is the formulation of a zero-one integer programming model of the MRCPSP, which includes many important characteristics of project scheduling: activity preemption, renewable and nonrenewable resources, variation in resource availability, time-cost and time-resource tradeoffs, and multiple objectives. Solution algorithms are presented and evaluated for three single-objective problems: with makespan minimization, cost minimization, and resource levelling, as well as for a multi-objective problem combining those three criteria. Prashant et al. (2001) study a version of the problem with renewable resources only. The paper addresses the use of a Petri net as a modelling and scheduling tool in this context. The benefits of Petri nets in project scheduling are discussed. The authors propose some extensions of Petri nets to suit scheduling of activities in a decision CPM. They also propose the use of a P-matrix for token movements in Petri nets. A genetic algorithm is used to find a better solution. Petri-net-aided software including genetic algorithm based search and heuristics is described to deal with a multi-mode, multi-constrained scheduling problem with preemption of activities. In Buddhakulsomsiri and Kim (2006) the authors introduced a new model of the MRCPSP, in which activities may be preempted under situations where resources may be temporarily not available. All resources considered are renewable, and each resource unit may not be available at all times due to resource vacations, which are known in advance, and assignments to other finite duration activities. Activities may only be preempted at discrete points in time, and mode switching is not allowed when activities are resumed after preemption. A designed experiment is conducted that investigates project makespan improvement when activity preemption is permitted in various project scenarios, where different project scenarios are defined by parameters that have been used in the literature. A B&B procedure is applied to solve a number of small project scheduling problems with and without activity preemption. The results show that, in the presence of resource vacations and temporary resource unavailability, activity preemption can significantly improve the optimal project makespan in many scenarios, especially when resources are tight. The results also show that the makespan improvement is primarily dependent on parameters that impact resource utilization. In the follow-on paper (Buddhakulsomsiri and Kim 2007) the authors propose a heuristic approach to the preemptive MRCPSP mentioned above. A new concept, called moving resource strength, is developed to help identify project situations where activity preemption is likely to be beneficial during scheduling. The moving resource strength concept is implemented in priority rule-based heuristics to control activity preemption when scheduling. Multiple comparisons of the performance of combination of activity-mode priority rules used in the heuristics are provided. Computational experiments demonstrate the effectiveness of the heuristic in reducing project makespan, and minimizing the number of preemptions. Zare et al. (2012) introduce a model for the preemptive multi-objective MRCPSP.

Wuliang and Chengen (2009) extend the general DTCTP to a new multi-mode resource-constrained DTCTP model (MRC-DTCTP), in which renewable resource constraints are added to the problem. By predefining the resource price, the renewable resources are related to the project costs, including direct cost and indirect cost. Each activity can be executed in the crashing way, in which the project direct costs are used to shorten the activity duration. According to the characteristics of the MRC-DTCTP, a genetic algorithm for solving it is developed, and its effectiveness is verified by a comparison with an exact algorithm. A similar problem is considered by Nabipoor Afruzi et al. (2013), who develop a multi-objective metaheuristic algorithm named adjusted fuzzy dominance genetic algorithm. The performance of the proposed algorithm is evaluated by a comparison with four other algorithms known from the literature, and the obtained results show the effectiveness of the proposed algorithm. Multi-mode resource-constrained discrete time–cost–resource optimization (MRC-DTCRO) in project scheduling is considered by Ghoddousi et al. (2013), who add the resource leveling capability to the MRC-DTCRO and propose non-domination based genetic algorithm (NSGA-II) to search for the non-dominated solutions considering total project time, cost, and resources moment deviation as three objectives.

21.7 Multi-Mode Problems with Other Objectives

As formulated in Sect. 21.2, in the classical MRCPSP the objective is to minimize the project duration. However, also other objectives have been considered in the project scheduling literature in the context of multiple execution modes of activities. In this section we present a review of the papers devoted to multi-mode problems with objectives other than the project duration, distinguishing between financial objectives (Sect. 21.7.1) and resource-based objectives (Sect. 21.7.2).

21.7.1 Financial Objectives

In this section we deal with multi-mode problems with discounted cash flows, including the basic multi-mode resource-constrained project scheduling problems with discounted cash flows, problems with probabilistic cash flows, and payment scheduling problems.

21.7.1.1 Multi-Mode Resource-Constrained Project Scheduling Problem with Discounted Cash Flows

The multi-mode resource-constrained project scheduling problem with discounted cash flows (MRCPSPDCF) is an extension of the MRCPSP where cash flows (positive and/or negative) are associated with the activities of a project. In the most general case the cash flows may depend on the processing modes of the activities. The objective of the MRCPSPDCF is to maximize the net present value (NPV) of all cash flows of the project. The MRCPSPDCF has been considered in several papers, where heuristic approaches have been developed. To the best of our knowledge, no exact procedure for the MRCPSPDCF has been proposed up to now.

Sung and Lim (1994) study a problem with positive and negative cash flows, and availability constraints imposed on capital and renewable resources. Resource-duration interactions are considered while analyzing the problem. The authors propose a two-phase heuristic algorithm, whose efficiency is tested with various numerical problems.

Ulusoy et al. (2001) study the MRCPSPDCF with renewable, nonrenewable, and doubly-constrained resources. Positive and negative cash flows are associated with events and/or activities, depending on the considered payment model. Four models are analyzed: lump-sum payment at the completion of the project, payments at fixed event nodes, payments at equal time intervals, and progress payments. A genetic algorithm with a specially designed crossover operator able to exploit the multi-component nature of the problem is proposed. A number of 93 problems from the set of instances proposed in Ulusoy and Özdamar (1995) are solved, under the four payment models and different resource combinations. The efficiency of the presented GA algorithm is tested.

Mika et al. (2005) consider the MRCPSPDCF with positive cash flows associated with activities. Four payment models are considered: lump-sum payment at the completion of the project, payments at activity completion times, payments at equal time intervals, and progress payments. Two local search metaheuristics, simulated annealing and tabu search, are proposed to solve the problem. In both the implementations, a solution is classically represented by an activity list and a mode assignment list. An extensive computational experiment is described, performed on a set of instances based on standard test problems constructed by ProGen, where, additionally, activity cash flows are generated randomly from the uniform distribution. The experiment is carried out for the adopted PSPLIB instances with 10, 12, 14, 16, 18, 20, and 30 instances, four payment models, five values of the discount rate, and various combinations of the interval lengths in payment models. In overall, a number of 192,100 MRCPSPDCF instances are solved by each of the metaheuristics tested.

Seifi and Tavakkoli-Moghaddam (2008) consider a problem similar to the one presented by Ulusoy et al. (2001). The main difference is the objective function, which is a sum of the NPV and the activity cost measure proposed by Liu and Wang (2006). The measure takes into account the costs of executing some activities by subcontractors, and is to be minimized. The same four payment models are considered but for the "payments at fixed event nodes" model events occur at the completion times of activities, and therefore the model becomes identical to the "payments at activity completion times" model presented by Mika et al. (2005). Simulated annealing is proposed for the considered problem, and its performance is computationally tested on the PSPLIB instances.

Kavlak et al. (2009) consider a so-called client-contractor bargaining problem in the context of the MRCPSPDCF with renewable resources only. Two payment models are analyzed: payments at activity completion times and progress payments. The project duration is bounded from above by a deadline imposed by the client, which constitutes a hard constraint. The objective is to maximize the bargaining objective function comprised of the objectives of both the client and the contractor. Simulated annealing and genetic algorithm approaches are proposed as solution procedures. A solution is represented by a combination of three serial lists: activity list, mode assignment list, and idle time list. The idle time value represents the exact idle time to be inserted before the start of the corresponding activity in the activity list. The proposed algorithms are experimentally compared on the PSPLIB instances with 14, 20, and 30 activities, adopted by eliminating the tardiness costs, relaxing the deadlines, and excluding the nonrenewable resources. Also sensitivity analysis is conducted for different parameters used in the model, namely the profit margin, the discount rate, and the bargaining power weights.

Chen et al. (2010) consider the MRCPSPDCF with cash inflows and outflows. The ant colony system (ACS) approach is proposed to solve the problem. In the presented algorithm, the AoA network is first converted into a mode-on-node (MoN) graph, which next becomes the construction graph for the ACS algorithm. Based on the construction graph, the authors apply the serial SGS for artificial ants to explore the solutions to the problem. In the process of the algorithm, each ant maintains a schedule generator, and builds its solution by selecting arcs on the graph using pheromone and heuristic information. Eight different domain-based heuristics are developed to enhance the search skill of ants by considering the factors of time, cost, resources, and precedence constraints. The proposed ACS approach is compared with the authors' implementations of the genetic algorithm by Ulusoy et al. (2001), as well as simulated annealing and tabu search by Mika et al. (2005), on 55 randomly generated instances with 13 up to 98 activities. On the basis of experimental results the authors state that their algorithm outperforms the other three metaheuristics.

21.7.1.2 Problems with Probabilistic Cash Flows

Although we do not deal with nondeterministic models in this overview, two papers considering probabilistic cash flows are worth mentioning, as they belong to the small group of papers on the NPV maximization under multiple processing modes of activities.

Özdamar and Dündar (1997) introduce a new model concerning housing projects, where an initial capital covers activity expenditures in the starting phase of the project, and then customers who arrive randomly over the project span, provide the necessary funds for continuation. Capital is considered as a nonrenewable resource, which is the only limited resource in the model. It is reduced by activity expenditures and augmented by the sales of flats. Activities can be carried out in different operating modes, and the total cost of an activity is fixed irrespective of its operating mode. The rate of activity expenditures differs among the modes. The authors propose a flexible heuristic algorithm for solving the capital-constrained mode selection problem, where there exist general precedence relationships among activities, and the magnitude of precedence lags depend on the specific activity mode. The algorithm is tested using a typical housing project with real data, and also by using hypothetical test problems.

Similar situations in the housing industry are further analyzed in Özdamar (1998), where the same stochastic model, involving probabilistic cash inflows, is used. The contractor, who is the owner of the project, starts with an initial capital to cover the activity expenditures, and then capital is augmented by the sale of flats, which take place randomly over the progress of the project. In this risky environment, the contractor has to decide on the rate of expenditure at each decision time in order to maintain a positive cash balance. Hence, activities are performed in multiple processing modes with different durations and the same total cost. A heuristic to construct and reconstruct schedules during the progress of the project is proposed with the aim of maximizing the NPV while completing the project on time. The heuristic incorporates dynamic mode selection objectives which change adaptively according to the current status of the project. Computational experiments demonstrate that the heuristic provides satisfactory results regarding the feasibility of the schedules with respect to the project due date and the nonrenewable resource constraints.

21.7.1.3 Payment Scheduling Problem

In the models presented above it is assumed that the amounts and timing of cash flows are known. In Dayanand and Padman (1997) the authors argue that the expenses associated with activities are usually known but the amounts and timing of payment can be important variables to negotiate by the contractor in order to improve financial returns. Consequently, the payment scheduling problem (PSP) is formulated, in which both the amounts and the timing of the payments have to be determined to maximize the NPV subject to a project deadline. The multi-mode version of the PSP has only been considered in a few papers.

Ulusoy and Cebelli (2000) report a new approach to the payment scheduling problem. The authors set up a special multi-mode problem, where the goals of the contractor and the client are joined in one model. They look for an equitable solution, in which both the contractor and the client deviate from their respective ideal solutions by an equal percentage. The ideal solutions for the contractor and the client result from having a lump-sum payment at the start and at the end of the project, respectively. A double-loop genetic algorithm is proposed to solve the problem, where the outer loop represents the client and the inner loop the contractor. A number of 93 problems from the set of instances proposed in Ulusoy and Özdamar (1995) are solved, and some computational results are reported.

He and Xu (2008) consider a so-called multi-mode project payment scheduling problem (MPPSP) with bonus-penalty structure, where activities can be performed with several modes and a bonus-penalty structure exists at the deadline of the project. In the problem the decisions on when to schedule events and payments, the magnitude of each payment, and the performing mode of each activity need to be optimized. Two-module simulated annealing is proposed to solve the mixed integer nonlinear programming (MINLP) models for the contractor and the client, and a satisfactory solution, which consists of payment event set, event schedule, and payment amount set, may be found through iterations between the heuristic's two modules. The profits of the two parties of the contract are changed significantly by the bonus-penalty structure, and the structure may be considered as a coordination mechanism essentially, which may enhance the flexibility of payment scheduling and be helpful for the two parties to get more profits from the project. The authors solve and analyze a numerical example, and draw the conclusions that the bonus-penalty structure may advance the project completion effectively, and improve the profits of the two parties in the meantime.

He et al. (2009) consider the MPPSP with no resource constraints, where the activities can be performed with one of several discrete modes. The objective is to assign activity modes and progress payments, so as to maximize the NPV of the contractor under the constraint of a project deadline. Using the event-based method, the basic model of the problem is constructed, and in terms of different payment rules it is further extended through changing the constraints on payment time as the progress-based, expense-based, and time-based models. The strong $\mathcal{NP}$-hardness of the problem is proved by simplifying it to the deadline subproblem of the discrete time-cost tradeoff problem. Simulated annealing and tabu search metaheuristics are proposed to solve the problem. The two approaches are computationally compared on a data set constructed by ProGen. The results show that the proposed simulated annealing heuristic is the most promising algorithm for solving the defined problem especially when the instances become larger. In addition, the effects of several key parameters on the net present value of the contractor are analyzed, and some conclusions are given based on the results of the computational experiment.

He et al. (2013) define a new problem named the capital-constrained project payment scheduling problem (CCPPSP) as the combination of the capital-constrained project scheduling problem (CCPSP) and the project payment scheduling problem (PPSP), and they consider the new problem in a multi-mode version (MCCPPSP). The objective is to assign activity modes and payments concurrently, so as to maximize the NPV of the contractor under the constraint of capital availability. Basing on some payment patterns known from the literature, four optimization models are constructed using the event-based method. Two metaheuristics, tabu search and simulated annealing, are developed and compared with multi-start iterative improvement and random sampling, based on a computational experiment performed on a data set generated randomly. The results indicate that the loop nested tabu search is the most promising procedure for the problem studied. Moreover, the effects of several key parameters on the contractor's NPV are investigated, and some conclusions are drawn. The authors also point out directions for future research, which are analyzing the problem from the client's and joint point of view, integrating the bonus/penalty structure into the models, and taking into account various types of resource constraints.

21.7.2 *Resource-Based Objectives*

Beside financial objectives, also resource-based objectives have been considered in a few papers for project scheduling problems with multiple execution modes.

Hsu and Kim (2005) consider the multi-mode resource investment problem (MRIP), which is an extension of the resource investment problem (RIP), where at least one activity may be executed in one of several modes. In this formulation activities are executed using renewable resources only, and a project duration is upper-bounded by a given due date. A priority rule-based heuristic, which simultaneously takes into account the resource usage level and the project due date in one decision rule, is proposed to find suboptimal solutions of the problem.

Sabzehparvar et al. (2008) propose a mathematical model for the multi-mode resource investment problem with general precedence relations (MRIP-GPR). In this model activities are executed using renewable resources only, a project has to be finished before its deadline, and for some pairs of activities there are minimal and maximal time lags which depend on the modes of both activities.

Mika and Węglarz (2004) consider the multi-mode resource leveling problem (MRLP), which is formulated similarly to the MRCPSP assuming that there are two types of limitations: limited availability of renewable and nonrenewable resources, and a deadline for the entire project. The objective is to minimize the weighted sum of squared deviation from the assumed resource usage level for all renewable resources. A simulated annealing approach is developed for the considered problem and tested on the basis of a computational experiment with instances with 10 and 20 activities. The obtained results show that simulated annealing performs better for the projects with tight deadlines.

Motivated by applications in chemical manufacturing, Megow et al. (2011) formulate the multi-mode shutdown and turnaround scheduling problem as an integrated problem that contains various optimization problems as subproblems, such as the time-cost tradeoff problem, the problem of scheduling with resource capacities and working shifts, and the resource leveling problem, all of which have been considered individually previously. The authors report on their successful solution approach within a more comprehensive decision support tool that additionally provides tools for risk analysis during the decision process and for the final schedule. The proposed optimization algorithm yields near-optimal solutions in a short time.

Coughlan et al. (2010) consider the MRLP with the objective to minimize the resource availability cost. They present an exact branch-and-price approach together with a new heuristic to optimally or near-optimally solve the more general turnaround scheduling problem. In addition to precedence and resource constraints, also resource availability periods and multiple execution modes of activities are taken into account. The authors formulate a mixed integer programming problem, which is based on working shifts, and thus has an exponential number of variables. The proposed branch-and-price algorithm to solve this model computes optimal schedules for instances with up to 50 activities. The derived lower bounds demonstrate that heuristic solutions are mostly near the optimum or at least near the best solution found by exact methods within half an hour.

Khalilzadeh et al. (2012) introduce a new problem called the multi-mode resource-constrained project scheduling problem with minimization of total weighted resource tardiness penalty cost (MRCPSP-TWRTPC), in which they consider renewable and nonrenewable resource costs. It is assumed that renewable resources are rented, and are not available in all periods of time of the project. In other words, there is a mandated ready date as well as a due date for each renewable resource type, so that no resource is used before its ready date. However, the resources are permitted to be used after their due dates by paying penalty costs.

The objective is to minimize the total costs of both renewable and nonrenewable resource usage. For this problem, the authors propose a metaheuristic algorithm based on a modified particle swarm optimization (PSO) approach, which uses a modified rule for the displacement of particles. They present a prioritization rule for activities and several improvement and local search methods. Experimental results reveal the effectiveness and efficiency of the proposed algorithm for the problem in question.

21.8 Conclusions

The main goal of this chapter has been to present the state-of-the-art in the area of multi-mode project scheduling problems. The main part of this review is devoted to the basic multi-mode resource-constrained project scheduling problem with the objective to minimize the project duration. For this problem several mixed-integer linear programming formulations as well as exact approaches based on branch-and-bound and branch-and-cut methods are presented. Moreover, the existing methods for lower bounds calculation are described, as well. However, the largest group of the described approaches are heuristic algorithms. The presentation of these algorithms has been divided into three parts concerning priority rule-based algorithms, metaheuristics and other heuristics based on local search procedures, and some other approaches. It is easy to observe that the largest number of papers published in recent years are devoted to the metaheuristic approaches, in particular genetic, evolutionary and swarm intelligence algorithms. Some special cases and extensions of the MRCPSP are also discussed including MRCPSP without nonrenewable resources, MRCPSP with generalized precedence constraints, mode identity RCPSP, RCPSP with multiple crashable modes, multi-skill project scheduling problems, MRCPSP with partially renewable resources, MRCPSP with schedule dependent setup times, MRCPSP with preemptable activities, and multi-mode version of DTCTP. Finally, the multi-mode project scheduling problems with two other categories of objectives, i.e., financial and resource-based ones, are analyzed.

Acknowledgements This work has been funded by the Polish National Science Centre as a research project in the years 2013–2016 under grant No. 2013/08/A/ST6/00296.

References

Afshar-Nadjafi B, Rahimi A, Karimi H (2013) A genetic algorithm for mode identity and the resource constrained project scheduling problem. Sci Iran 20:824–831

Ahn T, Erenguc SS (1998) The resource-constrained project scheduling problem with multiple crashable modes: a heuristic procedure. Eur J Oper Res 107:250–259

Alcaraz J, Maroto C, Ruiz R (2003) Solving the multi-mode resource-constrained project scheduling problem with genetic algorithms. J Oper Res Soc 54:614–626

Artigues C, Roubellat F (2000) A polynomial activity insertion algorithm in a multi-resource schedule with cumulative constraints and multiple modes. Eur J Oper Res 127:297–316

Ballestín F, Barrios A, Valls V (2011) An evolutionary algorithm for the resource-constrained project scheduling problem with minimum and maximum time lags. J Sched 14:391–406

Ballestín F, Barrios A, Valls V (2013) Looking for the best modes helps solving the MRCPSP/max. Int J Prod Res 51:813–827

Barrios A, Ballestín F, Valls V (2011) A double genetic algorithm for the MRCPSP/max. Comput Oper Res 38:33–43

Bartels JH, Zimmermann J (2009) Scheduling tests in automotive R&D projects. Eur J Oper Res 193:805–819

Bedworth DD (1973) Industrial systems: planning, analysis, and control. The Ronald Press, New York

Bellenguez O, Néron E (2005) Lower bounds for the multi-skill project scheduling problem with hierarchical level of skills. In: Burke EK, Trick M (eds) Practice and theory of automated timetabling V. Lecture Notes in Computer Science, vol 3616. Springer, Berlin, pp 229–243

Bellenguez-Morineau O (2008) Methods to solve multi-skill project scheduling problem. 4OR-Q J Oper Res 6:85–88

Bellenguez-Morineau O, Néron E (2008) Multi-mode and multi-skill project scheduling problem. In: Artigues C, Demassey S, Néron E (eds) Resource-constrained project scheduling: models, algorithms, extensions and applications. ISTE-Wiley, London, pp 149–160

Boctor FF (1993) Heuristics for scheduling projects with resource restrictions and several resource-duration modes. Int J Prod Res 31:2547–2558

Boctor FF (1996a) A new and efficient heuristic for scheduling projects with resource restrictions and multiple execution modes. Eur J Oper Res 90:349–361

Boctor FF (1996b) Resource-constrained project scheduling by simulated annealing. Int J Prod Res 34:2335–2351

Boschetti M, Maniezzo V (2009) Benders decomposition, Lagrangean relaxation and metaheuristic design. J Heuristics 15:283–312

Bouleimen K, Lecocq H (2003) A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. Eur J Oper Res 149:268–281

Brucker P, Knust S (2003) Lower bounds for resource-constrained project scheduling problems. Eur J Oper Res 149:302–313

Buddhakulsomsiri J, Kim DS (2006) Properties of multi-mode resource-constrained project scheduling problems with resource vacations and activity splitting. Eur J Oper Res 175:279–295

Buddhakulsomsiri J, Kim DS (2007) Priority rule-based heuristic for multi-mode resource-constrained project scheduling problems with resource vacations and activity splitting. Eur J Oper Res 178:374–390

Calhoun KM, Deckro RF, Moore JT, Chrissis JW, Van Hove JC (2002) Planning and re-planning in project and production scheduling. Omega Int J Manag S 30:155–170

Chen WN, Zhang J, Chung HSH, Huang RZ, Liu O (2010) Optimizing discounted cash flows in project scheduling: an ant colony optimization approach. IEEE T Syst Man Cybern C 40:64–77

Chiang CW, Huang YQ, Wang WY (2008) Ant colony optimization with parameter adaptation for multi-mode resource-constrained project scheduling. J Intell Fuzzy Syst 29:345–358

Christofides N, Alvarez-Valdes R, Tamarit JM (1987) Project scheduling with resource constraints: a branch-and-bound approach. Eur J Oper Res 29:262–273

Chyu CC, Chen AHL, Lin XH (2005) A hybrid ant colony approach to multi-mode resource-constrained project scheduling problems with nonrenewable types. In: Proceedings of first international conference on operations and supply chain management, Bali, 2005

Coelho J, Vanhoucke M (2011) Multi-mode resource-constrained project scheduling using RCPSP and SAT solvers. Eur J Oper Res 213:73–82

Coughlan ET, Lübbecke ME, Schulz J (2010) A branch-and-price algorithm for multi-mode resource leveling. Lecture notes in computer science, vol 6049. Springer, Berlin, pp 226–238

Damak J, Jarboui B, Siarry P, Loukil T (2009) Differential evolution for solving multi-mode resource-constrained project scheduling problems. Comput Oper Res 36:2653–2659

Dayanand N, Padman R (1997) On modeling progress payments in project networks. J Oper Res Soc 48:906–918

De Reyck B, Herroelen WS (1998) A branch-and-bound procedure for the resource-constrained project scheduling problem with generalized precedence relations. Eur J Oper Res 111:152–174

De Reyck B, Herroelen WS (1999) The multi-mode resource-constrained project scheduling problem with generalized precedence relations. Eur J Oper Res 119:538–556

Debels D, Vanhoucke M (2007) A decomposition-based genetic algorithm for the resource-constrained project scheduling problem. Oper Res 55:457–469

Demeulemeester EL, Herroelen WS (1992) A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. Manag Sci 38:1803–1818

Demeulemeester EL, Dodin B, Herroelen WS (1993) A random network activity generator. Oper Res 41:972–980

Dornorf U (2002) Project scheduling with time windows: from theory to applications. Physica, Heidelberg

Drexl A, Grünewald J (1993) Nonpreemptive multi-mode resource-constrained project scheduling. IIE Trans 25:74–81

Drexl A, Nissen R, Patterson JH and Salewski F (2000) ProGen/πx: an instance generator for resource constrained project scheduling problems with partially renewable resources and further extensions. Eur J Oper Res 125:59–72

Elloumi S, Fortemps P (2010) A hybrid rank-based evolutionary algorithm applied to multi-mode resource-constrained project scheduling problem. Eur J Oper Res 205:31–41

Elloumi S, Fortemps P, Teghem J, Loukil T (2006) A new bi-objective algorithm using clustering heuristics to solve the multi-mode resource-constrained project scheduling problem. In: Proceedings of the 25th workshop of the UK planning and scheduling special interest group (PlanSIG 2006), Nottingham, pp 113–120

Elmaghraby SE (1977) Activity networks: project planning and control by network models. Wiley, New York

Erenguc SS, Ahn T, Conway DG (2001) The resource constrained project scheduling problem with multiple crashable modes: an exact solution method. Nav Res Logist 48:107–127

Fischetti M, Lodi A (2003) Local branching. Math Program 98:23–47

Gagnon M, Boctor FF, d'Avignon G (2005) A tabu search algorithm for the multiple mode resource constrained project scheduling problem. In: Proceedings of the ASAC 2005, Toronto, Canada, pp 89–102

Ghoddousi P, Eshtehardian E, Jooybanpour S, Javanmardi A (2013) Multi-mode resource-constrained discrete time–cost–resource optimization in project scheduling using non-dominated sorting genetic algorithm. Automa Constr 30:216–227

Hartmann S (1998) A competitive genetic algorithm for resource-constrained project scheduling. Nav Res Logist 45:733–750

Hartmann S (2001) Project scheduling with multiple modes: a genetic algorithm. Ann Oper Res 102:111–135

Hartmann S, Briskorn D (2010) A survey of variants and extensions of the resource-constrained project scheduling problem. Eur J Oper Res 207:1–14

Hartmann S, Drexl A (1998) Project scheduling with multiple modes: a comparison of exact algorithms. Networks 32:283–297

Hartmann S, Sprecher A (1996) A note on "Hierarchical models for multi-project planning and scheduling". Eur J Oper Res 94:377–383

He Z, Xu Y (2008) Multi-mode project payment scheduling problems with bonus-penalty structure. Eur J Oper Res 189:1191–1207

He Z, Wang N, Jia T, Xu Y (2009) Simulated annealing and tabu search for multimode project payment scheduling. Eur J Oper Res 198:688–696

He Z, Liu R, Jia T (2013) Metaheuristics for multi-mode capital-constrained project payment scheduling. Eur J Oper Res 223:605–613

Heilmann R (2001) Resource-constrained project scheduling: a heuristic for the multi-mode case. OR Spektrum 23:335–357

Heilmann R (2003) A branch-and-bound procedure for the multi-mode resource-constrained project scheduling problem with minimum and maximum time lags. Eur J Oper Res 144:348–365

Hsu CC, Kim DS (2005) A new heuristic for the multi-mode resource investment problem. J Oper Res Soc 56:406–413

ILOG (2002) ILOG CPLEX 7.5: Reference manual. ILOG, Mountain View, CA

Jarboui B, Damak N, Siarry P, Rebai A (2008) A combinatorial particle swarm optimization for solving multi-mode resource-constrained project scheduling problems. Appl Math Comput 195:299–308

Jędrzejowicz P, Ratajczak E (2006) Population learning algorithm for the resource-constrained project scheduling. In: Józefowska J, Węglarz J (eds) Perspectives in modern project scheduling. Springer, Berlin, pp 275–296

Józefowska J, Mika M, Różycki R, Waligóra G, Węglarz J (2001) Simulated annealing for multi-mode resource-constrained project scheduling. Ann Oper Res 102:137–155

Kavlak N, Ulusoy G, Sivrikaya-Şerifoğlu F, Birbil I (2009) Client-contractor bargaining on net present value in project scheduling with limited resources. Nav Res Logist 56:93–112

Kelley Jr. JE (1963) The critical path method: resources planning and scheduling. In: Muth JF, Thompson GL (eds) Industrial scheduling. Prentice-Hall, Englewood Cliffs, pp 347–365

Khalilzadeh M, Kianfar F, Chaleshtari AS, Shadrokh S, Ranjbar M (2012) A modified PSO algorithm for minimizing the total costs of resources in MRCPSP. Math Probl Eng. doi:10.1155/2012/365697

Knotts G, Dror M, Hartman BC (2000) Agent-based project scheduling. IIE Trans 32:387–401

Knust S (2010) Scheduling non-professional table-tennis leagues. Eur J Oper Res 200:358–367

Knust S, von Thaden M (2006) Balanced home-away assignments. Discrete Optim 3:354–365

Kolisch R (1995) Project scheduling under resource constraints: efficient heuristics for several problem classes. Physica, Heidelberg

Kolisch R (1996) Serial and parallel resource-constrained project scheduling methods revisited: theory and computation. Eur J Oper Res 90:320–333

Kolisch R, Drexl A (1997) Local search for nonpreemptive multi-mode resource-constrained project scheduling. IIE Trans 29:987–999

Kolisch R, Hartmann S (1999) Heuristic algorithms for the resource-constrained project scheduling problem: classification and computational analysis. In: Węglarz J (ed) Project scheduling: recent models, algorithms, and applications. Kluwer Academic, Boston, pp 147–178

Kolisch R, Sprecher A (1997) PSPLIB: a project scheduling problem library. Eur J Oper Res 96:205–216

Kolisch R, Sprecher A, Drexl A (1995) Characterization and generation of a general class of resource-constrained project scheduling problems. Manag Sci 41:1693–1703

Kyriakidis T, Kopanos G, Georgiadis M (2012) MILP formulation for single- and multi-mode resource-constrained project scheduling problems. Comput Chem Eng 36:369–385

Li H, Zhang H (2013) Ant colony optimization-based multi-mode scheduling under renewable and nonrenewable resource constraints. Autom Constr 35:431–438

Li KY, Willis RJ (1992) An iterative scheduling technique for resource-constrained project scheduling. Eur J Oper Res 56:370–379

Li H, Womer K (2008) Modeling the supply chain configuration problem with resource constraints. Int J Proj Manag 26:646–654

Li H, Womer K (2009) Scheduling projects with multi-skilled personnel by a hybrid MILP/CP Benders decomposition algorithm. J Sched 12:281–298

Liu Z, Wang H (2006) Heuristic algorithm for the RCPSP with the objective of minimizing activities' cost. J Syst Eng Electron 17:96–102

Lova A, Tormos P, Barber F (2006) Multi-mode resource constrained project scheduling: scheduling schemes, priority rules and mode selection rules. Inteligencia Artif 10:69–86

Lova A, Tormos P, Cervantes M, Barber F (2009) An efficient hybrid genetic algorithm for scheduling projects with resource constraints and multiple execution modes. Int J Prod Econ 117:302–316

Maniezzo V, Mingozzi A (1999) A heuristic procedure for the multi-mode project scheduling problem based on Bender's decomposition. In: Węglarz J (ed) Project scheduling: recent models, algorithms and applications. Kluwer, Dordrecht, pp 179–196

Maroto C, Tormos P (1994) Project management: an evaluation of software quality. Int Trans Oper Res 1:209–221

Megow N, Möhring R, Schulz J (2011) Decision support and optimization in shutdown and turnaround scheduling. INFORMS J Comput 23:189–204

Mika M, Węglarz J (2004) Heurystyczne algorytmy równoważenia obciążenia w problemach rozdziału zasobów z wieloma sposobami wykonywania czynności. In: Gessing R, Szkodny T (eds) Automatyzacja Procesów Dyskretnych: Optymalizacja Dyskretna; Robotyka i Sterowniki Programowalne. Wydawnictwa Naukowo Techniczne, Warszawa, pp 161–168

Mika M, Waligóra G, Węglarz J (2005) Simulated annealing and tabu search for multi-mode resource-constrained project scheduling with positive discounted cash flows and different payment models. Eur J Oper Res 164:639–668

Mika M, Waligóra G, Węglarz J (2008) Tabu search for multi-mode resource-constrained project scheduling with schedule-dependent setup times. Eur J Oper Res 187:1238–1250

Mingozzi A, Maniezzo V, Ricciardelli S, Bianco L (1998) An exact algorithm for project scheduling with resource constraints based on a new mathematical formulation. Manag Sci 44:714–729

Mori M, Tseng CC (1997) A genetic algorithm for multi-mode resource constrained project scheduling problem. Eur J Oper Res 100:134–141

Nabipoor Afruzi E, Roghanian E, Najafi AA, Mazinani (2013) A multi-mode resource-constrained discrete time-cost tradeoff problem solving using an adjusted fuzzy dominance genetic algorithm. Sci Iran 20:931–944

Neumann K, Schwindt C, Zimmermann J (2002) Project scheduling with time windows and scarce resources: temporal and resource-constrained project scheduling with regular and nonregular objective functions. Springer, Berlin

Nonobe K, Ibaraki T (2002) Formulation and tabu search algorithm for the resource constrained project scheduling problem. In: Ribeiro CC, Hansen P (eds) Essays and surveys in metaheuristics. Kluwer, Boston, pp 557–588

Nudtasomboon N, Randhawa SU (1997) Resource-constrained project scheduling with renewable and nonrenewable resources and time-resource trade-offs. Comput Ind Eng 32:227–242

Özdamar L (1998) On scheduling project activities with variable expenditure rates. IIE Trans 30:695–704

Özdamar L (1999) A genetic algorithm approach to a general category project scheduling problem. IEEE T Syst Man Cybern C 29:44–59

Özdamar L, Dündar H (1997) A flexible heuristic for a multi-mode capital constrained project scheduling problem with probabilistic cash inflows. Comput Oper Res 24:1187–1200

Özdamar L, Ulusoy G (1994) A local constraint based analysis approach to project scheduling under general resource constraints. Eur J Oper Res 79:287–298

Patterson JH, Słowiński R, Talbot FB, Węglarz J (1989) An algorithm for a general class of precedence and resource constrained scheduling problems. In: Słowiński R, Węglarz J (eds) Advances in project scheduling. Elsevier, Amsterdam, pp 3–28

Patterson JH, Talbot FB, Słowiński R, Węglarz J (1990) Computational experience with a backtracking algorithm for solving a general class of precedence and resource-constrained scheduling problems. Eur J Oper Res 49:68–79

Pesch E (1999) Lower bounds in different problem classes of project schedules with resource constraints. In: Węglarz J (ed) Project scheduling: recent models, algorithms and applications. Kluwer, Dordrecht, pp 53–76

Prashant Reddy J, Kumanan S, Krishnaiah Chetty OV (2001) Application of Petri nets and a genetic algorithm to multi-mode multi-resource constrained project scheduling. Int J Adv Manuf Tech 17:305–314

Pritsker ABA, Watters LJ, Wolfe PM (1969) Multiproject scheduling with limited resources: a zero-one programming approach. Manag Sci 16:93–107

Rahimi A, Karimi H, Afshar-Nadjafi B (2013) Using meta-heuristics for project scheduling under mode identity constraints. Appl Softw Comput 13:2124–2135

Ranjbar M, De Reyck B, Kianfar F (2009) A hybrid scatter search for the discrete time/resource trade-off problem in project scheduling. Eur J Oper Res 193:35–48

Sabzehparvar M, Seyed-Hosseini SM (2008) A mathematical model for the multi-mode resource-constrained project scheduling problem with mode dependent time lags. J Supercomput 44:257–273

Sabzehparvar M, Seyed-Hosseini SM, Nouri S (2008) A mathematical model for the multi-mode resource investment problem. J Ind Eng 4:25–32

Salewski F, Schrimer A, Drexl A (1997) Project scheduling under resource and mode identity constraints: model, complexity, methods, and application. Eur J Oper Res 102:88–110

Santos MA, Tereso AP (2011) On the multi-mode, multi-skill resource constrained project scheduling problem: a software application. Adv Softw Comp 96:239–248

Seifi M, Tavakkoli-Moghaddam R (2008) A new bi-objective model for a multimode resource-constrained project scheduling problem with discounted cash flows and four payments model. Int J Eng Trans A Basics 21:347–360

Speranza MG, Vercellis C (1993) Hierarchical models for multi-project planning and scheduling. Eur J Oper Res 64:312–325

Sprecher A (1994) Resource-constrained project scheduling: exact methods for the multi-mode case. Springer, Berlin

Sprecher A, Drexl A (1998) Multi-mode resource-constrained project scheduling by a simple, general and powerful sequencing algorithm. Eur J Oper Res 107:431–450

Sprecher A, Hartmann S, Drexl A (1997) An exact algorithm for project scheduling with multiple modes. OR Spektrum 19:195–203

Sung CS, Lim SK (1994) A project activity scheduling problem with net present value measure. Int J Prod Econ 37:177–187

Talbot FB (1982) Resource-constrained project scheduling with time-resource trade-offs: the nonpreemptive case. Manag Sci 28:1197–1210

Tseng LY, Chen SC (2009) Two-phase genetic local search algorithm for the multimode resource-constrained project scheduling problem. IEEE T Evol Comput 13:848–857

Ulusoy G, Cebelli S (2000) An equitable approach to the payment scheduling problem in project management. Eur J Oper Res 127:262–278

Ulusoy G, Özdamar L (1995) A heuristic scheduling algorithm for improving the duration and net present value of a project. Int J Oper Prod Man 15:89–98

Ulusoy G, Sivrikaya-Şerifoğlu, Şahin S (2001) Four payment models for the multi-mode resource constrained project scheduling problem with discounted cash flows. Ann Oper Res 102:237–261

Valls V, Quintanilla S, Ballestín F (2003) Resource-constrained project scheduling: a critical activity reordering heuristic. Eur J Oper Res 149:282–301

Valls V, Ballestín F, Quintanilla S (2004) A population-based approach to the resource-constrained project scheduling problem. Ann Oper Res 131:305–324

Valls V, Ballestín F, Quintanilla S (2005) Justification and RCPSP: a technique that pays. Eur J Oper Res 165:375–386

Valls V, Pérez Á, Quintanilla S (2009) Skilled workforce scheduling in service centres. Eur Oper Res 193:791–804

Van Hove JC, Deckro RF (1998) Multi-modal project scheduling with generalized precedence constraints. In: Babarasoğlu G, Karabati S, Özdamar L, Ulusoy G (eds) Proceedings of the sixth international workshop on project management and scheduling, Istanbul, pp 137–140

Van Peteghem V, Vanhoucke M (2009) An artificial immune system for the multi-mode resource-constrained project scheduling problem. In: Cotta C, Cowling B (eds) Proceedings of the 9-th European conference EvoCOP 2009. Lecture notes in computer science, vol 5482. Springer, Berlin, pp 85–96

Van Peteghem V, Vanhoucke M (2010) A genetic algorithm for the preemptive and nonpreemptive multi-mode resource-constrained project scheduling problem. Eur J Oper Res 201:409–418

Van Peteghem V, Vanhoucke M (2011) Using resource scarceness characteristics to solve the multi-mode resource-constrained project scheduling problem. J Heuristics 17:705–728

Vanhoucke M, Coelho J, Debels D, Maenhout B, Tavares L (2008) An evaluation of the adequacy of project network generators with systematically sampled networks. Eur J Oper Res 187:511–524

Wang L, Fang Ch (2011) An effective shuffled frog-leaping algorithm for multi-mode resource-constrained project scheduling problem. Inform Sci 181:4804–4822

Wang L, Fang Ch (2012) An effective estimation of distribution algorithm for the multi-mode resource-constrained project scheduling problem. Comput Oper Res 39:449–460

Węglarz J, Józefowska J, Mika M, Waligóra G (2011) Project scheduling with finite or infinite number of activity processing modes: a survey. Eur J Oper Res 208:177–205

Wuliang P, Chengen W (2009) A multi-mode resource-constrained discrete time-cost trade-off problem and its genetic algorithm based solution. Int J Proj Manag 27:600–609

Zapata JC, Hodge BM, Reklaitis GV (2008) The multi-mode resource constrained multi-project scheduling problem: alternative formulations. AIChE J 54:2101–2119

Zare Z, Naddaf A, Reza Salehi M (2012) Proposing a model on preemptive multi-mode resource-constrained project scheduling problem. Int J Bus Soc Sci 3:126–127

Zhang H (2012) Ant colony optimization for multimode resource-constrained project scheduling. J Manag Eng 28:150–159

Zhang H, Tam CM, Li H (2006) Multi-mode project scheduling based on particle swarm optimization. Comput-Aided. Civ Inf 21:93–103

Zhu G, Bard JF, Yu G (2006) A branch-and-cut procedure for the multi-mode resource-constrained project scheduling problem. INFORMS J Comput 18:377–390

Chapter 22
The Multi-Mode Resource-Constrained Project Scheduling Problem

José Coelho and Mario Vanhoucke

Abstract This chapter reports on a new solution approach for the multi-mode resource-constrained project scheduling problem (MRCPSP, $MPS|prec|C_{max}$). This problem type aims at the selection of a single activity mode from a set of available modes in order to construct a precedence and a (renewable and nonrenewable) resource-feasible project schedule with a minimal makespan. The problem type is known to be $\mathcal{NP}$-hard and has been solved using various exact as well as (meta-)heuristic procedures. The new algorithm splits the problem type into a mode assignment and a single mode project scheduling step. The mode assignment step is solved by a satisfiability (SAT) problem solver and returns a feasible mode selection to the project scheduling step. The project scheduling step is solved using an efficient meta-heuristic procedure from literature to solve the resource-constrained project scheduling problem (RCPSP). However, unlike many traditional meta-heuristic methods in literature to solve the MRCPSP, the new approach executes these two steps in one run, relying on a single priority list. Straightforward adaptations to the pure SAT solver by using pseudo boolean nonrenewable resource constraints has led to a high quality solution approach in a reasonable computational time. Computational results show that the procedure can report similar or sometimes even better solutions than found by other procedures in literature, although it often requires a higher CPU time.

Keywords Makespan minimization • Multi-mode • Project scheduling • Resource constraints • SAT

J. Coelho (✉)
Department of Sciences and Technology, Universidade Aberta, Lisbon, Portugal
e-mail: Jose.Coelho@uab.pt

M. Vanhoucke
Faculty of Economics and Business Administration, Ghent University, Gent, Belgium

Technology and Operations Management Area, Vlerick Business School, Gent, Belgium

Department of Management Science and Innovation, University College London, London, UK
e-mail: mario.vanhoucke@ugent.be; mario.vanhoucke@vlerick.com; m.vanhoucke@ucl.ac.uk

C. Schwindt, J. Zimmermann (eds.), *Handbook on Project Management and Scheduling Vol. 1*, International Handbooks on Information Systems,
DOI 10.1007/978-3-319-05443-8_22

22.1 Introduction

This chapter presents a novel meta-heuristic approach to solve the non-preemptive multi-mode resource-constrained project scheduling problem (MRCPSP) within the presence of both limited renewable and nonrenewable resource constraints, as proposed in Coelho and Vanhoucke (2011). The MRCPSP is an extension of the well-known RCPSP to the presence of multiple activity modes where each activity can be executed under a different duration and a corresponding renewable and nonrenewable resource use. For a recent survey on MRCPSP we refer to Węglarz et al. (2011), and to Chap. 21 of this handbook. The chapter is organized as follows. Section 22.2 introduces the notation and describes the problem formulation in detail. In Sect. 22.3 we present our approach to solve the scheduling problem type under study and give illustrative examples. Moreover, it is shown that the solution approach is very general and can be used for various other scheduling extensions. Section 22.4 enhances this solution approach to cope with excessive memory requirements. Section 22.5 reports comparative computational results and Sect. 22.6 contains the conclusions.

22.2 Model Formulation

The multi-mode project scheduling problem with multiple renewable and nonrenewable resources ($MPS|prec|C_{max}$ in the three field classification) can be stated as follows. A set of activities V, numbered from a dummy start node 0 to a dummy end node $n+1$, is to be scheduled without pre-emption on a set $\mathscr{R}$ of renewable resources and a set of $\mathscr{R}^n$ of nonrenewable resources. Each renewable resource $k \in \mathscr{R}$ has a constant availability R_k per period while each nonrenewable resource $k \in \mathscr{R}^n$ is restricted to R_k units over the complete planning horizon. Each non-dummy activity $i \in V$ can be executed in one of M_i modes (p_{im}, r_{im}) with $r_{im} = (r_{ikm})_{k \in \mathscr{R} \cup \mathscr{R}^n}$ and $m \in \{1, 2, \ldots, M_i\}$. The selection of an activity mode involves a deterministic duration p_{im} for each activity i, which requires r_{ikm} units of resources $k \in \mathscr{R} \cup \mathscr{R}^n$. The start and end dummy activities representing the start and completion of the project have only one mode with a duration and renewable and nonrenewable resource requirements equal to zero. A project network is represented by a topologically ordered activity-on-node format where E is the set of pairs of activities between which a finish-start precedence relationship with time lag 0 exists. We assume graph $G = (V, E)$ to be acyclic. A schedule S is defined by a vector of activity start times and is said to be feasible if all precedence and renewable and nonrenewable resource constraints are satisfied. The objective of the problem type is to find a feasible schedule within the lowest possible project makespan, and hence, the problem type can be represented as $m, 1T|cpm, disc, mu|C_{max}$ using the classification scheme of Herroelen et al. (1999) or as $MPS|prec|C_{max}$ following the classification scheme of Brucker et al. (1999). The multi-mode resource-constrained project scheduling problem can be formulated as follows (see Talbot 1982):

$$\text{Min.} \sum_{t=ES_{n+1}}^{LS_{n+1}} t x_{n+1,1,t} \tag{22.1}$$

$$\text{s.t.} \sum_{m=1}^{M_i} \sum_{t=ES_i}^{LS_i} (t + p_{im}) x_{imt} \leq \sum_{m=1}^{M_j} \sum_{t=ES_j}^{LS_j} t x_{jmt} \quad ((i,j) \in E) \tag{22.2}$$

$$\sum_{m=1}^{M_i} \sum_{t=ES_i}^{LS_i} x_{imt} = 1 \quad (i \in V) \tag{22.3}$$

$$\sum_{i=1}^{n} \sum_{m=1}^{M_i} r_{ikm} \sum_{s=\max(t-p_{im},ES_i)}^{\min(t-1,LS_i)} x_{ims} \leq R_k \quad (k \in \mathscr{R};\ t = 1,\ldots,UB) \tag{22.4}$$

$$\sum_{i=1}^{n} \sum_{m=1}^{M_i} r_{ikm} \sum_{t=ES_i}^{LS_i} x_{imt} \leq R_k \quad (k \in \mathscr{R}^n) \tag{22.5}$$

$$x_{imt} \in \{0,1\} \quad (i \in V;\ m = 1,\ldots,M_i;\ t = 1,\ldots,UB) \tag{22.6}$$

where x_{imt} is equal to 1 if activity i is performed in mode m and started at time instance t, and 0 otherwise. Equation (22.1) minimizes the total project makespan. The constraints of Eq. (22.2) take the finish-start precedence relations with a time lag of zero into account. Constraints of Eq. (22.3) secure that each non-preemptable activity is performed exactly once in exactly one mode. The renewable resource constraints are satisfied thanks to constraints of Eq. (22.4) where UB is an upper bound on the project makespan. The constraints of Eq. (22.5) restricts the use of the nonrenewable resources over the complete time horizon. The constraints of Eq. (22.6) force the decision variables to be binary values. Note that the abbreviations ES_i and LS_i are used to denote the earliest and latest start for activity i given the project upper bound UB using traditional forward and backward critical path calculations.

Consider an example project taken from Kolisch and Drexl (1997) that will be used throughout the remainder of this chapter with five non-dummy activities, one renewable resource $k = 1$ with an availability of $R_1 = 4$ and one nonrenewable resource $k = 2$ with an availability $R_2 = 8$. Figure 22.1 shows the activity-on-node network with the different activity modes below each node. The right part of the figure displays the optimal renewable resource profile, resulting in a total project makespan of seven time units.

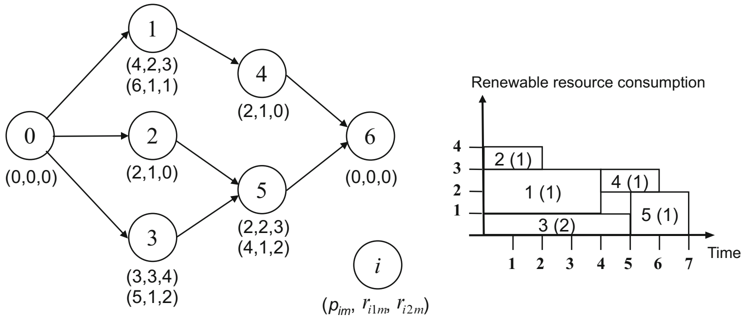


Fig. 22.1 A fictitious example project with optimal resource profiles (*Source*: Kolisch and Drexl 1997)

22.3 Solution Approach

The MRCPSP can be easily modeled as an RCPSP instance where each multi-mode activity i is split into M_i single-mode sub-activities among which exactly one sub-activity needs to be selected for execution. Consequently, the project network of Fig. 22.1 can be transformed into an RCPSP network with $\sum_{i=1}^{n} M_i$ non-dummy sub-activities as displayed in Fig 22.2, where the first number below the node denotes the sub-activity duration and the two other numbers below the node the renewable and nonrenewable resource requirements. Consequently, the MRCPSP can be split into a mode assignment step taking the nonrenewable resource constraints into account [i.e., constraints Eqs. (22.3) and (22.5)] and a single-mode resource-constrained project scheduling step taking the renewable resource and precedence constraints [constraints Eqs. (22.2) and (22.4)] into account. The mode assignment and the project scheduling steps will be discussed in Sects. 22.3.1 and 22.3.2, respectively. In literature, most meta-heuristic search procedures for the MRCPSP make use of an activity list and a mode list, and run the mode assignment step and the project scheduling step iteratively. In the solution approach of the current chapter, these steps will be performed in a single run, making use of only one priority list per run (which acts both as an activity list and a mode list). This new feature will be discussed in Sect. 22.3.2.

22.3.1 Mode Assignment

The mode assignment step boils down to the assignment of a single mode from the set of modes to each activity while not violating the limited nonrenewable resource availability constraints, and can be easily modelled as a Boolean satisfiability problem instance. The boolean satisfiability problem (SAT) is a well-known decision problem where an expression of Boolean variables (referred to as literals) linked by

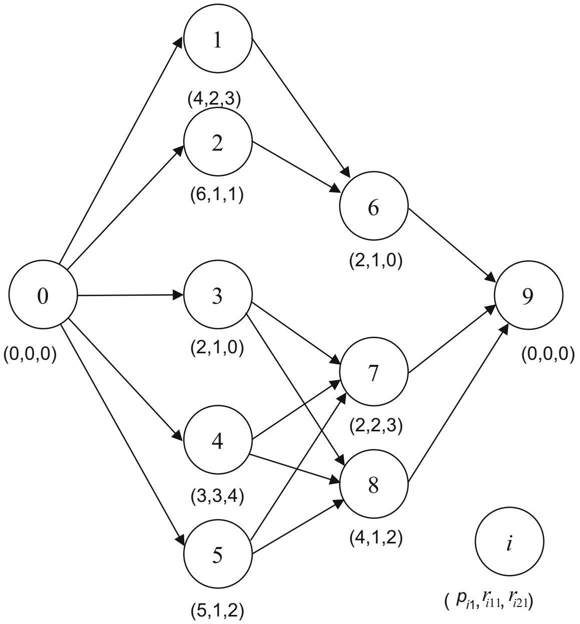


Fig. 22.2 The single-mode RCPSP network of Fig. 22.1 without any activity mode restrictions

means of the logical operators *and*, *or*, and *not* is questioned to be true or false. The problem has been studied extensively in literature (see, e.g., the paper by Marques-Silva and Sakallah 1999, amongst others) and is known to be $\mathcal{NP}$-complete (see Cook 1971).

Obviously, the solution of the mode assignment step needs to be checked and possibly adjusted for the nonrenewable resource infeasibility, and hence, this step can be easily modeled as a SAT instance. The enumeration of all feasible mode combinations is practically impossible due to the huge number of possible combinations. Moreover, Kolisch and Drexl (1997) have shown that finding a feasible mode combination for the MRCPSP is $\mathcal{NP}$-complete when two or more nonrenewable resources are taken into account. Therefore, the choice of selecting a SAT algorithm above an alternative enumeration algorithm is based on the following logic:

- A SAT algorithm allows a simple mode feasibility check and a scheduling step using a single activity list instead of two separate lists as normally done in literature.
- In a SAT algorithm it is easy to implement learning (Sect. 22.3.3.2) which can be used over different mode combination searches.

Consequently, the mode assignment step can be represented in the *conjunctive normal form (CNF)* which is a conjunction of clauses linked by the "and" operator. A SAT instance contains several clauses to deal with the various mode assignments and/or the nonrenewable resource constraints. The clauses for the network of Figs. 22.1 and 22.2 can be represented as follows:

Single-mode activities:	$x_0 + x_3 + x_6 + x_9 = 4$
Mode assignment for activity 1:	$x_1 + x_2 = 1$
Mode assignment for activity 3:	$x_4 + x_5 = 1$
Mode assignment for activity 5:	$x_7 + x_8 = 1$
Nonrenewable resource constraint:	$3x_1 + x_2 + 4x_4 + 2x_5 + 3x_7 + 2x_8 \leq 8$

with x_i a 0/1 variable of sub-activity i to denote whether the mode has been assigned (1, true) or not (0, false). All mode assignment constraints can be easily translated into the CNF where the 0/1 x_i variables are now boolean variables (literals). The nonrenewable resource constraint is known as a pseudo boolean constraint. These type of constraints can be solved by a pseudo boolean solver (Chai and Kuehlmann 2005; Markov et al. 2002) or can be translated into the CNF (Bailleux et al. 2006). However, the nonrenewable resource constraint also contains activity information (e.g., x_1 and x_2 are variables from the same activity and hence, only one variable can be set to true), but none of the previous methods takes this additional information into account. The enumeration scheme of Sect. 22.3.1.1 translates the pseudo boolean nonrenewable resource constraints into CNF using this extra activity information. The mode assignment constraints can be translated into CNF as follows[1]:

Single-mode activities:	$x_0 \wedge x_3 \wedge x_6 \wedge x_9$
Mode assignment for activity 1:	$(x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2})$
Mode assignment for activity 3:	$(x_4 \vee x_5) \wedge (\overline{x_4} \vee \overline{x_5})$
Mode assignment for activity 5:	$(x_7 \vee x_8) \wedge (\overline{x_7} \vee \overline{x_8})$
Nonrenewable resource constraint:	Enumeration scheme discussed in 22.3.1.1

In the remaining sections, the translation of pseudo boolean nonrenewable resource constraint clauses to a CNF is explained in detail. In Sect. 22.4, an adapted pseudo boolean solver is presented that incorporates these nonrenewable resource constraint clauses and both approaches are compared.

22.3.1.1 Nonrenewable Resource Constraint Clauses

The construction of the constraint clauses for the nonrenewable resource constraints is based on a simple yet efficient enumeration scheme. The enumeration scheme starts for the initial nonrenewable resource constraint (e.g., $3x_1 + x_2 + 4x_4 + 2x_5 + 3x_7 + 2x_8 \leq 8$ for the example project) and gradually reduces the size of this constraint by iteratively setting boolean variables to true. More precisely, the

[1] In general, a constraint $y_1 + y_2 + \ldots + y_n = 1$ can be represented in the CNF as $(y_1 \vee y_2 \vee \ldots \vee y_n) \wedge (\overline{y_1} \vee \overline{y_2}) \wedge \ldots \wedge (\overline{y_1} \vee \overline{y_n}) \wedge (\overline{y_2} \vee \overline{y_3}) \wedge \ldots \wedge (\overline{y_{n-1}} \vee \overline{y_n})$.

scheme enumerates all activity mode variables for a single activity at each level of the enumeration tree and creates a node at that level for each variable that is set to true. In doing so, the size of the nonrenewable resource constraint is gradually reduced and clauses are added when necessary.

The minimum and maximum nonrenewable resource demand for an activity i are defined as:

$$r_{ik}^{min} = \min_{m=1,\ldots,M_i} r_{ikm} \text{ and } r_{ik}^{max} = \max_{m=1,\ldots,M_i;} r_{ikm} \tag{22.7}$$

The total minimum and maximum remaining nonrenewable resource demand is the sum of the individual minimal and maximal resource requests, as:

$$r_k^{sum-min} = \sum_{i \in U} r_{ik}^{min} \text{ and } r_k^{sum-max} = \sum_{i \in U} r_{ik}^{max} \tag{22.8}$$

where $U \subset V$ is the set of activities that have not been evaluated at previous levels of the search tree. Likewise, R'_k is used to denote the remaining nonrenewable resource availability after reduction of the resource use of the boolean variables (i.e., mode selections) that have been set to true.

The remaining nonrenewable resource constraint at each node will be evaluated to detect whether backtracking with or without adding constraint clauses is possible. The two evaluation rules applied at each node of the tree can be defined as follows:

1. If the remaining nonrenewable resource constraint is satisfied, continue with the other node(s) at the current level of the tree without the insertion of a clause. Formally, if $r_k^{sum-max} \leq R'_k$ then the constraint is satisfied.
2. If the remaining nonrenewable resource constraint is violated, continue with the other node(s) at the current level of the tree and insert a clause that consists of the negation of all activity modes selected up to the current branch of the tree. Formally, if $r_k^{sum-min} > R'_k$ then the remaining nonrenewable constraint is impossible to satisfy.

The enumeration scheme of the example project can be graphically presented in Fig. 22.3.

The enumeration search of the example has found only one conflict clause as $(\overline{x_1} \vee \overline{x_4})$ at node 3. Indeed, at node 3 of the tree, $r_k^{sum-min} = 2 > 1$ and hence, this constraint cannot be satisfied. The conflict $(\overline{x_1} \vee \overline{x_4})$ is added and the algorithm continues with node 4 of the tree. Note that, as an example, $r_k^{sum-max} = 7 \leq 7$ at node 5 of the enumeration tree, and hence, the remaining constraint is satisfied and the algorithm backtracks to the previous level.

In order to improve the efficiency of the enumeration scheme, a number of additional node reduction rules are applied during the search which can be summarized as follows:

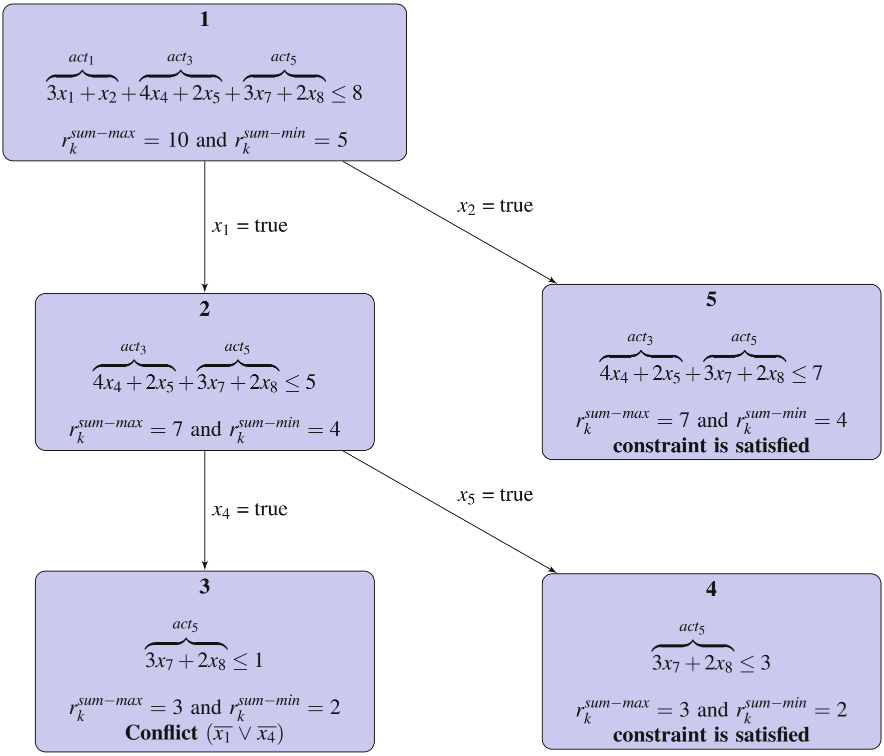

1
$\overbrace{3x_1 + x_2}^{act_1} + \overbrace{4x_4 + 2x_5}^{act_3} + \overbrace{3x_7 + 2x_8}^{act_5} \leq 8$
$r_k^{sum-max} = 10$ and $r_k^{sum-min} = 5$
x_1 = true
x_2 = true
2
$\overbrace{4x_4 + 2x_5}^{act_3} + \overbrace{3x_7 + 2x_8}^{act_5} \leq 5$
$r_k^{sum-max} = 7$ and $r_k^{sum-min} = 4$
5
$\overbrace{4x_4 + 2x_5}^{act_3} + \overbrace{3x_7 + 2x_8}^{act_5} \leq 7$
$r_k^{sum-max} = 7$ and $r_k^{sum-min} = 4$
constraint is satisfied
x_4 = true
x_5 = true
3
$\overbrace{3x_7 + 2x_8}^{act_5} \leq 1$
$r_k^{sum-max} = 3$ and $r_k^{sum-min} = 2$
Conflict $(\overline{x_1} \vee \overline{x_4})$
4
$\overbrace{3x_7 + 2x_8}^{act_5} \leq 3$
$r_k^{sum-max} = 3$ and $r_k^{sum-min} = 2$
constraint is satisfied

Fig. 22.3 Enumeration scheme of the example project

1. If an activity has an equal nonrenewable resource demand for all its modes then this activity can be deleted from the search and the total nonrenewable resource availability has to be reduced by this resource demand.
2. If the difference between the nonrenewable resource demand of activity i at mode m and its corresponding minimal nonrenewable resource demand is larger than the difference between the resource's availability and its total minimum remaining resource demand, then the activity mode can be set to false and removed from the formula. Formally, for each mode m of activity i, if $r_{ikm} - r_{ik}^{min} > R_k' - r_k^{sum-min}$ then remove the activity mode from the search.

No node improvement rules can be applied due to the small size of the example network. As an example, since $r_{ikm} - r_{ik}^{min}$ is equal to 2, 0, 0, 2, 0, 0, 1, and 0 for activities 1–5 of Fig. 22.1 and $R_k - r_k^{sum-min} = 8 - 5 = 3$, no activity mode can be removed from the search (improvement rule 2).

22.3.1.2 The Activity List SAT Mode Assignment Procedure

The procedure presented in this chapter makes use of a list with a dual role. First, the list serves as a *variable list* to solve the CNF and guarantees the selection of a single mode for each activity satisfying the nonrenewable constraints, if possible (see Sect. 22.3.1). Second, the list serves as a traditional *activity list* for the construction of a project schedule based on the selected modes (see Sect. 22.3.2). In the remainder of this chapter, we refer to this list as an activity list AL.

Despite the $\mathcal{NP}$-hardness of the SAT, research has proposed many advanced algorithms able to solve problem instances with up to thousands of literals and millions of constraints, which is far beyond the size of the SAT instances solved in our case (see, e.g., the SAT competition results of Kullmann 2006). Therefore, we rely on the efficient DPLL algorithm of Davis et al. (1962). The DPLL algorithm is a complete, backtracking-based algorithm for deciding the satisfiability of propositional logic formulae in the CNF. The algorithm sequentially selects literals (i.e., variables) from a pre-defined variable list and assigns a truth value to it in order to simplify the CNF formula. If this assignment leads to an unsatisfiable simplified formula (referred to as a conflict), the opposite value (false) is set to the selected literal and the algorithm continues. The unit propagation rule checks whether a clause is a unit clause, i.e., it contains only a single unassigned literal. When this is the case, this clause can only be satisfied by assigning the necessary value to make this literal true.

Assume a simple activity list AL $=$ {0,1,2,3,4,5,6,7,8,9} and a set L denoting the set of assigned literals at a given moment (where x and $\overline{x}$ can not belong to L at the same moment). The DPLL algorithm for the example CNF $= x_0 \wedge x_3 \wedge x_6 \wedge x_9 \wedge (x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2}) \wedge (x_4 \vee x_5) \wedge (\overline{x_4} \vee \overline{x_5}) \wedge (x_7 \vee x_8) \wedge (\overline{x_7} \vee \overline{x_8}) \wedge (\overline{x_1} \vee \overline{x_4})$ runs as follows:

1. Activity list AL $=$ {0,1,2,3,4,5,6,7,8,9}
   Unit clause rule x_0, x_3, x_6, x_9: $L = \{x_0, x_3, x_6, x_9\}$
   CNF:
   $(x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2}) \wedge (x_4 \vee x_5) \wedge (\overline{x_4} \vee \overline{x_5}) \wedge (x_7 \vee x_8) \wedge (\overline{x_7} \vee \overline{x_8})$
   $\wedge(\overline{x_1} \vee \overline{x_4})$

2. Selection of variable from AL, not in L: x_1: $L = \{x_0, x_3, x_6, x_9, x_1\}$
   CNF: $\overline{x_2} \wedge (x_4 \vee x_5) \wedge (\overline{x_4} \vee \overline{x_5}) \wedge (x_7 \vee x_8) \wedge (\overline{x_7} \vee \overline{x_8}) \wedge \overline{x_4}$
   Unit clause rule $\overline{x_2}, \overline{x_4}$: $L = \{x_0, x_3, x_6, x_9, x_1, \overline{x_2}, \overline{x_4}\}$
   CNF: $x_5 \wedge (x_7 \vee x_8) \wedge (\overline{x_7} \vee \overline{x_8})$
   Unit clause rule: x_5: $L = \{x_0, x_3, x_6, x_9, x_1, \overline{x_2}, \overline{x_4}, x_5\}$
   CNF: $(x_7 \vee x_8) \wedge (\overline{x_7} \vee \overline{x_8})$

3. Selection of variable from AL, not in L: x_7: $L{=}\{x_0, x_3, x_6, x_9, x_1, \overline{x_2}, \overline{x_4}, x_5, x_7\}$
   CNF: $\overline{x_8}$
   Unit clause rule: $\overline{x_8}$: $L = \{x_0, x_3, x_6, x_9, x_1, \overline{x_2}, \overline{x_4}, x_5, x_7, \overline{x_8}\}$

Table 22.1 The SAT mode assignment solution

Figure 22.1			Figure 22.2		
i	m	$(p_{im}, r_{i1m}, r_{i2m})$	i	L	p_{i1}
0	1	(0,0,0)	0	x_0	0
1	1	(4,2,3)	1	x_1	4
	2	(6,1,1)	2	$\overline{x_2}$	0
2	1	(2,1,0)	3	x_3	2
3	1	(3,3,4)	4	$\overline{x_4}$	0
	2	(5,1,2)	5	x_5	5
4	1	(2,1,0)	6	x_6	2
5	1	(2,2,3)	7	x_7	2
	2	(4,1,2)	8	$\overline{x_8}$	0
6	1	(0,0,0)	9	x_9	0

Note that no conflict has been generated during the DPLL algorithm since each assignment has led to a satisfiable simplified CNF formula. The selection of literals can be translated into sub-activity durations as shown in the last column of Table 22.1. The sub-activity durations (equal to zero when the corresponding activity mode has not been selected) of the table are input for the scheduling step of Sect. 22.3.2. An illustrative example to show the presence of conflict generation and algorithmic backtracking will be given in Sect. 22.3.3.2.

22.3.2 RCPSP Scheduling Step

The project scheduling step is a resource-constrained project scheduling problem where each activity has a single execution mode determined by the mode assignment step. The solution of SAT provides positive (in case the literal has been set to true) or zero (literal is set to false) duration sub-activities and hence determines the characteristics of the RCPSP instance. In this chapter, the scheduling step is performed based on the decomposition based genetic algorithm of Debels and Vanhoucke (2007). These authors present a genetic algorithm that makes use of an activity list to construct resource feasible schedules based on a forward and backward serial generation scheme and they have shown that their procedure outperforms the current state-of-the-art procedures.

While the details of this genetic algorithm will not be repeated here, a basic overview of the different elements is given along the following lines:

- Dual population: The population based heuristic splits the total number of generated schedules into two separate populations containing *left- and right-justified schedules*, inspired by the promising results found by Valls et al. (2005).
- Representation of a schedule: Based on the remarks by Debels et al. (2006) who have illustrated that a random key representation is very effective thanks to the use of the *topological ordering notation* (Valls et al. 2003), this notation has been

adapted to the dual population heuristic, as follows: the random key elements are equal to the activity finishing times for a left-justified schedule, and equal to the starting times for a right-justified schedule.

- Parent selection: Parents are selected using a *2-tournament selection* where two population elements from the population are chosen randomly, and the element with the best objective function value is selected. Afterwards, one element is randomly labelled as the father and the other element as the mother.
- Crossover operator: The combination of the genes of both parents is done by a two-point crossover operator based on a modified version of the peak crossover operator of Valls et al. (2008) that makes use of the *resource utilization ratio*. This ratio measures the resource utilization at time unit t allowing the selection of time intervals for which the resource utilization is high, so-called peaks, and time intervals with low resource utilization.
- Local search: The local search is based on an iterative *forward and backward search* (Li and Willis 1992) to improve the two separate populations containing left- and right-justified schedules.
- Decomposition approach: The genetic algorithm has been extended to a so-called *decomposition-based heuristic* which iteratively solves subparts of the project leading to the best results in literature.

While most research papers rely on two separate lists to solve the MRCPSP (one to determine the assignment of modes and a second to feed a schedule generation scheme), the algorithm presented here relies on a single list that takes both the mode assignments and the activity scheduling step into account.

Theorem 22.1. *Using a single activity list for the SAT mode assignment and the RCPSP scheduling steps does not exclude optimal solutions.*

Proof. It has been shown in Sect. 22.3 that a project network with multi-mode activities can be represented by a set of sub-activities while a corresponding feasible schedule can be represented by two subsets of this set of all sub-activities: One subset contains sub-activities with a positive duration and a corresponding starting time and the remaining subset contains sub-activities with a zero duration. It will be shown that any active project schedule has at least one sub-activity list AL that, when used as input lists by both the mode assignment step (SAT) and the scheduling step (RCPSP), leads to this schedule. If the existence of such an AL can be shown for any feasible active schedule, then this existence also holds for the optimal project schedule. Consider to that purpose a feasible solution, i.e., an active project schedule where each activity has a starting time (= scheduling step) and a positive duration defined by the selected mode (= mode assignment step). These activities belong to the subset of sub-activities with a positive duration while the remaining subset with zero duration sub-activities is obviously not visible in the project schedule. The existence of an AL that leads to the feasible project schedule using the scheduling and mode assignment steps can be shown through the following three steps.

1. Each schedule can be easily represented by an activity list using the unique standardized random key (RK) representation presented by Debels et al. (2006).

This representation consists of the starting times of the sub-activities with a positive duration in increasing order followed by set of remaining activity modes that are not part of the feasible schedule (i.e., the sub-activities with a duration of zero) and is an extension of the topological order representation of Valls et al. (2003, 2004). Since a feasible project schedule can be uniquely defined by its activity starting times, such an AL can always be constructed.

2. It has been shown by Debels et al. (2006) that such a standardized RK or AL has a unique correspondence with a project schedule where each positive duration activity has a starting time equal to its AL value, and this schedule can be generated by the use of the well-known serial schedule-generation scheme.
3. This unique RK or AL will also result in the mode assignments of the feasible project schedule, i.e., those sub-activities with a positive duration in the project schedule will be set to true in the SAT step, while the zero duration sub-activities will be set to false. Since the AL defines the ranking of sub-activities that will be selected by the SAT step, the SAT mode assignment step applied to this AL will select the first subset of sub-activities and set their values to true (i.e., with a positive duration). This will never generate a conflict since the activity list corresponds to a feasible project schedule. The remaining sub-activities will be set to false due to the clause that only one mode can be selected per activity, leading to the mode assignment represented in the feasible project schedule. □

The use of a single priority list for both the mode assignment step and the activity scheduling step is unique and in contrast with most meta-heuristic procedures to solve the MRCPSP in literature. The new solution approach presented in the current chapter transforms the MRCPSP instance to an RCPSP instance, where each multi-mode activity i is split into M_i single-mode sub-activities, and SAT restrictions are added to assure that only one sub-activity will be selected. Moreover, before applying the serial schedule-generation scheme to the RCPSP, the mode assignment is called using the same priority list as used for the RCPSP, and sub-activities that are not selected are set to zero. The example of Fig. 22.2 displays the general RCPSP instance used throughout the search to a high quality solution, without any SAT restriction forcing that only one mode can be selected per activity. A single priority list will transform this figure into a resource feasible schedule (i.e., the scheduling step) where exactly one mode per activity is selected (i.e., the mode assignment step).

22.3.3 Advantages of SAT Solvers

22.3.3.1 Pre-processing

The selection of the branching literals is an important factor for the efficiency (Hooker and Vinay 1995). Obviously, infeasible instances and instances with tight nonrenewable resource constraints might consume a lot of CPU time using the activity list discussed previously, since this list does not contain guiding information to select variables to branch. Therefore, at the initial start of the MRCPSP search, a random AL could lead to a high CPU consumption. Consequently, a pre-processing run using a selection rule (we use the greedy heuristic rule of Marques-Silva and Sakallah 1999) improves the decision assignment at each stage of the search process and leads to two advantages:

- Feasibility check: When the nonrenewable resource constraints are impossible to satisfy, the algorithm stops and there is no need to start the AL search.
- Clause learning: The information gathered during the initial pre-processing run can be saved to improve the remaining AL runs. This is explained in the next section.

22.3.3.2 Learning

Clause learning is an important technique in SAT, since Marques-Silva and Sakallah (1999) have shown that recording conflict-inducing clauses can help to prevent the occurrence of similar conflicts later on in the search. Rather than reducing the CPU time of a single search of a SAT instance, in our case there is a need to reduce the total CPU time used to repeatedly solve the SAT instance for all generated activity lists.

Therefore, we made a simplification and only introduced learning clauses with a maximum number of literals. More precisely, when a conflict arises due to an assignment at level three (or above) then a clause is added with the negation of the decision assignments used up to that level to prevent this conflict to occur again in the next searches of this instance using other activity lists.

The maximum level of three guarantees that only clauses with three or less literals are added to the SAT instance, keeping the instance size relatively stable. Computational tests have been done with a maximum of 4–10, and have shown that it leads to a higher CPU consumption.

In order to illustrate the effect of clause learning, a computational experiment has been set up that compares the SAT learning effect with an enumeration scheme that evaluates mode assignments in a similar way as the SAT procedure. More precisely, the enumeration scheme has exactly the same performance than the SAT algorithm under a single run, leading to exactly the same mode assignments, but it operates on modes rather than on literals and does not include clause learning. The test runs are done on test data truncated after a predefined number of schedules and compare the average number of backtracks used in both the enumeration and SAT approach since they can be considered as a proxy for the total computational time of the mode assignment step. Since the average number of backtracks varies heavily from instance to instance, a graph has been constructed for an example project instance shown in Fig. 22.4 that illustrates the main results of the experiments. In our computational tests, we have seen that approximately 25 % of the J30 problem instances clearly benefit from clause learning. As an example, the J308_6 instance

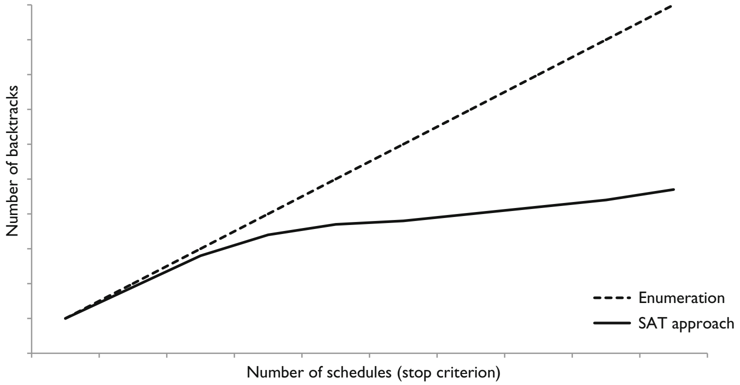


Fig. 22.4 An example graphical representation of the number of backtracks for two mode assignment procedures

benefits most from the SAT learning, and the computational tests have shown that the number of backtracks is equal to 131,362,000 and 5,500,000 for the enumeration approach and the SAT approach, respectively, under a stop criterion of 100 generated schedules. This number increases to 481,240,000 for the enumeration approach when the number of generated schedules is set to 1,000, while it stays relatively constant for the SAT approach, which illustrates that the incorporation of learning can lead to huge time reductions for some instances.

The graph shows that the initial performance of both mode assignment procedures is similar under a low stop criterion, but that the SAT approach benefits from learning while the enumeration approach is not able to do so. Indeed, while the number of backtracks is more or less linear with the number of predefined schedules set as a stop criterion for the enumeration approach, the SAT procedure is able to reach a relatively horizontal increase after a certain number of generated schedules, thanks to the incorporation of clause learning.

A small illustrative example is given along the following lines to show the presence of conflict generation, algorithmic backtracking, and the use of learning clauses. Assume a CNF $= (\overline{a} \vee \overline{c} \vee d) \wedge (\overline{b} \vee \overline{c} \vee \overline{d}) \wedge (\overline{a} \vee c \vee \overline{d}) \wedge (\overline{b} \vee c \vee d)$ and a simple activity list AL $= \{a, b, c, d\}$. The DPLL algorithm for the example CNF runs as follows:

1. Level 1. Selection of variable a from the AL, not in L: $L = \{a\}$
   CNF $= (\overline{c} \vee d) \wedge (\overline{b} \vee \overline{c} \vee \overline{d}) \wedge (c \vee \overline{d}) \wedge (\overline{b} \vee c \vee d)$

2. Level 2. Selection of variable b from the AL, not in L: $L = \{a, b\}$
   CNF $= (\overline{c} \vee d) \wedge (\overline{c} \vee \overline{d}) \wedge (c \vee \overline{d}) \wedge (c \vee d)$
   This instance can never be true, but the algorithm does not detect this since it does not generate a conflict

3. Level 3. Selection of variable c from the AL, not in L: $L = \{a, b, c\}$

CNF = $d \wedge \overline{d}$
Unit clause rule $d, \overline{d}$: $L = \{a, b, c, d, \overline{d}\}$
A conflict is generated since d and $\overline{d}$ cannot belong to L
Selection of other literal of variable c from AL, not in L: $L = \{a, b, \overline{c}\}$
CNF = $d \wedge \overline{d}$
Unit clause rule $d, \overline{d}$: $L = \{a, b, \overline{c}, d, \overline{d}\}$
A conflict is generated since d and $\overline{d}$ cannot belong to L
There are no other values for the variable c so the algorithm backtracks to level 2

4. Level 2. A learning clause $(\overline{a} \vee \overline{b})$
If this clause would exist at the start of the procedure, the unit clause rule of step 1 would assign $\overline{b}$ and the conflict would not be generated
Selection of other literal of variable b from AL, not in L: $L = \{a, \overline{b}\}$
CNF = $(\overline{c} \vee d) \wedge (c \vee \overline{d})$

5. Level 3. Selection of variable c from the AL, not in L: $L = \{a, \overline{b}, c\}$
CNF = d
Unit clause rule d: $L = \{a, \overline{b}, c, d\}$
This is the first valid assignment found with this activity list
The learning clause $(\overline{a} \vee \overline{b})$ will be inserted such that during the next search this conflict will not be generated, leading to a time saving

22.4 Adapted Pseudo Boolean Solver Approach

In the model presented earlier, each nonrenewable constraint is the subject of the enumeration scheme of Sect. 22.3.1.1, which leads to a set of clauses that need to be stored as an input file for the SAT solver (i.e., the DPLL procedure) that is called for each activity list generated during the search. When the size of the project network instance becomes relatively large, both in terms of the number of project activities and the number of nonrenewable resource constraints, the number of clauses translated from the pseudo boolean nonrenewable resource constraints can grow exponentially, leading to a large SAT input file and excessive use of memory. Table 22.2 illustrates the exponential growth of the number of clauses and the corresponding memory need for the PSPLIB (Kolisch and Sprecher 1996) instances. The column "SAT(3)" shows that the total disk space required to store the PSPLIB instance with the SAT approach of Sect. 22.3 grows very quickly up to almost 50 GB, using a stop criterion of 1 million clauses. From the J14 set on, several instances exceed this limit and are truncated in an early stage, as shown by column "#Ins-M".

Table 22.2 A comparison between the pure SAT and the SAT(k) approach

Set	#Var	#Ins	M (SAT(3))	M (SAT(4))	Avg.Cl(3)	Avg.Cl(4)	#Ins-M
J10	32	536	119	1.5	5,883	21.4	0
J12	38	547	801	2.1	32,435	25.4	0
J14	44	551	5,342	2.4	186,436	29.4	24
J16	50	550	14,445	2.7	454,055	33.4	162
J18	56	552	23,112	3.0	653,028	37.4	320
J20	62	554	26,759	3.4	694,061	41.4	378
J30	92	640	49,554	5.8	750,030	61.5	480

The abbreviations in the first row of the table can be explained along the following lines:

- #Var: Average number of variables in the SAT instance (equal to the number of activity modes).
- #Ins: Number of instances in each set.
- M (SAT(3)): Total disk space of SAT instance using the SAT approach of Sect. 22.3 (in MB).
- M (SAT(4)): Total disk space of SAT instance using the adapted SAT approach of this section (in MB).
- Avg.Cl(3): Average number of clauses generated using the SAT approach of Sect. 22.3.
- Avg.Cl(4): Average number of clauses generated using the adapted SAT approach of this section.
- #Ins-M: Number of instances leading to memory problems (i.e., $\geq$ 1 million added clauses) using the SAT approach of Sect. 22.3. Note that all instances can be solved by the adapted SAT approach as briefly discussed hereafter (i.e., #Ins-M = 0).

In order to avoid the heavy computational burden and the excessive memory requirement of the SAT(3) solution approach, the pseudo boolean nonrenewable resource constraints are not translated to CNF using the enumeration scheme of Sect. 22.3.1.1, but are used directly in the DPLL algorithm. Consequently, the two evaluation rules and the two node reduction rules of Sect. 22.3.1.1 are still applicable, but are dynamically used during the DPLL search. In doing so, this approach avoids the excessive memory increase of the SAT input file due to the enumeration of the boolean nonrenewable resource constraints in advance. The introduction of this approach leads to a dramatic reduction in the disk space (and hence the memory use) and the number of constraint clauses, as shown in columns "SAT(4)" and "Avg.Cl(4)" of Table 22.2. The reduction of memory has a beneficial effect on the initial memory allocation computation time, but does not speed up the rest of the search of the SAT solver in any way. Indeed, the results and the number of steps in the DPLL algorithm are the same for the SAT(3) and the SAT(4) approaches.

22.5 Computational Results

This section reports on computational results to evaluate the performance of the algorithm. The algorithm has been coded in C++ and tests have been run on a Dell Dimension DM051 with a Pentium D with a 2.80 GHz processor. The first benchmark test set is the well-known PSPLIB dataset which contains multi-mode project network instances generated by ProGen (Kolisch et al. 1995) with 10, 12, 14, 16, 18, 20, and 30 activities and with 2 renewable and 2 nonrenewable resources. The set is available from the ftp server of the University of Kiel (http://129.187.106.231/psplib/). Results are also compared with a second benchmark dataset taken from Boctor (1993), containing 240 instances with 50 and 100 activities and only renewable resource constraints.

The first computational results have been displayed in Table 22.3 for the new procedure (denoted by "This Work") under a stop criterion of 5,000 generated schedules. The values are average deviations from the optimal solution. The table shows that the new procedure is able to provide comparable results for some of the state-of-the-art procedures, but cannot outperform the best known results. The new procedure has also been truncated after 50,000 and 500,000 schedules in order to show the potential of the procedure to produce high-quality solutions. Although a comparison with the state-of-the-art results is not fair anymore, the table shows that near-optimal solution can be produced with the new procedure under high stop criterion values.

Table 22.4 reports results for the J30 instances as the average deviation from the minimal critical path length under four stop criterion values and compares the results

Table 22.3 Computational results for the PSPLIB dataset under a 5,000 schedule limit stop criterion

	J10	J12	J14	J16	J18	J20
Van Peteghem and Vanhoucke (2010)	0.01 %	0.09 %	0.22 %	0.32 %	0.42 %	0.57 %
Wang and Fang (2012)	0.12 %	0.14 %	0.43 %	0.59 %	0.90 %	1.28 %
Wang and Fang (2011)	0.10 %	0.21 %	0.46 %	0.57 %	0.94 %	1.39 %
Elloumi and Fortemps (2010) - v1	0.21 %	0.29 %	0.77 %	0.91 %	1.30 %	1.62 %
Elloumi and Fortemps (2010) - v2	0.14 %	0.24 %	0.80 %	1.14 %	1.53 %	2.09 %
Lova et al. (2009)	0.06 %	0.17 %	0.32 %	0.44 %	0.63 %	0.87 %
Jarboui et al. (2008)	0.03 %	0.09 %	0.36 %	0.44 %	0.89 %	1.10 %
Ranjbar et al. (2009)	0.18 %	0.65 %	0.89 %	0.95 %	1.21 %	1.64 %
Alcaraz et al. (2003)	0.24 %	0.73 %	1.00 %	1.12 %	1.43 %	1.91 %
Józefowska et al. (2001)	1.16 %	1.73 %	2.60 %	4.07 %	5.52 %	6.74 %
This Work (5,000)	0.07 %	0.16 %	0.32 %	0.48 %	0.56 %	0.80 %
	0.4 s	0.5 s	0.7 s	0.9 s	1.0 s	1.2 s
This Work (50,000)	0.00 %	0.01 %	0.05 %	0.06 %	0.08 %	0.12 %
	4.3 s	5.4 s	6.9 s	8.4 s	10.1 s	11.8 s
This Work (500,000)	0.00 %	0.00 %	0.01 %	0.01 %	0.01 %	0.02 %
	43.8 s	53.9 s	68.4 s	82.9 s	100.1 s	117.0 s

Table 22.4 Computational results for the PSPLIB J30 dataset under four different stop criteria (number of schedules)

	1,000	5,000	50,000	500,000
This Work	20.15 %	14.44 %	12.77 %	12.41 %
	2.8 s	4.5 s	25.1 s	210.1 s
Van Peteghem and Vanhoucke (2010)	15.30 %	13.75 %	13.31 %	13.09 %
	0.05 s	0.24 s	2.46 s	18.03 s

with the procedure of Van Peteghem and Vanhoucke (2010) which is described as the best performing procedure up to today. The results show that the new SAT based procedure is not able to outperform the best performing procedure when the stop criterion is set relatively low. However, when both procedures are truncated after a longer time period, the procedure of this chapter reports better results than the best known procedure in literature. Although the SAT procedure needs a higher CPU time for the same stop criterion (defined as a maximum number of generated schedules), it is able to find solutions which have never been found by the genetic algorithm of Van Peteghem and Vanhoucke (2010). It is worth noting that this genetic algorithm was able to report an average deviation from the critical path of 12.92 % when the stop criterion was set to 5,000,000 schedules (not shown in the table). These deviations were found after approximately 200 s, which corresponds to the time needed for the 500,000 schedules stop criterion of the SAT procedure. However, the latter procedure reports better results with average deviations of 12.41 %. It is also worth mentioning that four new best known solutions have been found with a stop criterion of 50,000 schedules, and another one with a stop criterion of 500,000 schedules. It should be noted that an increase from, e.g., 5,000 to 50,000 schedules (i.e., by a factor 10) does not lead to similar CPU time increase (the increase in CPU is equal to $\frac{25.1}{4.5} = 5.57$ which is lower than a factor 10). This can be explained by the introduction of the learning clauses that gradually avoids the generation of identical conflicts, leading to time savings when repeatedly solving the SAT instances. This is not the case for the J10 to J20 instances, which might indicate that the nonrenewable resource constraints are not a constraining factor making the clause learning less relevant. However, it should be noted that the computational results must be placed into the right perspective. Although the table shows that our procedure is able to generate high quality solutions which outperform the best known results found in literature, they often come at a higher computational cost. We have used the number of generated schedules as a stop criterion, since this is widely used in the academic literature. However, this approach assumes that the effort for one schedule is essentially the same in all methods (Kolisch and Hartmann 2006), which is not the case for our procedure. Due to the often CPU intensive search process during the mode assignment step which evaluates multiple mode assignments per schedule, this assumption is clearly violated and hence, the comparison in number of schedules not always fair. Since a fair and unambiguous comparison of CPU times is very hard, we have chosen to report CPU times in

Table 22.5 Computational results for the PSPLIB J30 dataset under five different stop criteria (CPU time, in seconds)

	1 s	5 s	30 s	120 s	300 s
This Work	31.03 %	16.66 %	12.75 %	12.64 %	12.54 %
	0	8	53	49	42
Van Peteghem and Vanhoucke (2010)	13.40 %	13.07 %	12.96 %	12.88 %	12.73 %
	454	243	34	20	17

Table 22.6 Computational results for the Boctor dataset under two different stop criteria

	50		100	
	1,000	5,000	1,000	5,000
This Work	31.37 %	25.11 %	38.58 %	30.03 %
Van Peteghem and Vanhoucke (2010)	27.36 %	23.41 %	29.70 %	24.67 %
Lova et al. (2009)	24.89 %	23.70 %	26.96 %	24.85 %
Alcaraz et al. (2003)	33.83 %	26.52 %	41.85 %	29.16 %

the tables, showing the promising character of our procedure in terms of solution quality, although the computational effort is often much higher.

However, in order to make the fair comparison complete, Table 22.5 shows a computational comparison similar to Table 22.4 but now truncated after a predefined running time. Since both algorithms have been developed by the same author(s) and tested on the same computer, it can be reasonably assumed that they have been programmed under the same implementation skills. The first rows of each algorithm display the average deviation from the minimal critical path length truncated after 1, 5, 30, 120, and 300 s, while the second rows display the number of instances for which a better solution is found than with the other solution procedure. As an example, the SAT procedure is able to find better solutions for 53 instances under a stop criterion of 30 s, while the solution procedure of Van Peteghem and Vanhoucke (2010) finds 34 better solutions. All other solutions have the same solution quality. The results show that the new SAT procedure is competitive with the best procedure currently available in the literature and even outperforms it when the running time stop criteria is set to 30 s or higher.

Finally, Table 22.6 reports results for the dataset of Boctor (1993) as the percentage deviation above the minimal critical path length. The procedure is not able to outperform the state-of-the-art procedures. However, when extending the stop criterion to 50,000 schedules, the deviations decreased to 23.26 and 24.42 %, for the 50 and 100 activity instances, respectively. A further increase to 500,000 schedules resulted in deviations of 22.87 and 23.11 %. However, using the procedure on these problem instances is not so relevant since these instances have no nonrenewable resources. Consequently, for these instances, there are no infeasible activity lists/mode assignment combinations, and hence, the advantage of the SAT approach to deal with inconsistencies of nonrenewable resources is no longer present.

22.6 Conclusions

In this chapter, a novel approach has been presented to solve the multi-mode resource-constrained project scheduling problem (MRCPSP). The algorithm splits the problem into a mode assignment step and a single mode project scheduling step. The mode assignment step is solved using a fast and efficient SAT solver. Due to excessive memory requirements, a number of small and straightforward adaptations to this solver have been implemented to solve the SAT problem instances in less memory. The single mode project scheduling step is solved using a current state-of-the-art RCPSP meta-heuristic from literature. When better RCPSP meta-heuristics become available in the literature, they can easily replace the current one, possibly leading to improved solutions.

The computational results for the MRCPSP have shown to be able to generate solutions comparable with the solution quality found by many state-of-the-art procedures, and outperforms them when the procedures run long enough. Moreover, the procedure was able to find better solutions on five problem instances under high stop criterion values. In our future research, it will be shown that the novel solution approach has potential to solve numerous extensions to the well-known MRCPSP problem, and hence, the solution approach will be used for alternative problem formulations or problem extensions.

References

Alcaraz J, Maroto C, Ruiz R (2003) Solving the multi-mode resource-constrained project scheduling problem with genetic algorithms. J Oper Res Soc 54:614–626

Bailleux O, Boufkhad Y, Roussel O (2006) A translation of pseudo-boolean constraints to SAT. J Satisf Bool Model Comput 2:191–200

Boctor F (1993) Heuristics for scheduling projects with resource restrictions and several resource-duration modes. Int J Prod Res 31:2547–2558

Brucker P, Drexl A, Möhring R, Neumann K, Pesch E (1999) Resource-constrained project scheduling: notation, classification, models, and methods. Eur J Oper Res 112:3–41

Chai D, Kuehlmann A (2005) A fast pseudo-boolean constraint solver. IEEE T Comput Aid D 24:305–317

Coelho J, Vanhoucke M (2011) Multi-mode resource-constrained project scheduling using RCPSP and SAT solvers. Eur J Oper Res 213:73–82

Cook S (1971) The complexity of theorem-proving procedures. In: Proceedings of the third annual ACM symposium on theory of computing, pp 151–158

Davis M, Logemann G, Loveland D (1962) A machine program for theorem proving. Comm ACM 5(7):394–397

Debels D, Vanhoucke M (2007) A decomposition-based genetic algorithm for the resource-constrained project scheduling problems. Oper Res 55:457–469

Debels D, De Reyck B, Leus R, Vanhoucke M (2006) A hybrid scatter search/electromagnetism meta-heuristic for project scheduling. Eur J Oper Res 169:638–653

Elloumi S, Fortemps P (2010) A hybrid rank-based evolutionary algorithm applied to multi-mode resource-constrained project scheduling problem. Eur J Oper Res 205:31–41

Herroelen W, Demeulemeester E, De Reyck B (1999) A classification scheme for project scheduling problems. In: Węglarz J (ed) Project scheduling: recent models, algorithms and applications. Kluwer Academic, Dordrecht, pp 1–26

Hooker J, Vinay V (1995) Branching rules for satisfiability. J Autom Reasoning 15:359–383

Jarboui B, Damak N, Siarry P, Rebai A (2008) A combinatorial particle swarm optimization for solving multi-mode resource-constrained project scheduling problems. Appl Math Comput 195:299–308

Józefowska J, Mika M, Rózycki R, Waligóra G, Węglarz J (2001) Simulated annealing for multi-mode resource-constrained project scheduling. Ann Oper Res 102:137–155

Kolisch R, Drexl A (1997) Local search for nonpreemptive multi-mode resource-constrained project scheduling. IIE Trans 29:987–999

Kolisch R, Hartmann S (2006) Experimental investigation of heuristics for resource-constrained project scheduling: an update. Eur J Oper Res 174:23–37

Kolisch R, Sprecher A (1996) PSPLIB: A project scheduling problem library. Eur J Oper Res 96:205–216

Kolisch R, Sprecher A, Drexl A (1995) Characterization and generation of a general class of resource-constrained project scheduling problems. Manag Sci 41:1693–1703

Kullmann O (2006) The SAT 2005 solver competition on random instances. J Satisf Bool Model Comput 2:61–102

Li K, Willis R (1992) An iterative scheduling technique for resource-constrained project scheduling. Eur J Oper Res 56:370–379

Lova A, Tormos P, Cervantes M, Barber F (2009) An efficient hybrid genetic algorithm for scheduling projects with resource constraints and multiple execution modes. Int J Prod Econ 117:302–316

Markov I, Sakallah K, Ramani A, Aloul F (2002) Generic ILP versus specialized 0–1 ILP: an update. In: Proceedings of the international conference on computer-aided design (ICCAD '02), pp 450–457

Marques-Silva J, Sakallah K (1999) GRASP: a search algorithm for propositional satisfiability. IEEE T Comput 48:506–521

Ranjbar M, De Reyck B, Kianfar F (2009) A hybrid scatter-search for the discrete time/resource trade-off problem in project scheduling. Eur J Oper Res 193:35–48

Talbot F (1982) Resource-constrained project scheduling problem with time-resource trade-offs: the nonpreemptive case. Manag Sci 28:1197–1210

Valls V, Quintanilla S, Ballestín F (2003) Resource-constrained project scheduling: A critical activity reordering heuristic. Eur J Oper Res 149:282–301

Valls V, Ballestín F, Quintanilla S (2004) A population based approach to the resource-constrained project scheduling problem. Ann Oper Res 131:305–324

Valls V, Ballestín F, Quintanilla S (2005) Justification and RCPSP: a technique that pays. Eur J Oper Res 165(2):375–386

Valls V, Ballestín F, Quintanilla S (2008) A hybrid genetic algorithm for the resource-constrained project scheduling problem. Eur J Oper Res 185(2):495–508

Van Peteghem V, Vanhoucke M (2010) A genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem. Eur J Oper Res 201:409–418

Wang L, Fang C (2011) An effective shuffled frog-leaping algorithm for multi-mode resource-constrained project scheduling problem. Inform Sci 181:4804–4822

Wang L, Fang C (2012) An effective estimation of distribution algorithm for the multi-mode resource-constrained project scheduling problem. Comput Oper Res 39:449–460

Węglarz J, Józefowska J, Mika M, Waligóra G (2011) Project scheduling with finite or infinite number of activity processing modes: a survey. Eur J Oper Res 208:177–205

Chapter 23
The Multi-Mode Capital-Constrained Net Present Value Problem

Zhengwen He, Nengmin Wang, and Renjing Liu

Abstract This chapter deals with a special resource-constrained multi-mode net present value problem, i.e., the capital-constrained multi-mode project payment scheduling problem where the objective is to assign activity modes and payments so as to maximize the net present value (NPV) of the contractor under the capital constraint. With the different payment patterns adopted, four optimization models are constructed using the event-based method. Metaheuristics, including tabu search and simulated annealing, are developed and compared with other two simple heuristics based on a computational experiment performed on a data set generated randomly. The results indicate that the loop nested tabu search is the most promising procedure for the problem studied. Moreover, the effects of key parameters on the NPV are studied and the following conclusions are drawn: The NPV rises with the increase of the initial capital availability, the payment number, the payment proportion, or the project deadline; the marginal return decreases as the initial capital availability goes up; the NPVs under the milestone event based payment pattern are not less than those under the other three payment patterns.

Keywords Capital constraints • Multi-mode • Net present value • Project scheduling

23.1 Introduction

Since the introduction of cash flows in project scheduling problems by Russell (1970), the problem of scheduling activities of a project in order to maximize its NPV has gained increasing attention throughout the literature. For the Max-NPV project scheduling problem, the research efforts have led to a large amount of models and algorithms under a wide variety of assumptions with respect to network representations, cash flow patterns, resource constraints, and time-cost tradeoffs.

Z. He (✉) • N. Wang • R. Liu
School of Management, Xi'an Jiaotong University, Xi'an, China
e-mail: zhengwenhe@mail.xjtu.edu.cn; wangnm@mail.xjtu.edu.cn; renjingl@mail.xjtu.edu.cn

C. Schwindt, J. Zimmermann (eds.), *Handbook on Project Management and Scheduling Vol. 1*, International Handbooks on Information Systems,
DOI 10.1007/978-3-319-05443-8_23

The resource-constrained multi-mode net present value problem is an important branch of the Max-NPV project scheduling problem where there exist resource constraints and activities can be accomplished in more than one way. In this branch, researchers have proposed a few models and algorithms, which are described as follows. Sung and Lim (1994) study a problem with positive and negative cash flows, and availability constraints imposed on capital and renewable resources. Resource-duration interactions are considered in analyzing the problem and a two-phase heuristic solution algorithm is exploited and tested with various numerical problems for its effectiveness and efficiency. Ulusoy and Özdamar (1995) construct a general model for the problem where the performance measures considered are the minimization of project duration and the maximization of net present value. They propose a heuristic iterative scheduling algorithm which consists of forward/backward scheduling passes where consecutive scheduling passes are linked by updated activity time windows. When the results of the iterative algorithm are compared with the results given by the initial forward schedule, a considerable amount of improvement in both performance criteria is observed. Taking renewable, nonrenewable, and doubly constrained resources into account simultaneously, Ulusoy et al. (2001) investigate a problem with four payment models, i.e., lump-sum payment at the completion of the project, payments at fixed event nodes, payments at equal time intervals, and progress payments. A genetic algorithm with a special crossover operator which can exploit the multi-component nature of the problem is proposed, and the efficiency of the algorithm is tested on 93 problems from the set of instances from the literature (Ulusoy and Özdamar 1995). Mika et al. (2005) consider a problem where a project is represented by an AoN network and the four payment models are considered. Two metaheuristics, namely simulated annealing and tabu search, are proposed to solve the problem and a comprehensive computational experiment is performed on a set of instances based on standard test problems constructed by the ProGen project generator. The metaheuristics are computationally compared, the results are analyzed and discussed and some meaningful conclusions are given. For the computational intractability of the problem, Chen et al. (2010) develop an ant colony system (ACS) algorithm where the AoA network of the problem is first converted into a mode-on-node graph, which next becomes the construction graph for the ACS algorithm. The proposed ACS approach is compared with the authors' implementations of genetic algorithm by Ulusoy et al. (2001), as well as simulated annealing and tabu search by Mika et al. (2005), on 55 randomly generated instances with from 13 up to 98 activities. On the basis of experimental results the authors state that their algorithm outperforms the other three metaheuristics.

It should be pointed out that in this branch, there is a special category problem named the capital-constrained multi-mode project scheduling problem (CCMPSP), where investment in project activities is constrained by capital constraint while payments are reinvested in the project to increase the capital availability. Considering the CCMPSP, Özdamar and Dündar (1997) introduce a model concerning housing projects, where capital is reduced by activity expenditures and augmented by the sales of flats. Activities can be carried out in different operating modes, and the rate of activity expenditures differs from mode to mode. The authors propose a flexible heuristic algorithm for solving the capital-constrained mode selection problem, where there exist general precedence relationships among activities, and the magnitude of precedence lags depend on the specific activity mode selected. The algorithm is tested using a typical housing project with real data and also by using hypothetical test problems. A similar scheduling problem is further analyzed in Özdamar (1998), where the contractor has to construct and reconstruct schedules during the progress of the project so as to maintain a positive cash balance dynamically. The author establishes a stochastic model involving probabilistic cash inflows which take place randomly over the progress of the project. Activities are performed in multiple processing modes with different durations and the same total cost, and the contractor has to decide on the rate of expenditure at each decision time in order to maintain a positive cash balance. A heuristic, which incorporates dynamic mode selection objectives, is proposed and computational experiments demonstrate that the heuristic provides satisfactory results regarding the feasibility of the schedules with respect to the project due date and the nonrenewable resource constraints.

Another interesting extension in this branch, which needs to be mentioned here, is the combination of the resource-constrained multi-mode net present value problem with the project payment scheduling problem, which leads to the resource-constrained multi-mode project payment scheduling problem (RCMPPSP). Concerning the RCMPPSP Ulusoy and Cebelli (2000) set up an interesting model where the goals of the contractor and the client are joined together. The purpose is to look for an equitable payment schedule, for which both the contractor and the client deviate from their respective ideal solutions by an equal percentage. The ideal solutions for the contractor and the client result from having a lump-sum payment at the start and at the end of the project, respectively. A double-loop genetic algorithm is proposed to solve the problem, where the outer loop represents the client and the inner loop the contractor. Ninety-three problems from the set of instances from the literature (Ulusoy and Özdamar 1995) are solved, and some computational results are reported. Kavlak et al. (2009) study a so-called client-contractor bargaining problem within the context of the RCMPPSP where two payment models, i.e. progress payments and payments at activity completion times, are considered. The bargaining objective is to maximize the bargaining objective function which reflects the two-party nature of the problem environment and seeks a compromise between the client and the contractor. Simulated annealing algorithm and genetic algorithm approaches are proposed as solution procedures for the problem and they are experimentally compared on the PSPLIB instances with 14, 20, and 30 activities. Also, sensitivity analysis is conducted for different parameters used in the model, namely the profit margin, the discount rate, and the bargaining power weights.

Based on the literature review above, this chapter involves a new problem named the capital-constrained multi-mode project payment scheduling problem (CCMPPSP) proposed by He et al. (2012). In a sense, the CCMPPSP can be recognized as a generalization of the CCMPSP with the consideration of the arrangement of payments or an extension of the RCMPPSP where the availability of the only one resource, i.e., capital, is a function of project schedule. In He et al. (2009), the authors have proven that the multi-mode project payment scheduling problem (MPPSP), which is a simplified version of the CCMPPSP in fact where the capital constraints are neglected, is strongly $\mathscr{NP}$-hard for general project networks, so the CCMPPSP must be strongly $\mathscr{NP}$-hard as well. In the CCMPPSP, besides the time-cost tradeoff there also exists the tradeoff between the delay of activities incurring cash outflows and the early completion of activities leading to cash inflows. Therefore, it is worthwhile to be investigated intensively and this research may have a great implication for the contractor to improve the project profitability.

We study the CCMPPSP using the event-based method by which the project is represented as an Activity-on-Arc (AoA) network and both cash inflows and outflows in the project are attached to events. During the course of the project, the amount of payments is calculated based on the contractor's accumulative earned value and the payment proportion, and the total amount of payments equals a given contract price. On the basis of He et al. (2009), we consider the following four different payment patterns.

- Milestone event based payment pattern (MEBPP): A payment occurs once a milestone event is realized by the contractor.
- Progress based payment pattern (PBPP): A payment is made by the client once the contractor's accumulative earned value reaches a given threshold.
- Expense based payment pattern (EBPP): The client makes a payment to the contractor once the latter's accumulative expense attains a certain value.
- Time based payment pattern (TBPP): Payments are connected to events periodically based on the time elapsed from the beginning of the project.

In the problem, the number of payments over the course of project is fixed while the time of payments is arranged in the light of one of the four payment patterns aforementioned. On the basis of the assumptions above, the objective is to assign activity modes and payments concurrently so as to maximize the NPV of the contractor under the constraint of capital availability.

The remainder of the chapter is organized as follows. In the next section, we provide the optimization models of the CCMPPSP. Metaheuristics are developed in Sects. 23.3 and 23.4 reports on the results of the computational experiments. Section 23.5 concludes the chapter.

23.2 Problem Formulation

23.2.1 Optimization Model with MEBPP

Consider a project represented as an AoA network $G = (V, E)$ with node set V and arc set E. The set of alternative execution modes for activity (i, j) ($(i, j) \in E$) is $\mathscr{M}_{ij}$ and the duration and cost of activity (i, j) under mode m ($m \in \mathscr{M}_{ij}$) are

p_{ijm} and c_{ijm}^{F-}, respectively. The expense of event i $(i \in V)$ is $c_i^{F-} (c_i^{F-} \geq 0)$: $c_i^{F-} = \sum_{(i,j) \in E_i^+} \left(\zeta_{ij} c_{ijm}^{F-}\right) + \sum_{(h,i) \in E_i^-} \left[(1 - \zeta_{hi})\, c_{him}^{F-}\right]$ where E_i^+ is the set of the activities with start event i, E_i^- the set of the activities with end event i, and ζ_{ij} $(0 \leq \zeta_{ij} \leq 1)$ and ζ_{hi} $(0 \leq \zeta_{hi} \leq 1)$ are the distribution proportion of the cost of activities (i, j) and (h, i) over their start and end events respectively. The earned value of event i is c_i^{F+} $(c_i^{F+} \geq 0)$: $c_i^{F+} = \sum_{(h,i) \in E_i^-} c_{hi}^{F+}$ where c_{hi}^{F+} is the earned value of activity (h, i). The compensation proportion is θ $(0 \leq \theta \leq 1)$ while the amount of the p-th $(p = 1, 2, \ldots, P; P \leq |V|)$ payment is c_p^{F+}. ICA, Φ, $\bar{d}$, and β are the contractor's initial capital availability, the contract price, the project deadline, and the interest rate per period, respectively.

The payment pattern adopted is MEBPP and the decision variables in the problem include following three groups.

x_{ijm}: Binary variable which equals 1 if activity (i, j) is performed with mode m and 0 otherwise.

y_{it}: Binary variable which equals 1 if event i is realized at period t and 0 otherwise.

z_{pi}: Binary variable which equals 1 if payment$p$ is assigned to event i and 0 otherwise.

Based on the notations defined above, the optimization model of the problem is constructed as follows.

$$\text{Max. } npv = \sum_{p=1}^{P} \left\{ c_p^{F+} \sum_{i \in V} \left[z_{pi} \sum_{t=ES_i}^{LS_i} (\exp(-\beta t) y_{it}) \right] \right\} - \sum_{i \in V} \left\{ c_i^{F-} \sum_{t=ES_i}^{LS_i} [\exp(-\beta t) y_{it}] \right\} \tag{23.1}$$

$$\text{s.t.} \quad \sum_{i \in V \setminus \{I\}} z_{pi} = 1 \quad (p = 1, 2, \ldots, P-1) \tag{23.2}$$

$$\sum_{p=1}^{P-1} z_{pi} \leq 1 \quad (i \in V \setminus \{I\}) \tag{23.3}$$

$$\sum_{m \in \mathscr{M}_{ij}} x_{ijm} = 1 \quad ((i, j) \in E) \tag{23.4}$$

$$c_i^{F-} = \sum_{(i,j) \in E_i^+} \left[\zeta_{ij} \sum_{m \in \mathscr{M}_{ij}} \left(c_{ijm}^{F-} x_{ijm} \right) \right] + \sum_{(h,i) \in E_i^-} \left[(1 - \zeta_{hi}) \sum_{m \in \mathscr{M}_{ij}} \left(c_{him}^{F-} x_{him} \right) \right] \quad (i \in V) \tag{23.5}$$

$$\sum_{t=ES_i}^{LS_i} y_{it} = 1 \quad (i \in V) \tag{23.6}$$

$$\sum_{t=ES_i}^{LS_i} (y_{it}t) + \sum_{m \in \mathscr{M}_{ij}} \left(p_{ijm}x_{ijm}\right) \leq \sum_{t=ES_j}^{LS_j} \left(y_{jt}t\right) \quad ((i,j) \in E) \tag{23.7}$$

$$c_p^{F+}{=}\theta \left[\sum_{i \in V} c_i^{F+} \sum_{t=0}^{t_p} y_{it} - \sum_{i \in V} c_i^{F+} \sum_{t=0}^{t_{p-1}} y_{it} \right] (p{=}1, 2, \ldots, P{-}1) \tag{23.8}$$

$$c_P^{F+} = \Phi - \sum_{p=1}^{P-1} c_p^{F+} \tag{23.9}$$

$$\sum_{i \in V} c_i^{F-} \sum_{t=0}^{\Gamma} y_{it} {\leq} ICA {+} \sum_{p=1}^{P} \left[c_p^{F+} \sum_{i \in V} z_{pi} \sum_{t=0}^{\Gamma} y_{it} \right] (\Gamma{=}0, 1, \ldots, \bar{d}) \tag{23.10}$$

$$\sum_{t=ES_I}^{LS_I} (y_{It}t) \leq \bar{d} \tag{23.11}$$

$$x_{ijm}, y_{it}, z_{pi} \in \{0, 1\} \tag{23.12}$$

where ES_i and LS_i are the earliest and latest precedence feasible times of event i, respectively; I is the end event in the project; t_p and t_{p-1} are the occurrence times of the p-th and $(p-1)$-th payments, respectively.

In the model, the objective (23.1) is to maximize *npv*, i.e., the contractor's NPV. Constraints (23.2) attach payments to events, making the milestone events to be identified (note that the last payment must be arranged at the end event of the project). Constraints (23.3) secure that at a certain event, only one payment can be arranged at most. Constraints (23.4) select one mode for each activity exactly and (23.5) calculate expenses for events. Constraints (23.6) make sure that the occurrence times of events must be within their time windows, and their precedence feasibility is maintained by (23.7). Constraints (23.8) compute payment amounts while constraint (23.9) forces that the sum of payments equals the contract price. Constraints (23.10) stipulate that the cumulative cash outflows cannot exceed the contractor's initial capital availability plus the cumulative payments obtained. A deadline is imposed to the project by constraint (23.11) and constraints (23.12) define the binary status of the decision variables.

23.2.2 Optimization Model with the Other Three Payment Patterns

- Optimization model with PBPP

  In PBPP, the payment threshold is determined by the contractor price, Φ, and the payment number, P. Once the contractor's accumulative earned value reaches to an integer multiples of $\lceil \Phi / P \rceil$ a payment occurs. Hence, this payment pattern can be formulated formally by

$$z_{p\xi}=1, \quad \xi=\left\{ i : y_{i\tau}=1, \tau = \min \left\{ \Gamma : \sum_{t=0}^{\Gamma} \sum_{i \in V} \left(c_i^{F+} y_{it} \right) \geq p \left\lceil \Phi / P \right\rceil \right\} \right\}$$
$$(p = 1, 2, \ldots, P-1) \qquad (23.13)$$

  The above constraints force the p-th payment to be attached to the earliest event of those making the contractor's accumulative earned value reach or surpass $p \lceil \Phi / P \rceil$. Replacing constraints (23.2) and (23.3) in the optimization model with MEBPP by constraints (23.13), we can get the optimization model with PBPP.
- Optimization model with EBPP

  Suppose that BC is the benchmark cost of the project for determining payment events in EBPP (note that in reality, BC is often approved of by the two sides in advance so it is a known parameter in the problem). According to EBPP, a payment occurs once the contractor's expense accumulates to an integer multiples of $\lceil BC / P \rceil$. In terms of the fact aforementioned, the optimization models with EBPP can be constructed by removing constraints (23.2) and (23.3) from the optimization model with MEBPP and adding the following constraints into it in the meantime.

$$z_{p\xi}=1, \quad \xi=\left\{ i : y_{i\tau}=1, \tau=\min \left\{ \Gamma : \sum_{t=0}^{\Gamma} \sum_{i \in V} \left(c_i^{F-} y_{it} \right) \geq p \left\lceil BC / P \right\rceil \right\} \right\}$$
$$(p=1, 2, \ldots, P-1) \qquad (23.14)$$

  Constraints (23.14) stipulate that the p-th payment is tied to the earliest event of those making the contractor's accumulative expense be equal to or greater than $p \lceil BC / P \rceil$.
- Optimization model with TBPP

  The arrangement of payment times in TBPP depends upon the project deadline, $\bar{d}$, and the payment number, P. Given $\bar{d}$ and P, a payment is made whenever the time of integer multiples of $\left\lceil \bar{d} \middle/ P \right\rceil$ has passed from the start of the project. Similarly, TBPP can be formulated as constraints (23.15):

$$z_{p\xi}=1, \xi=\left\{ i : y_{i\tau}=1, \tau=\min \left\{ \Gamma : \Gamma \geq p \left\lceil \bar{d} \middle/ P \right\rceil \right\} \right\} \quad (p=1, 2, \ldots, P-1) \qquad (23.15)$$

Constraints (23.15) ensure the p-th payment to be assigned to the earliest event of those occurring no earlier than $p\left\lceil \bar{d} \middle/ P \right\rceil$. Through substituting constraints (23.15) for constraints (23.2) and (23.3) in the optimization model with MEBPP, we can get the optimization model with TBPP.

23.3 Metaheuristics

In this section, two well-known metaheuristics, i.e., tabu search (TS) and simulated annealing (SA), are developed for the solution of the CCMPPSP. The metaheuristics are designed to solve the CCMPPSP with MEBPP and for the similarity of the four payment patterns, they can be extended to the other three payment patterns conveniently by the adjustment of the way to determine milestone events. Besides TS and SA we also present two other methods, namely multi-start iterative improvement (MSII) and random sampling (RS) (Mika et al. 2008; Waligóra 2008), to provide comparable computational efforts for the metaheuristics.

23.3.1 Common Features

23.3.1.1 Solution Representation

Referring to Kolisch and Hartmann (1999), we represent a solution for the problem using the following three lists.

- Payment event (PE): This list includes $|V|$ 0-1 elements. The i-th ($i = 1, 2, \ldots, |V|$) element is set at 1 if a payment is attached to event i and 0, otherwise.
- Mode assignment (MA): This list is composed of $|E|$ elements which define the mode of activities in set E.
- Shift vector (SV): This list is an $|V|$-element list and the i-th ($i = 1, 2, \ldots, |V|$) element, Δ_i ($\Delta_i \in [0, LS_i - ES_i]$), indicates how many units of event i's occurrence time deviating from ES_i.

Note that in PBPP, EBPP, and TBPP, a solution consists only of MA and SV in fact, and PE is determined by constraints (23.13), (23.14), and (23.15), respectively.

Let $\mathscr{E}$ be the set of eligible events, i.e., the unscheduled events whose all predecessor events have been scheduled, S_i^p be event i's occurrence time assigned preliminarily, S_i^u be event i's occurrence time assigned ultimately, and CCA and CEV be the contractor's cumulative capital availability and the cumulative earned value, respectively. A solution can be transformed into a project schedule according to the decoding procedure described as follows.

Step 1. Set $\mathscr{E} := \{\text{event } 1\}$, $S_1^u := 0, CCA := ICA - c_1^{F-}, CEV := c_1^{F+}$.

Step 2. Update $\mathscr{E}$, i.e., remove the scheduled events from $\mathscr{E}$ and add the new eligible events into it. If $\mathscr{E} = \emptyset$ stop the procedure and output all S_i^u; otherwise, calculate ES_i of the new eligible events in $\mathscr{E}$ and set $S_i^p := ES_i + \Delta_i$. Sort the events in $\mathscr{E}$ according to an ascending sequence of S_i^p. Note that if several events have a common S_i^p, place them at the same position in the sequence and process them concurrently in the subsequent steps. Denote the first event in $\mathscr{E}$ as *i_curr*. If *i_curr* is a milestone event go to step 3; otherwise, go to step 5.

Step 3. Set $CCA := CCA - c_{i\_curr}^{F-} + \theta(CEV + c_{i\_curr}^{F+})$. If $CCA \geq 0$ set $S_{i\_curr}^u := S_{i\_curr}^p$, $CEV := 0$, and go to step 2; otherwise, go to step 4.

Step 4. Denote the second event in $\mathscr{E}$ as *i_next* and calculate LS_{i_curr}. If $S_{i\_next}^p > LS_{i\_curr}$ stop the procedure and output the result that the solution is capital infeasible; otherwise, set $CCA := CCA - c_{i\_next}^{F-} + \theta c_{i\_next}^{F+}$. If $CCA \geq 0$ set $S_{i\_curr}^u := S_{i\_next}^p$, $S_{i\_next}^u := S_{i\_next}^p$, $CEV := 0$, and go to step 2; otherwise, process the third event in $\mathscr{E}$ as above. If the last event in $\mathscr{E}$ has been processed and CCA is still less than 0, stop the procedure and output the result that the solution is capital infeasible.

Step 5. Set $CCA := CCA - c_{i\_curr}^{F-}$ and $CEV := CEV + c_{i\_curr}^{F+}$. If $CCA \geq 0$ set $S_{i\_curr}^u := S_{i\_curr}^p$ and go to step 2; otherwise, go to step 6.

Step 6. Check whether there are milestone events in $\mathscr{E}$ or not. If the answer is false, stop the procedure and output the result that the solution is capital infeasible; otherwise, denote the first milestone event in $\mathscr{E}$ as *i_next* and calculate LS_{i_curr}. If $S_{i\_next}^p > LS_{i\_curr}$, stop the procedure and output the result that the solution is capital infeasible; otherwise, set $CCA := CCA - \sum_{i \in O_{next}} c_i^{F-} + \theta \sum_{i \in O_{next}} c_i^{F+}$ where $O_{next} = \{\psi : S_{i\_curr}^p < S_\psi^p \leq S_{i\_next}^p, \psi \in \mathscr{E}\}$. If $CCA \geq 0$ set $S_{i\_curr}^u := S_{i\_next}^p$, $S_i^u := S_{i\_next}^p (i \in O_{next})$, $CEV := 0$, and go to step 2; otherwise, process the second milestone event in $\mathscr{E}$ as above. If the last milestone event in $\mathscr{E}$ has been processed and CCA is still less than 0, stop the procedure and output the result that the solution is capital infeasible.

Note that in the above process, LS_{i_curr} is calculated using the CPM where $LS_I := \bar{d}$.

23.3.1.2 Objective Function

The above decoding procedure can be used to examine the capital feasibility of solutions. However, there also exists the probability of the generated schedule being a time infeasible schedule where the project deadline constraint is violated. In order not to overly restrict the search space, the deadline constraint is transformed into a soft constraint based on a time feasibility test function defined as

$$TFT = \max \left\{ 0, \sum_{t=ES_I}^{LS_I} (y_{It} t) - \bar{d} \right\} \tag{23.16}$$

During the searching process, if the *TFT* of a solution is greater than 0 its objective function value will be penalized according to the following formula.

$$npv = \Phi\left\{\exp\left[-\beta(\bar{d} + TFT)\right]\right\} - TC \tag{23.17}$$

where *TC* is the total cost of the project in which all activities are performed with the most expensive mode.

23.3.1.3 Preprocessing

Similar with Sprecher et al. (1997), we adapt the project data to the implementation of algorithms using a preprocessing procedure by which all inefficient modes and infeasible modes are eliminated. An inefficient mode is a mode with duration not shorter and, simultaneously, cost not less than any other mode of the considered activity. An infeasible mode is defined as follows. Suppose that $x^{min}(i, j, m)$ is an MA in which activity (i, j) is performed with mode m while other activities are performed with the mode corresponding to the minimal duration. Denote the ES_I under $x^{min}(i, j, m)$ as $ES_I\left[x^{min}(i, j, m)\right]$. If $ES_I\left[x^{min}(i, j, m)\right] > \bar{d}$ mode m is an infeasible mode for activity (i, j), since the project deadline constraint cannot be satisfied no matter how to adjust the mode of other activities.

23.3.1.4 Starting Solution

A feasible starting solution is generated according to the following steps.

Step 1. Select randomly $P - 1$ events from all events except for event I. On PE, set the element of the selected events and event I at 1 while that of the others at 0.

Step 2. On MA, select randomly an executable mode for each activity and compute ES_I. If $ES_I \leq \bar{d}$ accept the MA; otherwise, decline it and repeat this step until the condition that $ES_I \leq \bar{d}$ is satisfied.

Step 3. On SV, set $\Delta_i := 0$ $(i \in V)$, obtaining a solution. If the solution is capital feasible and $TFT = 0$, accept it; otherwise, select a Δ_i $(i \in V \setminus \{1\})$ randomly and change its value within $[0, LS_i - ES_i]$ by one unit. Judge whether the condition that the solution is capital feasible and $TFT = 0$ is satisfied or not. If the answer is false, repeat the above operations until the answer becomes true.

Note that the above operations will be executed in an infinite loop if no feasible solution exists. To avoid the occurrence of such a case, the operations will be terminated if the iteration number reaches a given value, which is set at 1,000$\cdot|E|$ in our implementation. If this happens, the result that there is no feasible solution for the problem will be output.

23.3.1.5 Neighbourhood

There are three operators utilized in the neighbourhood generation mechanism.

- Element swap (SW): On PE, select two elements whose values are 1 and 0 respectively and swap them. Check the capital feasibility of the generated solution and accept it if it is capital feasible. A set of neighbouring solutions can be created by selecting and swapping every such two elements on the list separately. Note that during this course the last element must not be selected so that its value, which is set at 1 in the starting solution, can remain unchanged.
- Mode change (MC): On MA, select one activity and change its mode to another one. Compute ES_I under the generated solution and then check its capital feasibility if the condition that $ES_I \leq \bar{d}$ is satisfied. Accept the generated solution if it is capital feasible. A set of neighbouring solutions can be obtained through changing the mode of the activity to every other executable mode and operating each activity on the list separately according to the way aforementioned.
- Deviation change (DC): On SV, select an element and change its value to another possible one. Check the capital feasibility of the generated solution and accept it if it is capital feasible. The value of the element can be changed to every other possible one and except for the first element, each element on the list can be selected and operated separately as above, generating a set of neighbouring solutions.

23.3.2 Tabu Search

23.3.2.1 Moves

In TS, a set of neighbouring solutions is created by the above three operators and the best one is chosen to act as the move to improve the current solution. Corresponding to the operators, the three moves are defined as follows.

- Move for SW: It is a couple of (position number of the chosen element whose value is 1, position number of the chosen element whose value is 0). For example, if element 1 in position 6 is swapped with element 0 in position 9 then the move is represented as (6,9), meaning that a payment is shifted from event 6 to event 9. In the meantime, the reverse move, which is denoted as (6), is added to the tabu list, forbidding a payment to be rearranged at event 6.
- Move for MC: It is a triple of (number of the chosen position, original value, new value). For example, if the value in position 5 is changed from 2 to 1 then the move is expressed as (5,2,1), implying that the mode of activity 5 is changed from 2 to 1. Consequently the reverse move, which has the form of (5,2), is added to the tabu list, preventing the mode of activity 5 from being changed back to 2.
- Move for DC: It is a triple of (number of the chosen position, original value, new value). For example, if the value in position 6 is changed from 1 to 2 then

the move is described as (6,1,2), indicating that the time deviation of event 6 is changed from 1 to 2. In consequence, the reverse move is denoted as (6,1) and added to the tabu list, forbidding the time deviation of event 6 to be assigned as 1 again.

23.3.2.2 Tabu List

The tabu list is managed according to the First-in-First-out rule. Whenever a move is performed, its reverse move is added to the bottom of the tabu list and the oldest existing move is removed from the top of the list. All the moves in the tabu list are forbidden. However, if a tabu move can generate a solution better than the best one found so far, its tabu status will be cancelled so that the algorithm can move to this solution. The length of the tabu list is set at a relative low value so that the current region can be explored intensively. However, if there are no admissible solutions in a neighbourhood or an assumed number of iterations have been performed without improving the objective function, the diversification will be employed to make the search jump into a different region. When this occurs the tabu list is cleared, the counter of iterations without improvement is reset, and the search is restarted from another different starting solution generated randomly.

23.3.2.3 Two Versions of Tabu Search

We design two tabu search algorithms with different searching structures. The first one is named random selection tabu search (RSTS) in which for each move, one of the three operators is selected randomly with an equal probability to generate neighbouring solutions. The second one, which is named loop nested tabu search (LNTS), is based on the fact that in the studied problem, the arrangement of payments on events is independent of that of activities' mode and events' occurrence time, and activities' mode can be arranged without any consideration of the assignment of events' occurrence time. LNTS is constructed with the following three nested loops.

- Inner loop: This loop finds the satisfactory SV under the given PE and MA.
- Middle loop: This loop, which contains the inner loop, looks for the satisfactory MA and SV based on a given PE.
- Outer loop: This loop, which includes the middle loop and can be regarded as the main program of LNTS, searches the satisfactory PE, MA, and SV, thus forming a desirable solution for the problem.

The stop criterion of both RSTS and LNTS is defined as an assumed number of the visited solutions, which is set at $10{,}000 \cdot |E|$ in this implementation.

23.3.3 Simulated Annealing

In SA, operators SW, MC, and DC are chosen in the same fashion as that applied in RSTS and when an operator is selected, it only generates one neighbouring solution in a random way for the transition. The cooling scheme of SA is described as follows.

- Initial temperature: The initial value of the temperature, T_0, is calculated from the following equation: $T_0 = \Delta npv_{max} / \ln \pi_0$, where Δnpv_{max} is the difference between the maximal objective value and the minimal one which are chosen from the objective values of 50 randomly generated neighbours of the initial solution, and π_0, which is set at 0.95 in this application, is the initial acceptance ratio defined as the number of accepted neighbours divided by that of proposed neighbours.
- Cooling rate: Beginning from T_0, the temperature is progressively reduced according to a given cooling rate, which is set at 0.88 in our implementation.
- Markov chain length: The length of Markov chains, which determines the number of transitions for a given value of the temperature, is set at $10 \cdot |E|$ in this application.
- Stop criterion: To compare the metaheuristics on a common computational basis, SA utilizes the same stop criterion as that used in RSTS and LNTS, i.e., the search process terminates when $10{,}000 \cdot |E|$ solutions have been visited.

23.3.4 Multi-Start Iteration Improvement and Random Sampling

To evaluate the performance of the metaheuristics, we use multi-start iteration improvement (MSII) and random sampling (RS) to generate benchmark schedules for comparison. MSII utilizes the same neighbourhood generation mechanism as that employed in RSTS. It starts from an initial solution and chooses the most improving neighbouring solution as the move. When there are no improving moves, it restarts with another feasible solution generated randomly. MSII terminates and takes the best solution found as the desirable one when the number of the visited solutions reaches $10{,}000 \cdot |E|$. In RS, $10{,}000 \cdot |E|$ feasible solutions are generated randomly and the best one is selected as the desirable solution.

23.4 Computational Experiment

23.4.1 Experimental Design

The algorithms are tested on a data set constructed by ProGen project generator (Kolisch and Sprecher 1996). The set consists of 600 instances and the parameter

Table 23.1 Parameter setting used to generate the data set

Parameter	Setting
Number of non-dummy activities, $\lvert E \rvert$	10, 20, 30, or 40
Number of instances generated under a given number of non-dummy activities	150
Number of initial and terminal activities	Randomly selected from 2, 3, and 4
Maximal number of successors and predecessors	4
Number of performing modes	2
Activity durations under mode 1, p_{ij1}	Randomly selected from interval [1, 10]
Activity costs under mode 1, c_{ij1}^{F-}	Randomly selected from interval [10, 20]
Activity durations under mode 2, p_{ij2}	$\eta_1 p_{ij1}$, where η_1 is randomly selected from interval [0.8, 1]
Activity costs under mode 2, c_{ij2}^{F-}	$\eta_2 c_{ij1}^{F-}$, where η_2 is randomly selected from interval [1, 1.2]
Earned values of activities, c_{ij}^{F+}	$\eta_3 c_{ij2}^{F-}$, where η_3 is randomly selected from interval [1.3, 1.5]

Table 23.2 Levels of the key parameters

Parameter	Value
Contractor's initial capital availability, ICA	$\eta_{ICA} ACE$, where $ACE = \frac{1}{2P}\sum_{(i,j)\in E}\left(c_{ij1}^{F-} + c_{ij2}^{F-}\right)$ and η_{ICA} is set at 0.8, 1, and 1.2
Payment number, P	3, 4, 5
Payment proportion for the contractor's accumulative earned value, θ	0.7, 0.8, 0.9
Project deadline, $\bar{d}$	$\eta_{\bar{d}}(LB_0^{max} - LB_0^{min}) + LB_0^{min}$, where LB_0^{max} and LB_0^{min} are the length of the critical path of the network when all the activities are performed with modes 1 and 2 respectively and $\eta_{\bar{d}}$ is set at 0.4, 0.6, and 0.8

setting used to generate the instances is represented in Table 23.1. The value of the key parameters, including $ICA, P, \theta, \bar{d}$, is set at three levels given in Table 23.2. A full factorial experiment of the four parameters with three levels results in 81 replicates for each instance and 48,600 ones for every type of the CCMPPSP. Other parameters are set as follows: $\zeta_{ij} = 0.5, \beta = 0.01$, $BC = \frac{1}{2}\sum_{(i,j)\in E}\left(c_{ij1}^{F-} + c_{ij2}^{F-}\right)$. The following five indices are defined to evaluate the performance of the algorithms.

- n_{best}: The number of instances for which the algorithm finds a solution equal to the best solution known, i.e., the best solution found by any of the algorithms.
- $\Delta_{UB}^{\varnothing}(\%)$: Average relative percent below the best solution known.

Table 23.3 Computational results of RSTS

Payment pattern	$\|E\|$	n_{best}	$\Delta_{UB}^{\varnothing}$	Δ_{UB}^{max}	$t_{cpu}^{\varnothing}$	t_{cpu}^{max}
MEBPP	10	8,462	0.48	1.37	9.14	15.33
	20	7,987	1.00	2.87	23.06	40.28
	30	7,205	1.58	4.13	37.49	67.44
	40	6,159	2.23	5.82	65.27	108.77
PBPP	10	8,643	0.39	1.08	8.23	14.08
	20	7,878	1.13	3.33	21.77	41.27
	30	7,152	1.80	4.11	33.34	63.24
	40	6,232	2.61	6.05	58.26	98.68
EBPP	10	8,523	0.42	1.11	8.13	13.87
	20	7,810	1.08	2.68	20.43	35.52
	30	7,084	1.84	4.44	34.44	62.40
	40	5,947	2.83	6.74	58.88	101.31
TBPP	10	8,655	0.32	0.94	8.25	15.10
	20	7,854	1.21	3.55	20.00	37.27
	30	6,923	1.81	4.37	34.65	66.14
	40	6,221	2.94	6.88	56.63	95.28

- $\Delta_{UB}^{max}(\%)$: Maximal relative percent below the best solution known.
- $t_{cpu}^{\varnothing}(s)$: Average computational time of the algorithm.
- $t_{cpu}^{max}(s)$: Maximal computational time of the algorithm.

The algorithms are coded and compiled with Visual Basic 6.0 and the computational experiment is performed on a Pentium-based personal computer with 1.60 GHz clock-pulse and 256 MB RAM.

23.4.2 Experimental Results

The computational results, which are presented in Tables 23.3, 23.4, 23.5, 23.6, and 23.7, show that as far as the comparison of algorithms is concerned, RSTS, LNTS, and SA outperform MSII and RS remarkably and the superiority grows with the number of activities. This result is not surprising and confirms the expectations that intelligent search algorithms generally get an advantage over simple search ones, and the advantage augments as the problem becomes more complex. Among the three intelligent search algorithms, LNTS gives the most encouraging results, especially for the larger instances. The reason for this result is described as follows. In LNTS, the inner loop is used to find the satisfactory SV under the given PE and MA and on the basis of the satisfactory SV returned by the inner loop, the middle loop seeks the satisfactory MA under the given PE. With the results of the inner and middle loops, the outer loop searches the satisfactory PE and thus forms a desirable solution for the problem. However, in RSTS and SA the three operators are selected

Table 23.4 Computational results of LNTS

Payment pattern	$\lvert E\rvert$	n_{best}	$\Delta_{UB}^{\varnothing}$	Δ_{UB}^{max}	$t_{cpu}^{\varnothing}$	t_{cpu}^{max}
MEBPP	10	8,926	0.33	1.04	8.26	15.51
	20	9,156	0.30	0.97	20.11	36.23
	30	9,569	0.17	0.76	35.13	63.92
	40	10,135	0.05	0.23	58.45	101.40
PBPP	10	9,240	0.26	0.63	7.09	12.67
	20	9,681	0.18	0.44	18.86	36.66
	30	10,033	0.09	0.32	28.75	58.59
	40	10,255	0.04	0.19	53.44	97.46
EBPP	10	9,269	0.37	0.90	7.10	13.44
	20	9,581	0.24	0.81	18.22	32.75
	30	10,085	0.11	0.47	30.32	55.55
	40	10,350	0.04	0.21	54.76	92.19
TBPP	10	8,975	0.28	0.74	6.82	14.04
	20	9,472	0.20	0.56	17.58	35.27
	30	9,931	0.13	0.43	31.74	60.11
	40	10,425	0.04	0.23	52.66	90.00

Table 23.5 Computational results of SA

Payment pattern	$\lvert E\rvert$	n_{best}	$\Delta_{UB}^{\varnothing}$	Δ_{UB}^{max}	$t_{cpu}^{\varnothing}$	t_{cpu}^{max}
MEBPP	10	7,216	0.69	4.19	9.03	16.73
	20	7,509	1.08	5.90	21.17	35.72
	30	7,819	1.28	8.87	36.34	69.15
	40	8,370	1.55	10.17	66.63	112.22
PBPP	10	7,125	0.75	3.95	7.87	15.05
	20	7,470	1.02	5.34	22.00	34.39
	30	8,145	1.36	8.52	32.78	57.82
	40	8,437	1.72	10.78	55.88	100.22
EBPP	10	6,848	1.03	4.35	8.29	16.25
	20	7,381	1.15	6.42	19.66	33.83
	30	7,839	1.37	8.07	32.35	58.82
	40	8,536	1.69	9.89	57.13	96.74
TBPP	10	7,085	0.67	3.84	7.46	14.83
	20	7,415	1.15	5.77	20.19	36.58
	30	7,951	1.46	8.38	33.73	68.24
	40	8,310	1.88	10.37	57.38	99.80

in a random fashion, without any consideration of the characteristics of the problem. This may lead to the searching structure of RSTS and SA less organized and thus less reasonable than that of LNTS, making the desirable solutions found by RSTS and SA worse than those obtained by LNTS especially when the problem gets larger.

Table 23.6 Computational results of MSII

Payment pattern	$\|E\|$	n_{best}	$\Delta_{UB}^{\varnothing}$	Δ_{UB}^{max}	$t_{cpu}^{\varnothing}$	t_{cpu}^{max}
MEBPP	10	5,152	3.31	7.25	5.82	9.99
	20	4,045	7.26	13.27	13.28	25.74
	30	1,927	13.35	20.64	29.37	46.83
	40	653	19.70	29.33	47.34	75.01
PBPP	10	5,531	2.86	5.73	5.14	9.69
	20	3,842	7.48	12.71	12.17	23.82
	30	1,559	12.25	19.85	23.68	43.36
	40	710	18.66	28.92	44.77	72.84
EBPP	10	5,360	2.21	6.01	4.87	8.28
	20	3,923	8.35	13.22	12.61	24.77
	30	1,826	13.72	20.13	25.83	45.21
	40	851	19.84	30.10	45.79	68.47
TBPP	10	5,615	2.19	6.23	4.29	8.35
	20	4,105	7.25	14.43	13.00	22.93
	30	1,995	13.43	19.75	26.55	41.76
	40	617	19.13	31.31	44.68	65.66

Table 23.7 Computational results of RS

Payment pattern	$\|E\|$	n_{best}	$\Delta_{UB}^{\varnothing}$	Δ_{UB}^{max}	$t_{cpu}^{\varnothing}$	t_{cpu}^{max}
MEBPP	10	2,134	4.55	11.22	3.66	7.33
	20	1,101	12.66	27.71	9.03	19.27
	30	218	20.17	46.23	18.86	39.36
	40	14	31.94	81.80	34.55	67.77
PBPP	10	2,371	3.90	9.42	3.24	6.65
	20	1,521	11.38	24.68	6.94	19.24
	30	322	21.37	45.38	16.73	31.65
	40	25	32.63	78.46	29.32	56.37
EBPP	10	1,986	5.30	12.32	2.89	7.01
	20	1,358	12.51	26.93	7.77	15.35
	30	264	19.94	48.65	15.69	35.21
	40	19	30.75	79.24	30.11	52.66
TBPP	10	2,180	4.73	10.84	3.15	5.91
	20	1,563	11.25	28.93	7.63	18.24
	30	300	20.00	47.36	15.55	35.47
	40	31	32.42	76.24	28.25	54.28

A further comparison of RSTS and SA shows that RSTS performs better for smaller number of activities whereas the efficiency of SA grows with the value of $|E|$, both in terms of indices n_{best} and $\Delta_{UB}^{\varnothing}$. Moreover, the index Δ_{UB}^{max} for SA is greater than for RSTS, suggesting that if SA fails to find a good solution, its result tends to be worse than the one obtained by RSTS. The above facts may be explained by the random nature of SA, which makes SA work better for the larger problems in general. Ultimately, as to MSII and RS, it is quite understandable that the former can get better results since it employs the same neighbour generation mechanism as that used in RSTS. RS performs worst from among the five algorithms and it becomes worse rapidly with the increase of the problem scale. This rather proves the fact that the problem is too complicated for a random algorithm to generate good results.

The computational times are ones to be expected—RS is the fastest, then MSII, and RSTS, LNTS, and SA run more slowly than MSII. This follows from the fact that RS does not generate neighbours so the times obtained for RS are significantly shorter than those for the others. Compared with the three intelligent search algorithms, MSII owns a simpler searching process hence it requires less computational efforts to get a satisfactory solution for the problem.

Concerning RSTS, LNTS, and SA, LNTS works a little faster than RSTS and SA and this phenomenon is explained below. Recall that during the searching process of the three algorithms, operators SW, MC, and DC are used to generate neighbouring solutions based on the current one. In RSTS and SA, the check for the capital feasibility is required in all the three operators while in LNTS, this is only necessary in operator DC. Furthermore, in RSTS and SA when operator MC is selected to generate neighbouring solutions, the MA is changed while the SV remains unchanged. This increases the probability of the generated solution being time infeasible since $LS_i - ES_i$ varies with the change of the MA. However, in LNTS this deficiency is avoided by the loop nested searching structure where operator DC works after the MA is determined. The above facts may explain why LNTS tends to run faster than RSTS and SA, although LNTS seems to own a more complicated searching structure.

The effects of the key parameters on the contractor's NPV are shown in Table 23.8 where the results are obtained by LNTS. From the table it can be seen clearly that the contractor's NPV goes up with the increase of η_{ICA}, P, θ, or $\eta_{\bar{d}}$.

Table 23.8 Effects of the key parameters on the contractor's NPV

Parameter	Value	MEBPP	PBPP	EBPP	TBPP
η_{ICA}	0.8	112.51	109.76	110.67	110.54
	1.0	118.82	114.14	114.81	115.07
	1.2	120.36	115.20	116.00	116.36
P	3	107.51	103.02	104.93	105.28
	4	117.15	113.76	113.64	113.72
	5	127.04	121.91	121.84	122.06
θ	0.7	104.93	100.91	101.96	102.83
	0.8	116.85	113.43	113.22	112.75
	0.9	129.81	124.74	124.91	125.28
$\eta_{\bar{d}}$	0.4	114.33	109.63	110.67	109.90
	0.6	117.00	113.23	113.52	114.01
	0.8	120.19	116.18	115.84	116.95

The reasons for this fact are obvious: Firstly, increasing the value of η_{ICA} results in an increased value of ICA. A higher ICA can loosen the capital constraint to some extent, causing the NPV improved. Secondly, increasing P makes that some payments occur at earlier events and hence the obtained NPV cannot be worse than the one obtained for the smaller P. Thirdly, increasing the value of θ leads to the greater part of the cash inflows to occur earlier, thus increasing the NPV. At last, when $\eta_{\bar{d}}$ increases $\bar{d}$ augments subsequently, so, some activities may be executed with their cheaper mode and hence the NPV is improved.

Another interesting phenomenon, which can be observed from Table 23.8, is that as η_{ICA} increases the rate of increase in the contractor's NPV decreases. The explanation for this phenomenon is that when the capital availability increases, the contractor always employs the increased capital to improve the arrangement of the activities and events which can bring the greatest increment in the NPV. Therefore, the contractor's marginal return from such adjustments goes down with the increase of the capital availability. Table 23.8 also shows that the contractor's NPVs in MEBPP are not less than the corresponding ones in the other three payment patterns. This is because in MEBPP the milestone events are chosen without any limitations while in the other three payment patterns, they are determined by constraints (23.13), (23.14), and (23.15), respectively. This in fact withdraws the freedom of the contractor's selecting payment events, thus making the NPVs decreased.

23.5 Conclusions

This paper considers the CCMPPSP where the objective is to assign activity modes and payments concurrently so as to maximize the NPV under the constraint of capital availability. In terms of the different payment patterns adopted, the optimization models of the problem are constructed using the event-based method. For the $\mathscr{N}\mathscr{P}$-hardness of the problem, two versions of TS, namely RSTS and LNTS, which own different searching structures, and SA are developed and compared with MSII and RS on the basis of a computational experiment performed on a data set generated by ProGen. The results indicate that among the five algorithms, LNTS is the most promising procedure for the problem studied. The influences of several key parameters on the NPV of the contractor are also investigated and the following conclusions are drawn: The NPV goes up with the increase of ICA, P, θ, or $\bar{d}$, the contractor has a decreasing marginal return as ICA goes up, and the NPVs in MEBPP are not less than those in the other three payment patterns.

References

Chen WN, Zhang J, Chung HSH, Huang RZ, Liu O (2010) Optimizing discounted cash flows in project scheduling: an ant colony optimization approach. IEEE Trans Syst Man Cybern C 40(1):64–77

He Z, Wang N, Jia T, Xu Y (2009) Simulated annealing and tabu search for multi-mode project payment scheduling. Eur J Oper Res 198(3):688–696

He Z, Liu R, Jia T (2012) Metaheuristics for multi-mode capital-constrained project payment scheduling. Eur J Oper Res 223(3):605–613

Kavlak N, Ulusoy G, Şerifoğlu FS, Birbil Şİ (2009) Client-contractor bargaining on net present value in project scheduling with limited resources. Nav Res Log 56(2):93–112

Kolisch R, Hartmann S (1999) Heuristic algorithms for the resource-constrained project scheduling problem: classification and computational analysis. In: Węglarz J (ed) Project scheduling: recent models, algorithms, and applications. Kluwer Academic, Boston, pp 147–178

Kolisch R, Sprecher A (1996) PSPLIB: a project scheduling problem library. Eur J Oper Res 96(1):205–216

Mika M, Waligóra G, Węglarz J (2005) Simulated annealing and tabu search for multi-mode resource-constrained project scheduling with positive discounted cash flows and different payment models. Eur J Oper Res 164(3):639–668

Mika M, Waligóra G, Węglarz J (2008) Tabu search for multi-mode resource-constrained project scheduling with schedule-dependent setup times. Eur J Oper Res 187(3):1238–1250

Özdamar L (1998) On scheduling project activities with variable expenditure rates. IIE Trans 30(8):695–704

Özdamar L, Dündar H (1997) A flexible heuristic for a multi-mode capital constrained project scheduling problem with probabilistic cash inflows. Comput Oper Res 24(12):1187–1200

Russell AH (1970) Cash flows in networks. Manag Sci 16(5):357–373

Sprecher A, Hartmann S, Drexl A (1997) An exact algorithm for project scheduling with multiple modes. OR Spektrum 19(3):195–203

Sung CS, Lim SK (1994) A project activity scheduling problem with net present value measure. Int J Prod Econ 37(2–3):177–187

Ulusoy G, Cebelli S (2000) An equitable approach to the payment scheduling problem in project management. Eur J Oper Res 127(2):262–278

Ulusoy G, Özdamar L (1995) A heuristic scheduling algorithm for improving the duration and net present value of a project. Int J Oper Prod Manag 15(1):89–98

Ulusoy G, Sivrikaya-Serifoglu F, Sahin S (2001) Four payment models for the multi-mode resource constrained project scheduling problem with discounted cash flows. Ann Oper Res 102(1–4):237–261

Waligóra G (2008) Discrete-continuous project scheduling with discounted cash flows: a tabu search approach. Comput Oper Res 35(7):2141–2153

Chapter 24
The Resource-Constrained Project Scheduling Problem with Work-Content Constraints

Philipp Baumann, Cord-Ulrich Fündeling, and Norbert Trautmann

Abstract For executing the activities of a project, one or several resources are required, which are in general scarce. Many resource-allocation methods assume that the usage of these resources by an activity is constant during execution; in practice, however, the project manager may vary resource usage by individual activities over time within prescribed bounds. This variation gives rise to the project scheduling problem which consists in allocating the scarce resources to the project activities over time such that the project duration is minimized, the total number of resource units allocated equals the prescribed work content of each activity, and precedence and various work-content-related constraints are met.

This chapter compares a priority-rule based method known from the literature against a recent MILP formulation on a benchmark test set of small-sized problem instances. Our computational results indicate that the priority-rule based method derives feasible solutions to all instances of the test set. The MILP formulation provides feasible solutions to a surprisingly large number of instances; most of these solutions are optimal or near-optimal, and on these instances the MILP formulation outperforms the priority-rule based method.

Keywords Project scheduling • Resource constraints • Work-content constraints • Mixed-integer linear programming

24.1 Introduction

A dynamic environment requires firms to execute many projects such as the development of new products and services. A project is a unique endeavor that can be divided into activities each of which requires time and scarce resources for

P. Baumann (✉) • N. Trautmann
Department of Business Administration, University of Bern, Bern, Switzerland
e-mail: philipp.baumann@pqm.unibe.ch; norbert.trautmann@pqm.unibe.ch

C.-U. Fündeling
Galenicare Management SA, Bern, Switzerland
e-mail: cord-ulrich.fuendeling@galenicare.com

C. Schwindt, J. Zimmermann (eds.), *Handbook on Project Management and Scheduling Vol. 1*, International Handbooks on Information Systems,
DOI 10.1007/978-3-319-05443-8_24

its completion. Due to technological or organizational restrictions, these activities usually must be performed according to precedence constraints. The planning of such projects includes *temporal scheduling*, which consists of determining the early start schedule, the late start schedule, and the slack times for each of the activities, and *resource allocation*, which consists in allocating scarce resources to the execution of the activities over time. The efficient allocation of scarce resources to the execution of the project activities is of particular importance for companies that wish to remain competitive. Many project scheduling methods assume that the activities have fixed durations and constant resource requirements. These assumptions are often too restrictive for practical applications, in which the decision maker may change the resource usage of an activity over time. Examples of such applications can be found in the pharmaceutical and software development industries. Typically, the availability of a single renewable work-content resource (e.g., labor) is constant throughout the project, and an individual amount of work content (e.g., number of person-days) is specified for each activity. Activities can be executed in various ways, i.e., the number of units of the work-content resource allocated to an activity may vary over time, as long as the requirement for the total work content is met. This flexibility allows for a more efficient usage of the work-content resource. In addition to the work-content resource an activity often requires additional resources, such as tools or supplies. We assume that the amount of the work-content resource used by an activity determines the activity's requirement for further resources. Currently, most project planning software packages allow the specification of individual work contents for activities, which emphasizes the relevance of this concept to project managers.

We consider the problem of scheduling the activities of a project to minimize the project duration; thereby, the activities (i) are subject to finish-start precedence relationships, (ii) impose a given workload on the work-content resource, (iii) require units of multiple resources in addition to the work-content resource, and (iv) may use variable amounts of the work-content resource during their execution, subject to some organizational restrictions. These restrictions include a lower and an upper bound on the usage of the work-content resource and a minimum time lag between consecutive changes of the usage of the work-content resource. We refer to this time lag as the minimum block length.

For this project scheduling problem, Fündeling (2006) presents an exact branch-and-bound method. Although this approach is tailored to the specific problem setting, only few and small-sized problem instances can be solved to optimality. To tackle large-scale instances, Fündeling and Trautmann (2010) develop a priority-rule method that schedules activities iteratively using a serial schedule-generation scheme. In each iteration, a start time and a feasible resource profile are determined for an activity. To construct the resource profile, the work-content resource is allocated iteratively period by period. A consistency test is thereby used to exclude allocations that would violate the minimum block length constraint. The quality of the generated schedules cannot be evaluated reliably because of the lack of a sufficiently large number of optimal schedules. Both approaches address the problem in which, in a feasible schedule, the total number of resource units allocated to each activity must coincide with its prescribed work content. Alongside, the discrete time-resource tradeoff problem has been discussed in the literature. In this problem, only the work-content resource is considered, and an execution mode must be selected for each activity; each mode corresponds to a combination of a duration and a constant resource usage such that the total number of resource units allocated to the activity is greater than or equal to its prescribed work content. Respective literature surveys are provided in Fündeling and Trautmann (2010), Węglarz et al. (2011), and Chap. 21 of this book. Recently Naber and Kolisch (2013) have presented four different MILP formulations for a relaxation of the problem discussed in this chapter; they assume that the total number of resource units allocated to an activity is to be greater than or equal to its prescribed work content, and that the capacity of the renewable resources can be divided continuously among the activities.

In this chapter, which draws on Baumann and Trautmann (2013), we formulate the project scheduling problem with work-content constraints as a mixed-integer linear program (MILP). Thereby we use the priority-rule based method presented in Fündeling and Trautmann (2010) to compute an upper bound on the minimum project duration; the length of the planning horizon for the MILP model is set according to this upper bound. Due to its general structure, the proposed MILP formulation can easily be adapted to different project scheduling problems, including the well-known RCPSP; in this sense, we also contribute to the recent development of novel MILP formulations for various types of project scheduling problems (cf., e.g., Rieck et al. 2012; Bianco and Caramia 2013; Chaps. 2, 5, and 17 of this book). We have applied the proposed formulation to 480 problem instances with 10 activities that were introduced in Fündeling and Trautmann (2010). The proposed model solves 389 out of the 480 instances to optimality within short CPU times.

The remainder of the chapter is organized as follows. In Sect. 24.2, we describe the planning problem. In Sect. 24.3, we summarize the priority-rule based method for computing the upper bound. In Sect. 24.4, we present the mixed-integer linear programming formulation. In Sect. 24.5, we report our computational results. In Sect. 24.6, we provide some concluding remarks.

24.2 Planning Problem

Given are a set of activities V, a set of discrete resources $\mathscr{R}$, and a set of finish-start precedence relationships between the activities. We assume that all resources are renewable and that their capacities are constant over time. The set of resources $\mathscr{R}$ consists of the work-content resource k^* and the non-work-content resources $k \in \mathscr{R} \setminus \{k^*\}$. For each activity i, a work content $wc_i \in \mathbb{Z}_{\geq 0}$ is given that represents the total number of units of the work-content resource k^* required for its execution. The allocation of the work-content resource per period can vary over integer values between a prescribed lower bound $r_{ik^*}^{min}$ and a prescribed upper bound $r_{ik^*}^{max}$. The requirements for the non-work-content resources depend on the usage of the work-content resource. Here, a linear relation between the requirement of the non-work-content resources and the usage of the work-content resource is assumed, i.e., an increase (decrease) in the usage of the work-content resource results in a proportional increase (decrease) in the requirements for further resources. However, because we assume that all resources are discrete in nature, fractional requirements are rounded up. Let r_{ikt} denote the requirement for resource $k \in \mathscr{R}$ by activity i in period t. The requirement r_{ikt} for resource $k \in \mathscr{R} \setminus \{k^*\}$ is computed as follows.

$$r_{ikt} = \lceil r_{ik}^{min} + s_{ik}(r_{ik^*t} - r_{ik^*}^{min}) \rceil$$

where r_{ik}^{min} denotes the minimal requirement for resource k by activity i, and s_{ik} represents the increment per additional unit of the work-content resource k^*. The parameter s_{ik} equals

$$s_{ik} = \frac{r_{ik}^{max} - r_{ik}^{min}}{r_{ik^*}^{max} - r_{ik^*}^{min}}$$

where r_{ik}^{max} denotes the maximal requirement for resource k by activity i; we set $s_{ik} = 0$ if $r_{ik^*}^{min} = r_{ik^*}^{max}$.

The minimum block length is denoted by ℓ, i.e., a time lag of at least ℓ periods is required between consecutive changes of the usage of the work-content resource. Such changes include the start and the end of an activity.

A feasible allocation of the work-content resource to the execution of the activities is sought such that the precedence relationships are satisfied, all activities are scheduled without interruption, the capacities of the resources are never exceeded, and the duration of the project is minimized.

In contrast to the well-known resource-constrained project scheduling problem RCPSP, for some instances of the problem at hand, in each optimal solution some activities are started later than the earliest resource- and precedence-feasible point in time. Figure 24.1 illustrates this peculiarity for an instance with $n = 3$ activities with work contents $wc_1 = 3$, $wc_2 = 9$, and $wc_3 = 7$, a single precedence relationship $1 \prec 2$, one renewable resource $k^* = 1$ with capacity $R_1 = 2$ (which also represents

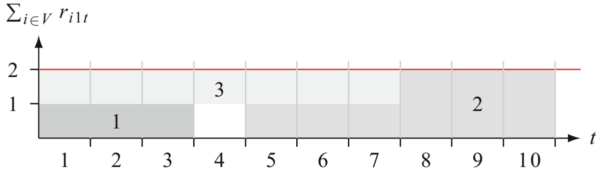


Fig. 24.1 Optimal schedule

the work-content resource), and minimum block length $\ell = 3$; the minimum and the maximum requirements for the work-content resource are given by $r_{1,1}^{min} = r_{1,1}^{max} = 1$, $r_{2,1}^{min} = 1, r_{2,1}^{max} = 2$, and $r_{3,1}^{min} = r_{3,1}^{max} = 1$. If activity $i = 2$ starts at point in time 3, it cannot be completed before point in time 12 due to the minimum block length.

24.3 Upper Bound on the Project Duration

In this section, we briefly summarize the priority-rule based heuristic for computing an upper bound on the project duration; this upper bound is used to determine the length of the planning horizon in the MILP model. For a detailed presentation of the priority-rule based heuristic, we refer to Fündeling and Trautmann (2010).

In each iteration of the priority-rule based heuristic, one activity j is scheduled. To this end, a feasible allocation of the resources to the execution of activity j is determined (possibly by full enumeration) such that activity j starts at the earliest precedence-feasible point in time; thereby it is aspired to allocate in each period as many units of the work-content resource as possible to the execution of activity j. If no such allocation exists, then the earliest possible start time of activity j is successively increased until activity j has been scheduled. For determining the corresponding resource profile for activity j, Fündeling and Trautmann (2010) apply a consistency test, which is based on the SUBSET SUM decision problem (cf. Garey and Johnson 1979) and allows to identify partial resource profiles that cannot be completed to a feasible profile.

For selecting the next activity to be scheduled, Fündeling and Trautmann (2010) propose to apply a priority rule, e.g., the so-called longest-path-following (LPF) rule or the most-total-successors (MTS) rule, or to generate random priority values, preferably in a multi-start implementation. The computational results presented in Fündeling and Trautmann (2010) indicate that the best results are in general obtained by the multi-start procedure.

24.4 MILP Scheduling Model

In this section, we present a formulation of the project scheduling problem described in Sect. 24.2 as an MILP. We closely follow Baumann and Trautmann (2013).

Several formulations for general project scheduling problems in the literature are based on a discrete representation of time. These formulations can be divided into three categories based on the type of decision variables used. The first category is based on binary variables x_{it} that equal 1 if the activity i starts or ends in period t (cf., e.g., Pritsker et al. 1969). In the second category of formulations, binary variables x_{it} equal 1 if activity i is processed in period t (cf., e.g., Kaplan 1988). In the third category, binary variables x_{it} equal 1 if activity i is in progress or has been processed before period t (cf., e.g., Klein 2000). In addition, there exist flow-based and event-based continuous-time formulations that comprise sequencing variables or event variables, respectively (cf., e.g., Koné et al. 2011; Chap. 2 of this book).

In this chapter, we present a formulation that is based on a discrete representation of time and on binary variables x_{it} that are equal to 1 if the activity i is processed in period t and are equal to 0, otherwise. In Sect. 24.4.1, we introduce the notation. In Sects. 24.4.2 and 24.4.3, we discuss the time-related and resource-related constraints of the formulation, respectively. In Sect. 24.4.4, we introduce the objective function and summarize the optimization problem.

24.4.1 Symbols

We use the following notation.

Indices

i	Activity
t	Time period
k	Resource
k^*	Work-content resource

Sets

V	Set of activities ($V = \{1, \ldots, n + 1\}$)
H	Set of periods ($H = \{1, \ldots, LC_{n+1} + 1\}$)
$\mathscr{R}$	Set of resources
V^a	Set of real activities ($V^a = \{1, \ldots, n\}$)
V^a_{kt}	Set of real activities that can be processed in period $t \in H$ and require resource $k \in \mathscr{R}$
H_i	Set of relevant periods for activity i ($H_i = \{ES_i + 1, \ldots, LC_i, LC_i + 1\}$)
$\mathscr{R}_i$	Set of resources required by activity i
$Pred(i)$	Set of immediate predecessors of activity i

Parameters

wc_i	Work content of activity i
r^{max}_{ik}	Upper bound on the amount of resource $k \in \mathscr{R}$ used by activity i
r^{min}_{ik}	Lower bound on the amount of resource $k \in \mathscr{R}$ used by activity i
ℓ	Minimum block length
s_{ik}	Increment of requirement for resource $k \in \mathscr{R} \setminus \{k^*\}$ by activity i per additional unit of work content
R_k	Capacity of resource $k \in \mathscr{R}$

Integer decision variable (non-negative)

r_{ikt} Amount of resource $k \in \mathscr{R}$ used by activity $i \in V_{kt}^a$ in period $t \in H_i$

Binary decision variables

$$x_{it} = \begin{cases} 1, \text{ if activity } i \text{ is processed in period } t \in H_i \\ 0, \text{ otherwise} \end{cases}$$

$$y_{it} = \begin{cases} 1, \text{ if the amount of resource } k^* \text{ used by activity } i \in V^a \text{ in period } t \in H_i \\ \quad \text{differs from period } t-1 \\ 0, \text{ otherwise} \end{cases}$$

24.4.2 Time-Related Constraints

For each activity $i \in V$, we determine the set of available periods H_i based on the earliest start and the latest finish times (ES_i, LC_i). The decision variables related to activity i are defined only for these periods. The earliest start times are computed by forward recursion. The earliest start time of an activity is thereby set to the latest of the earliest finish times of all of its immediate predecessors. The earliest finish time of an activity is computed by adding the lower bound on its duration $\lceil wc_i / r_{ik^*}^{max} \rceil$ to its earliest start time. The latest finish times are computed similarly using backward recursion. Thereby LC_{n+1} must be sufficiently large to guarantee feasibility. Constraint (24.1) ensures that the project completion, i.e., the dummy activity $n+1$, is scheduled within the planning horizon.

$$\sum_{t \in H_{n+1}} x_{(n+1)t} = 1 \tag{24.1}$$

Constraints (24.2) prevent activities from being interrupted. If activity i is executed in period $t-1$ and its work content has not been entirely processed by the end of this period, then activity i must also be processed in period t.

$$x_{i(t-1)} - \sum_{t' \in H_i :\, t' < t} \frac{r_{ik^*t'}}{wc_i} \leq x_{it} \quad (i \in V^a;\; t \in H_i \,:\, t > ES_i + 1) \tag{24.2}$$

The finish-to-start precedence relationships are formulated by constraints (24.3). Activity i can start only if the total work content of all of its predecessors has been processed.

$$x_{it} \leq \sum_{t' \in H_{i'} : t' < t} \frac{r_{i'k^*t'}}{wc_{i'}} \quad (i \in V;\; i' \in Pred(i);\; t \in H_i) \tag{24.3}$$

24.4.3 Resource-Related Constraints

Constraints (24.4) guarantee that the work content of each activity is processed exactly during its execution.

$$\sum_{t \in H_i} r_{ik^*t} = wc_i \quad (i \in V^a) \tag{24.4}$$

Constraints (24.5) and (24.6) impose lower and upper bounds on the usage of the work-content resource, respectively.

$$r_{ik^*}^{min} x_{it} \leq r_{ik^*t} \quad (i \in V^a;\ t \in H_i) \tag{24.5}$$

$$r_{ik^*}^{max} x_{it} \geq r_{ik^*t} \quad (i \in V^a;\ t \in H_i) \tag{24.6}$$

Constraints (24.7), (24.8), and (24.9) force the variable y_{it} to assume the value 1 if the usage of the work-content resource by activity i in period $t-1$ differs from its usage in period t. When an activity is completed in period $t = LC_i$, the variable y_{it} in the period $t = LC_i + 1$ will capture the last change in the usage of the work-content resource.

$$r_{ik^*t} \leq r_{ik^*}^{max} y_{it} \quad (i \in V^a;\ t = ES_i + 1) \tag{24.7}$$

$$r_{ik^*t} - r_{ik^*(t-1)} \leq r_{ik^*}^{max} y_{it} \quad (i \in V^a;\ t \in H_i : t > ES_i + 1) \tag{24.8}$$

$$r_{ik^*(t-1)} - r_{ik^*t} \leq r_{ik^*}^{max} y_{it} \quad (i \in V^a;\ t \in H_i : t > ES_i + 1) \tag{24.9}$$

Constraints (24.10) prevent an activity i from being processed after its latest finish time.

$$x_{it} = 0 \quad (i \in V^a;\ t = LC_i + 1) \tag{24.10}$$

The minimum block length is ensured through constraints (24.11), which allow only one change in the usage of the work-content resource during ℓ consecutive periods $t \in H_i$.

$$\sum_{t'=0}^{\ell-1} y_{i(t+t')} \leq 1 \quad (i \in V^a;\ t \in H_i : t \leq LC_i - (\ell - 2)) \tag{24.11}$$

The requirement for resource $k \in \mathscr{R} \setminus \{k^*\}$ by activity i in period t is computed by constraints (24.12). As this requirement must be an integer value, we use integer variables r_{ikt} to round up fractional values.

$$r_{ik}^{min} x_{it} + s_{ik}(r_{ik^*t} - r_{ik^*}^{min}) \leq r_{ikt} \quad (i \in V^a;\ k \in \mathscr{R}_i : k \neq k^*;\ t \in H_i) \tag{24.12}$$

Constraints (24.13) ensure that the total requirement for each resource $k \in \mathscr{R}$ does not exceed its capacity R_k.

$$\sum_{i \in V_{kt}^a} r_{ikt} \leq R_k \quad (k \in \mathscr{R};\ t \in H) \tag{24.13}$$

24.4.4 Objective Function

The objective is to minimize the duration of the project. The dummy activity $n+1$ represents the end of the project, as it can be scheduled only after the completion of all real activities $i \in V^a$. The optimization problem (P) reads as follows.

$$\text{Min.} \sum_{t \in H_{n+1}} t x_{(n+1)t} - 1 \tag{24.14}$$

$$\text{s. t.} \quad (24.1)\text{–}(24.13)$$

$$r_{ikt} \in \mathbb{Z}_{\geq 0} \quad (i \in V^a;\ k \in \mathscr{R}_i;\ t \in H_i) \tag{24.15}$$

$$y_{it} \in \{0, 1\} \quad (i \in V^a;\ t \in H_i) \tag{24.16}$$

$$x_{it} \in \{0, 1\} \quad (i \in V;\ t \in H_i) \tag{24.17}$$

24.5 Computational Results

In this section, we apply the proposed MILP model to a set of 480 problem instances introduced in Fündeling and Trautmann (2010). Each instance represents a project with ten activities, one work-content resource and up to three non-work-content resources. The instances were generated by systematically varying three complexity parameters. The parameter *order strength OS* defines the density of the project network and was chosen from the set $\{0.25, 0.5, 0.75\}$; the higher the value, the more dense the project network. The parameter *resource factor RF* defines the mean percentage of resources used by an activity and was chosen from the set $\{0.25, 0.5, 0.75, 1\}$. The parameter *resource strength RS* defines the degree of resource scarcity and was chosen from the set $\{0, 0.25, 0.5, 0.75\}$; the lower the value, the scarcer the resources. The test set contains ten instances for each parameter combination. The minimum block length was randomly chosen from the set $\{2, 3, 4\}$. The parameters wc_i, r_{ik}^{min} and r_{ik}^{max} were selected randomly such that a feasible solution exists for each instance. For further details, we refer to Fündeling and Trautmann (2010).

We have used the method of Fündeling and Trautmann (2010) to compute upper bounds on the makespan of these instances. To facilitate the computation

Table 24.1 Computational results of model IE

RS	0.00			0.25			0.50			0.75		
ℓ	2	3	4	2	3	4	2	3	4	2	3	4
# Instances	36	44	40	41	43	36	38	44	38	40	30	50
# Feasible	35	40	24	41	37	30	38	43	33	40	29	40
# Optimal	27	36	16	39	34	26	37	40	31	40	26	37
$\Delta^{\varnothing}_{MIP}$ [%]	8.9	7.3	6.0	2.1	2.1	4.7	3.4	2.4	1.6	na	1.8	2.8
Δ^{max}_{MIP} [%]	23.6	20.3	10.8	2.3	2.8	12.3	3.4	2.6	1.8	na	2.4	4.7
$\Delta^{\varnothing}_{LB}$ [%]	18.3	21.3	26.3	10.0	18.2	28.8	7.5	12.1	22.4	5.9	12.1	18.5
$\Delta^{\varnothing}_{UB}$ [%]	−2.9	−3.1	−2.7	−3.4	−3.4	−2.6	−2.5	−2.8	−2.9	−2.0	−2.3	−1.6
Δ^{min}_{UB} [%]	−9.8	−11.8	−9.6	−9.5	−16.3	−10.0	−11.1	−10.9	−12.9	−10.7	−9.1	−10.8

of a feasible solution, we have set the latest finish time of the dummy activities $n+1$ to the respective upper bound multiplied by a factor of 1.05. We have implemented the proposed MILP formulation in AMPL and used Gurobi 5.5 as a solver on a standard workstation with two Intel Xeon 3.1 GHz CPUs and 128 GB RAM. First we investigate the problem setting where the number of resource units allocated to an activity must be an integer, and the exact work content of each activity is to be processed (setting IE). Due to the complexity of problem (P), some instances require large CPU times. Therefore, we prescribed a time limit of $t^{lim}_{cpu} = 600$ s per instance. A feasible solution was found for 430 instances, and solution optimality was proven for 389 instances. Table 24.1 lists the results for different combinations of the parameters RS and ℓ. These two parameters appear to drive the CPU time requirement of this test set. In rows 3 to 5 of Table 24.1, we report the number of existing instances with the respective combination of RS and ℓ, the number of instances for which a feasible solution was found within the time limit, and the number of instances for which optimality could be proven within the time limit, respectively. In rows 6 and 7, we report the average and the maximum MIP gap of the instances for which a feasible solution was found but for which optimality could not be proven within the time limit. In row 8, we give the average relative deviation of our solutions to the lower bounds computed in Fündeling (2006). In rows 9 and 10, we list the average and the minimum relative deviations (i.e., the negative of the maximum relative improvements) of our solutions from the solutions found by the multi-start priority-rule method of Fündeling (2006).

We conclude that the scarcer the resources or the longer the minimum block length, the more CPU time is required to find a feasible solution and to prove optimality. The latter observation may be explained by the fact that the number of feasible resource profiles for an activity decreases with increasing minimum block length. If a feasible solution can be found within the time limit, the MIP gap is generally rather small. Our schedules can be up to 16.3 % shorter than the schedules obtained by the priority-rule method of Fündeling and Trautmann (2010). However, the method of Fündeling and Trautmann (2010) provides a feasible solution to all 480 instances.

Table 24.2 Comparison of different types of model relaxations

Model	IE	IG	CE	CG
# Instances	480	480	480	480
# Feasible	430	480	480	480
# Optimal	389	475	476	475
$\Delta_{MIP}^{\varnothing}$ [%]	5.0	4.9	1.6	1.6
Δ_{MIP}^{max} [%]	23.6	11.5	2.4	2.4
$\Delta_{LB}^{\varnothing}$ [%]	16.2	7.1	7.0	6.7

For the sake of completeness, we summarize the results obtained for the respective MILP models for the following relaxations of this problem setting:

- setting IG: integral resource allocation, work content processed greater than or equal to prescribed work content
- setting CE: continuous resource allocation, work content processed equal to prescribed work content
- setting CG: continuous resource allocation, work content processed greater than or equal to prescribed work content

The results displayed in Table 24.2 indicate that any of these relaxations reduces the problem complexity considerably; in particular, all instances are solved to feasibility, and almost all instances are solved to optimality.

24.6 Conclusions

For the work-content-constrained project scheduling problem, heuristic solution methods and a specific branch-and-bound procedure have been proposed in the literature. In this chapter, we reviewed a priority-rule heuristic, and we presented a novel MILP formulation. The MILP formulation covers in particular the minimum block-length constraint and considers the dependent requirement for non-work-content resources. We performed an experimental performance analysis with a set of 480 test instances previously introduced by Fündeling and Trautmann (2010). For 430 instances, the model found an optimal or near-optimal solution within a short CPU time; in these 430 instances, the proposed model outperforms the state-of-the-art method.

The model presented in this chapter contributes to the development of efficient MILP formulations of resource-allocation problems that can then be solved with standard software. Our model will help to formulate exact models for related applications such as those discussed, e.g., in Kolisch et al. (2003). The possible extensions of this model include work-content-based precedence relationships, i.e., only a prescribed percentage of the work content of an activity must be completed before the succeeding activity can be started, as well as time-varying resource capacities. The model could easily be modified to consider these extensions by adapting constraints (24.3) and (24.13). Moreover, to further evaluate the performance of the proposed model, we plan to develop MILP formulations that are based on different types of decision variables.

References

Baumann P, Trautmann N (2013) Optimal scheduling of work-content-constrained projects. In: Laosirihongthong T, Jiao R, Xie M, Sirovetnukul R (eds) Proceedings of the international conference on industrial engineering and engineering management. IEEE, Bangkok

Bianco L, Caramia M (2013) A new formulation for the project scheduling problem under limited resources. Flex Serv Manuf J 25:6–24

Fündeling C (2006) Ressourcenbeschränkte Projektplanung bei vorgegebenen Arbeitsvolumina. Gabler, Wiesbaden

Fündeling C, Trautmann N (2010) A priority-rule method for project scheduling with work-content constraints. Eur J Oper Res 203:568–574

Garey M, Johnson D (1979) Computers and intractability: a guide to the theory of NP-completeness. Freeman, New York

Kaplan LA (1988) Resource constrained project scheduling with preemption of jobs. Ph.D. dissertation, University of Michigan, MI, USA

Klein R (2000) Scheduling of resource-constrained projects. Kluwer, Amsterdam

Kolisch R, Meyer K, Mohr R, Schwindt C, Urmann M (2003) Ablaufplanung für die Leitstrukturoptimierung in der Pharmaforschung. Z Betriebswirt 73:825–848

Koné O, Artigues C, Lopez P, Mongeau M (2011) Event-based MILP models for resource-constrained project scheduling problems. Comp Oper Res 38:3–13

Naber A, Kolisch R (2013) MIP models for resource-constrained project scheduling with flexible resource profiles. Eur J Oper Res 239:335–348

Pritsker A, Watters L, Wolfe P (1969) Multiproject scheduling with limited resources: a zero-one programming approach. Manag Sci 16:93–107

Rieck J, Zimmermann J, Gather T (2012) Mixed-integer linear programming for resource leveling problems. Eur J Oper Res 221:27–37

Węglarz J, Józefowska J, Mika M, Waligóra G (2011) Project scheduling with finite or infinite number of activity processing modes: a survey. Eur J Oper Res 208:177–205

Part VIII
Project Staffing and Scheduling Problems

Chapter 25
A Modeling Framework for Project Staffing and Scheduling Problems

Isabel Correia and Francisco Saldanha-da-Gama

Abstract This chapter addresses modeling issues associated with project staffing and scheduling problems. Emphasis is given to mixed-integer linear programming formulations. The need for such formulations is motivated and a general modeling framework is introduced, which captures many features that have been considered in the literature on project staffing and scheduling problems. The use of the general framework is then exemplified using two problems that have been addressed in the literature. Several model enhancements and preprocessing procedures are discussed.

Keywords Additional inequalities • MILP models • Project scheduling • Project staffing

25.1 Introduction

Project staffing and scheduling problems lead, in general, to very complex optimization problems as they often extend a resource-constrained project scheduling problem. Accordingly, it is important to develop methodologies which can efficiently solve such problems. Large instances of the problems can often be expected to be solved only approximately by using heuristics. Nevertheless, this does not mean that exact approaches should not be attempted, firstly, because this is a mean for evaluating (for small or medium size instances) new heuristic procedures and secondly, because special classes of large instances may be solved to optimality in an efficient way.

I. Correia (✉)
Departamento de Matemática/Centro de Matemática e Aplicações, Universidade Nova de Lisboa, Caparica, Portugal
e-mail: isc@fct.unl.pt

F. Saldanha-da-Gama
Departamento de Estatística e Investigação Operacional/Centro de Investigação Operacional, Universidade de Lisboa, Lisboa, Portugal
e-mail: fsgama@fc.ul.pt

C. Schwindt, J. Zimmermann (eds.), *Handbook on Project Management and Scheduling Vol. 1*, International Handbooks on Information Systems,
DOI 10.1007/978-3-319-05443-8_25

Many project staffing and scheduling problems are, in fact, optimization problems. Hence, one possibility for developing exact approaches is to consider mathematical programming based approaches. Such is only possible if the problems are firstly formulated as the maximization or minimization of a function of several variables subject to a set of constraints in these variables. The nature of the decisions involved in such problems leads often to mixed-integer linear programming (MILP) formulations. This is the focus of this chapter.

Once a mixed-integer linear programming formulation is available for a problem several approaches may be attempted. These include (1) the direct use of the formulation in an off-the-shelf solver, (2) branch-and-bound approaches, and (3) decomposition approaches.

In many resource-constrained project scheduling problems, resources are flexible or, as often called in the literature, multi-skilled. This means that each resource can perform at least one skill. This is often the case when human resources are involved. Note, however, that multi-skilled resources can also refer to machines or other resources not necessarily human.

When resources are multi-skilled we talk of completely skilled resources when all the resources have all the skills. We say that we have resources with heterogeneous efficiencies if the amount of work requiring a specific skill that can be performed in one time unit varies from one resource to another. Otherwise we have homogeneous efficiencies.

When resources are multi-skilled, not only it becomes necessary to decide which resources will be assigned to each activity but also the skills with which they contribute to each activity. Accordingly, in terms of the decisions to be made, project staffing and scheduling problems add one dimension to resource-constrained project scheduling problems, which, in turn, represent already a significant increase of complexity when compared to the basic project scheduling problems.

The main purpose of this chapter is to revisit and discuss different models that have been proposed in the literature for project staffing and scheduling problems namely those proposed by Li and Womer (2009) and Correia et al. (2012). In particular, a general modeling framework is first presented, which generalizes those models. We focus only on project scheduling problems with multi-skilled resources. Hence, we do not consider the work that has been done on single-skilled problems. Furthermore, we consider a single project setting, i.e., we assume that we have a project which is defined by a set of activities such that for several pairs of activities some time dependency exists between the corresponding starting times. As mentioned above, the focus will be put on modeling issues namely on the possibility of using a MILP model for such type of problem.

In this chapter we avoid being too specific, thus making the modeling framework proposed easily adjustable to more complex situations (as, for instance, the problem arising in an industrial context, which is addressed in Chap. 28). Furthermore, the information provided by this chapter should be complemented with further reading namely that provided by Chaps. 26 and 27, where techniques based on column generation, Lagrangian relaxation and Benders decomposition are proposed.

The idea of considering mathematical programming formulations for complex scheduling problems is far from new. Some examples are the well-known mixed-integer linear programming formulations proposed by Applegate and Cook (1991) for the job-shop scheduling problem, and the formulations proposed by Alvarez-Valdes and Tamarit (1993), Baptiste and Demassey (2004), Carlier and Néron (2003), Christofides et al. (1987), and Demassey (2008) for the resource-constrained project scheduling problem.

The use of mathematical programming formulations for project staffing and scheduling problems has also been attempted. This is the case with the integer programming time-dependant formulations proposed by Bellenguez and Néron (2005) for a multi-skill resource-constrained project scheduling problem. Li and Womer (2009) and Correia et al. (2012) also explore the possibility of using mixed-integer linear programming formulations for this type of problems. In the first work, a hybrid Benders decomposition algorithm is proposed which combines the complementary strengths of both mixed-integer linear programming and constraint programming. The second work explores the possibility of strengthening a MILP formulation using valid inequalities as a way of making the use of a general solver more efficient.

Finally, we would like to cite the work by Heimerl and Kolisch (2010), who proposed a MILP model for a quite general project staffing and scheduling problem in the context of multiple projects. The model is enhanced with the goal of using a general solver.

The remainder of this chapter is organized as follows: In Sect. 25.2 we discuss several important ingredients that often have to be gathered before starting developing a MILP formulation for a project staffing and scheduling problem. In the following section, a general modeling framework is proposed, which captures many features that have been considered in the literature. In Sects. 25.4 and 25.5, the utility of the general framework is illustrated by using two particular problems that are addressed in the literature. Finally, in Sect. 25.6 we discuss some extensions. The chapter ends with some conclusions and ideas for modeling developments in the context of the problems addressed.

25.2 Ingredients for a General Modeling Framework

Before going deeper in terms of modeling considerations, it is important to understand what is involved in the basic setting for a project staffing and scheduling problem. Typically, we have a set of activities to be processed such that for several pairs of activities, some time dependency exists between their starting times. In order to be processed, each activity requires the use of resources, which are limited. In project staffing and scheduling problems, such resources are people each of which having various skills. In order to process each activity, more than one skill may be necessary. In particular, a pre-determined number of resources for each skill is needed. The goal is typically to decide the best way of assigning the resources to the activities so as to optimize some predefined performance measure.

The final step before developing a modeling framework is to understand what can be involved in terms of parameters, decisions to be made, and performance measures to be optimized. Regarding the parameters, they may include information concerning the activities, the resources, and the project as a whole.

As far as the activities are concerned, some information must be gathered:

- *Processing times*. For each activity, some processing time is considered, which is assumed to be defined in advance.
- *Minimum delays*. When some activity can start only some pre-specified time after the beginning of some other, we say that we have a minimum delay between the two activities. For instance, when some activity must precede another one, the latter can only start after the former has been completed, i.e., there is a minimum delay equal to the processing time of the first activity.
- *Deadlines*. A deadline exists for some activity when the completion time of the activity cannot exceed some pre-specified moment in time. A deadline sets a hard constraint on the time for finishing the execution of the activity.
- *Due dates*. A due date is a softer version of a deadline in the sense that it sets some desirable completion time for an activity, which, however, may not be respected. The existence of due dates is often associated with some performance measure (e.g., lateness).
- *Resource requirements*. Each activity may require resources with different skills. Thus, for each activity, it is important to identify in advance the skills that will be required by the activity. Furthermore, the amount of resources required for each skill is a crucial complementary information.

As far as the resources are concerned, one can identify the following crucial information:

- *Workload*. This stands for the total time that each resource (individual) can work in the project.
- *Resource potential and availability*. In a multi-skill problem, each resource may have more than one skill. Accordingly, it is important to identify the skills that each individual can perform.
- *Costs*. The use of each resource may, for instance, imply a fixed cost which is incurred by using the resource in the project.

Finally, as a complement to all the previous information, one may have to consider information/data for the project as whole. A maximum prescribed duration (i.e., a deadline) is possibly the most commonly used. When this is the case, such information must be included in the model, leading to a multi-skill resource-constrained project scheduling problem with a given deadline.

The next step towards the development of a model is the identification of the decision variables. These can be classified into two groups: one contains the variables, which all together implicitly represent a solution to the problem. In other words, these are variables whose values are enough to completely define a solution to the problem. The second set of decision variables contains what can be called auxiliary variables. These are variables which are not needed for defining a solution but that are important for modeling purposes either because they help describing the constraints of the problem or because they are fundamental for accounting for the costs.

We detail the sets of decision variables which are typically involved in a modeling framework for a project staffing and scheduling problem:

- *Starting time for the activities*. These are decision variables considered in every project scheduling problem, which convey the major decision to be made.
- *Resource allocation*. These variables define how the resources will be assigned to the different activities. In particular, when multi-skilled resources are involved, these variables also specify the skills with which each resource contributes to each activity.
- *Sequencing*. These are auxiliary variables which help to model the resource constraints. The issue arises because it is often the case that there are pairs of activities such that no precedence constraint or minimum delay exists between them. In such situation, the activities may be processed simultaneously or following some sequence to be decided. This will be decided, among other things, by the resource availability. In any case, it is important to consider decision variables which state the sequence (if any) by which the activities will be processed.
- *Resources usage*. Such variables are important when a fixed cost is associated to each resource. In such a case, these variables help accounting for the costs.

Finally, the performance measure to be optimized in a project staffing and scheduling problem is a key information for a complete definition of the problem. Typically, two performance measures that have been considered in project staffing and scheduling problems are the time necessary to finish the entire project (i.e., the makespan) and the total cost associated with the use of resources. Both are to be minimized.

25.3 A General Modeling Framework

In the basic setting of the project staffing and scheduling problems underlying the (MILP) models that will be presented in this chapter, the following is assumed:

- Preemption is not allowed. This means that once an activity starts to be executed it can not be interrupted.
- All processing times are integer. In particular it is assumed that the time necessary to execute an activity can be measured in hours, days, weeks, months, etc. Thus fractional values are not accepted.
- The resource requirements for each activity are known in advance and do not change over the processing of the activity.
- In each point in time, each resource can be involved in at most one activity and in this case, with a single skill.

- Each activity must be completed before its deadline.
- The resources have homogeneous efficiencies.
- All data of the problem is known in advance and is not subject to any type of uncertainty.

Taking into account all aspects discussed above, the following notation is now considered:
Sets:

$\mathscr{R} = \{1, \ldots, k, \ldots, K\}$	Set of resources.
$V = \{1, \ldots, n\}$	Set of activities to be executed.
$\mathscr{L} = \{1, \ldots, l, \ldots, L\}$	Set of skills.
$\mathscr{L}_i$	Set of skills required by activity $i \in V$.
$\mathscr{L}_k$	Set of skills that can be performed by resource $k \in \mathscr{R}$.

Based upon the previous notation we can define:

$\mathscr{R}_l = \{k \in \mathscr{R} \mid l \in \mathscr{L}_k\}$	Set of resources with skill $l \in \mathscr{L}$.
$V_k = \{i \in V \mid \mathscr{L}_i \cap \mathscr{L}_k \neq \emptyset\}$	Set of activities requiring skills that resource $k \in \mathscr{R}$ has.
$\mathscr{R}_i = \{k \in \mathscr{R} \mid \mathscr{L}_i \cap \mathscr{L}_k \neq \emptyset\}$	Set of resources with skills required by activity $i \in V$.

Parameters:

r_{il}	Number of resource units (workers) with skill $l \in \mathscr{L}$ required by activity $i \in V$.
p_i	Processing time for activity $i \in V$.
d_{ij}^{min}	Minimum time lag between the start of activities i and j $(i, j \in V)$.
d_{ji}^{max}	Maximum time lag between the start of activities j and i $(i, j \in V)$.
δ_{ij}	Weight of arc (i, j), chosen in the set $\{d_{ij}^{min}, -d_{ji}^{max}, p_i, -\infty\}$.
$\overline{d}_i$	Deadline for the completion of activity $i \in V$.
$\overline{d}$	Deadline for the entire project.
WL_k	Workload capacity of resource $k \in \mathscr{R}$.
c_k	Cost of using resource $k \in \mathscr{R}$ in the project.
UB	Upper bound on the project makespan.

Decision Variables: Many mixed-integer linear programming models for project scheduling problems consider non-negative decision variables for the starting time of the activities as well as binary variables defining the execution order for any pair of activities. The same will also be done here. Accordingly, we have

S_i : Starting time for activity $i \in V$.

Regarding the resource allocation we consider

$$x_{ikl} = \begin{cases} 1 \text{ if resource } k \text{ contributes with skill } l \\ \quad \text{for activity } i \\ 0 \text{ otherwise} \end{cases} \qquad (i \in V;\ k \in \mathscr{R}_i;\ l \in \mathscr{L}_k \cap \mathscr{L}_i)$$

In addition to the decision variables above the following two sets of auxiliary variables are also introduced as they are crucial for the modeling framework to be presented:

$$y_{ij} = \begin{cases} 1 \text{ if activity } i \text{ is completed before activity } j \text{ is started} \\ 0 \text{ otherwise} \end{cases} \quad (i, j \in V)$$

$$z_k = \begin{cases} 1 \text{ if resource } k \text{ is selected to perform the project} \\ 0 \text{ otherwise} \end{cases} \quad (k \in \mathscr{R})$$

Under the assumptions made above, a generic modeling framework for a project staffing and scheduling problem, which gathers the models proposed by Li and Womer (2009) and Correia et al. (2012), is the following:

$$(P) \quad \text{Min. } f(S, x, y, z) \tag{25.1}$$

$$\text{s.t.} \quad S_j \geq S_i + \delta_{ij} \quad (i, j \in V) \tag{25.2}$$

$$S_j \geq S_i + p_i - M\left(1 - y_{ij}\right) \quad (i, j \in V) \tag{25.3}$$

$$\sum_{k \in \mathscr{R}_l} x_{ikl} = r_{il} \quad (i \in V;\ l \in \mathscr{L}_i) \tag{25.4}$$

$$\sum_{l \in \mathscr{L}_i \cap \mathscr{L}_k} x_{ikl} \leq 1 \quad (k \in \mathscr{R};\ i \in V_k) \tag{25.5}$$

$$\sum_{l \in \mathscr{L}_k \cap \mathscr{L}_i} x_{ikl} + \sum_{l \in \mathscr{L}_k \cap \mathscr{L}_j} x_{jkl} \leq y_{ij} + y_{ji} + 1 \quad (k \in \mathscr{R};\ i, j \in V_k) \tag{25.6}$$

$$S_i + p_i \leq \min\{\overline{d}_i, \overline{d}\} \quad (i \in V) \tag{25.7}$$

$$\sum_{i \in V_k} \sum_{l \in \mathscr{L}_k \cap \mathscr{L}_i} p_i x_{ikl} \leq WL_k z_k \quad (k \in \mathscr{R}) \tag{25.8}$$

$$S_i \geq 0 \quad (i \in V) \tag{25.9}$$

$$x_{ikl} \in \{0, 1\} \quad (i \in V;\ k \in \mathscr{R}_i;\ l \in \mathscr{L}_k \cap \mathscr{L}_i) \tag{25.10}$$

$$y_{ij} \in \{0, 1\} \quad (i, j \in V) \tag{25.11}$$

$$z_k \in \{0, 1\} \quad (k \in \mathscr{R}) \tag{25.12}$$

The generic function in (25.1) represents the objective function to be minimized. Constraints (25.2) are the time lag constraints. In constraints (25.3) M denotes a large value. These constraints define the values for the sequencing variables assuring that if y_{ij} is equal to 1, then activity j can start only after activity i is finished. Constraints (25.4) assure that for each skill required for each activity, the necessary resources are allocated. Inequalities (25.5) state that each resource contributes with at most one skill to each activity (to which it can be assigned). Constraints (25.6) assure that if one resource k is assigned to two activities i and j (assuming $i, j \in V_k$), then $y_{ij} + y_{ji} \geq 1$, i.e., either i is performed before j or j is performed before i. Constraints (25.7) assure that the completion time of the activities respects the deadlines. Constraints (25.8) assure that the workload capacity of the resources is satisfied. Constraints (25.9)–(25.12) are domain constraints.

Remark 25.1.

1. If $\delta_{ij} = p_i$ for two activities $i, j \in V$, then constraints (25.2) reduce to the traditional precedence constraints. If $\delta_{ij} = -\infty$, then no relation exists between the starting time of activities i and j. Finally, as stressed by Li and Womer (2009), if $\delta_{ij} \geq 0$ the constraint represents a minimum time lag between activities i and j; if $\delta_{ij} < 0$ a maximum time lag is being assured between activity j and activity i.
2. The previous formulation assures implicitly that $y_{ij} + y_{ji} \leq 1$, $i, j \in V$. In fact, suppose that for two activities i and j we had $y_{ij} + y_{ji} > 1$. The only possibility would be to have $y_{ij} = 1$ and $y_{ji} = 1$. In this case, by (25.3) we would have $S_j \geq S_i + p_i$ and $S_i \geq S_j + p_j$. Accordingly, we could write $S_j \geq S_i + p_i \geq S_j + p_j + p_i$. As the processing times are positive we would obtain $S_j > S_j$, which makes no sense. Hence, there is no need to introduce consistency constraints for the choice of the values for the variables y.
3. It is in general assumed that 0 is the origin of time.
4. The formulation presented above accommodates very easily the imposition of a sequence between two activities. This is done simply by setting the value for two y variables. In fact, if we realize that some activity i must be processed before some other activity j, then we can simply set $y_{ij} = 1$ and $y_{ji} = 0$.

25.3.1 Preprocessing and Model Enhancement

In practice, a model such as P may be too large. Accordingly, one important aspect arising when facing such a model is the possibility of somehow simplify it. One way for achieving this is to perform a preprocessing, which can lead to fixing the values of some (ideally as much as possible) binary variables. Even when it is not possible to directly fix the values of some binary variables, it may be possible to add additional conditions for enhancing the model. These aspects may be relevant when it comes to use an off-the-shelf general solver for tackling a model such as P.

One obvious simplification is the following:

If for some resource $k \in \mathscr{R}$ and for some activity $i \in V_k$ we have $WL_k < p_i$, then it must be

$$x_{ikl} = 0 \quad (l \in \mathscr{L}_i \cap \mathscr{L}_k) \tag{25.13}$$

This, in turn, leads to a reduction on the cardinality of the sets V_k and $\mathscr{R}_i$:

$$V_k := V_k \setminus \{i\} \qquad \mathscr{R}_i := \mathscr{R}_i \setminus \{k\} \tag{25.14}$$

In what follows, we assume that this reduction has been performed.

The following definition is crucial for the discussion to be presented below.

Definition 25.1. Two activities $i, j \in V$ are said to be incompatible if there is never a slot in time in which they can be executed simultaneously.

The previous incompatibility concept is far from new. In fact, it has been proposed for the resource-constrained project scheduling problem (see, for instance, Alvarez-Valdes and Tamarit 1993; Klein and Scholl 1999).

By considering the availability of the resources as well as the skills and the corresponding amount of resources required by each activity it is not difficult to state sufficient conditions for incompatibility. Before doing so, consider two activities i and j and an auxiliary directed network $N' = (V', E')$ (V' is the set of nodes; E' is the set of arcs) built as follows:

- Apart from a source and a sink node two intermediate sets of nodes are considered in V'.
  - Each node in the first intermediate set is associated with a resource which can be allocated to at least one of the activities i and j. Hence, the first intermediate set of nodes is defined by the resources in $\mathscr{R}_i \cup \mathscr{R}_j$.
  - The second intermediate set of nodes is partitioned into two subsets. In the first subset, each node is associated with a skill in $\mathscr{L}_i$; in the second subset, each node is associated with a skill in $\mathscr{L}_j$.
- The set of arcs E' is defined as follows.
  - From the source node there will be an arc converging in each node in the first intermediate set with a minimum throughput 0 and capacity 1.
  - Each node (resource) in the first intermediate set will be linked to a node (skill) in the first subset of the second intermediate set if the resource can be allocated to activity i with that skill. Each node (resource) in the first intermediate set will be linked to a node (skill) in the second subset of the second intermediate set if the resource can be allocated to activity j with that skill. In every case, the minimum throughput associated with an arc will be set to 0 and the capacity to 1.
  - From every node in the second intermediate set there will be an arc directed to the sink node. Each such arc has a minimum throughput and capacity equal to the number of resources required for the corresponding skill in the corresponding activity i or j.

If a feasible flow exists in this auxiliary graph, then there are enough resources to execute i and j simultaneously.

Two immediate sufficient conditions for two activities i and j to be incompatible are the following:

C1 $\delta_{ij} \geq p_i$.
C2 No feasible flow exists in the auxiliary network $N' = (V', E')$ built as shown above.

The concept of incompatibility can be straightforwardly extended to a triplet of activities $\{i, j, h\}$. In this case, it is also straightforward to extend the sufficient conditions C1 and C2.

The knowledge of the sets of incompatible activities can be easily introduced in model P. This was first proposed by Alvarez-Valdes and Tamarit (1993) for the project scheduling problem and can be done here in exactly the same way. In fact, as mentioned before, if two activities i, j are incompatible, then either i will be executed before j or the other way round. Therefore, we can impose

$$y_{ij} + y_{ji} = 1 \tag{25.15}$$

because in this case we know in advance that it is not possible to have $y_{ij} = y_{ji} = 0$.

If $\{i, j, h\}$ is a triplet of incompatible activities, then the following inequality is valid for P:

$$y_{ij} + y_{ji} + y_{ih} + y_{hi} + y_{jh} + y_{hj} \geq 1 \tag{25.16}$$

Note that the inequality above is only of interest if $\{i, j\}$, $\{i, h\}$ and $\{j, h\}$ are not incompatible otherwise the inequality is implied by (25.15) (using the appropriate indices).

This inequality follows from the fact that if the three activities i, j, h are incompatible, then at least one of the variables involved in the previous inequality should be equal to 1. Again, note that this type of inequality was proposed for project scheduling problems by Alvarez-Valdes and Tamarit (1993).

As pointed out by Correia et al. (2012) the reasoning behind inequality (25.16) could be used to derive inequalities for sets with more than three incompatible activities. However, in this process, a tradeoff should be reached between the gain we get with the inequality and the computational effort required. In fact, finding sets of four incompatible variables leads to a significant increase in the computational effort.

In addition to the previous sets of inequalities, a new set of inequalities can be derived by making use of the binary variables associated with the resources. Suppose that for each resource a value WL'_k is found, which represents the maximum total time that the resource can spend in the overall project. A natural choice for WL'_k is WL_k. However, when an upper bound (UB) on the project makespan is known we can possibly do better. In fact, if for some $k \in \mathscr{R}$, $UB < WL_k$, then we know that in an optimal solution, the total time that resource k is allocated to the project will not exceed UB. Hence, we define, for each $k \in \mathscr{R}$, $WL'_k = \min\{WL_k, UB\}$. Therefore, we can write

$$\sum_{i\in V_k}\sum_{l\in\mathcal{L}_k\cap\mathcal{L}_i} p_i x_{ikl} \leq WL'_k \quad (k\in\mathcal{R}) \tag{25.17}$$

From the previous inequalities and taking into account that we have binary variables in the left term, we can divide both members in (25.17) by an integer $q \in \{2,\ldots,\max_{i\in V_k}\{p_i\}\}$ obtaining

$$\sum_{i\in V_k}\sum_{l\in\mathcal{L}_k\cap\mathcal{L}_i} \frac{p_i}{q} x_{ikl} \leq \frac{WL'_k}{q} \quad (k\in\mathcal{R};\ q\in\{2,\ldots,\max_{i\in V_k}\{p_i\}\})$$

By rounding down each coefficient in the left-hand side in the previous inequalities and then by doing the same in the right-hand side we obtain a new set of additional inequalities that can be added to model P:

$$\sum_{i\in V_k}\sum_{l\in\mathcal{L}_k\cap\mathcal{L}_i} \left\lfloor\frac{p_i}{q}\right\rfloor x_{ikl} \leq \left\lfloor\frac{WL'_k}{q}\right\rfloor \quad (k\in\mathcal{R}; q\in\{2,\ldots,\max_{i\in V_k}\{p_i\}\}) \tag{25.18}$$

It should be noted that in the previous set of inequalities it may happen that several inequalities have the same right-hand side. In such case, only the 'stronger' should be added to model P.

One can strengthen the previous inequalities as follows:

$$\sum_{i\in V_k}\sum_{l\in\mathcal{L}_k\cap\mathcal{L}_i} \left\lfloor\frac{p_i}{q}\right\rfloor x_{ikl} \leq \left\lfloor\frac{WL'_k}{q}\right\rfloor z_k \quad (k\in\mathcal{R};\ q\in\{2,\ldots,\max_{i\in V_k}\{p_i\}\}) \tag{25.19}$$

25.4 A Makespan Minimization Multi-Skill Resource-Constrained Project Staffing and Scheduling Problem

In order to make more clear the utility of the generic model presented in the previous section we particularize it according to what has been presented by Correia et al. (2012). In the problem addressed by these authors, one finds the following particularizations:

- Two dummy activities 0 and $n+1$ are included in set V. This activities are assumed to have processing time equal to 0 and no resource requirements.
- The objective function is the makespan, i.e., $f(S,x,y,z) = S_{n+1}$.
- $\overline{d}_i = \sum_{i\in V} p_i, i\in V$.
- $\overline{d} = \sum_{i\in V} p_i$.
- $WL_k = \sum_{i\in V} p_i, k\in\mathcal{R}$.
- $\delta_{ij}\in\{p_i,-\infty\}, i,j\in V$.

The authors assume that the project is represented by an activity-on-node network $N = (V, E, \delta)$, where V represents the set of activities, E represents the precedences between such activities, and δ represents the set of arc weights. The network is acyclic and the nodes are labeled numerically in such a way that each activity has a label smaller than the label of every of its successors. The weight of an arc is equal to the processing time of the activity corresponding to the initial node of the arc. The dummy activity 0 represents the beginning of the project whereas $n+1$ represents the conclusion.

Precedence relations defined in the set E above imply other (non-direct) precedences. In fact, if some activity i_1 precedes i_2 and i_2 precedes i_3, then i_1 precedes i_3.

The following additional set (two-element antichains) can now be defined:

$$\mathscr{A}_2 = \{(i,j) \in V \times V \mid \text{no direct or transitive relation exists between } i \text{ and } j\}$$

This set contains the pairs of activities which can be processed simultaneously with respect to the temporal and resource constraints (feasible anti-chain). Initially, in this set, we only include the pairs of activities such that no precedence relation exists between them. Afterwards, using some preprocessing it will be possible to remove some pairs from that set due to incompatibilities arising from the resource constraints.

The generic model presented before reduces now to:

$$(P1) \quad \text{Min. } S_{n+1} \tag{25.20}$$

$$\text{s.t.} \quad S_j \geq S_i + p_i \quad ((i,j) \in E) \tag{25.21}$$

$$S_j \geq S_i + p_i - M\left(1 - y_{ij}\right) \quad ((i,j) \in \mathscr{A}_2) \tag{25.22}$$

$$\sum_{k \in \mathscr{R}_l} x_{ikl} = r_{il} \quad (i \in V;\ l \in \mathscr{L}_i) \tag{25.23}$$

$$\sum_{l \in \mathscr{L}_i \cap \mathscr{L}_k} x_{ikl} \leq 1 \quad (k \in \mathscr{R};\ i \in V_k) \tag{25.24}$$

$$\sum_{l \in \mathscr{L}_k \cap \mathscr{L}_i} x_{ikl} + \sum_{l \in \mathscr{L}_k \cap \mathscr{L}_j} x_{jkl} \leq y_{ij} + y_{ji} + 1$$

$$(k \in \mathscr{R};\ i,j \in V_k; (i,j) \in \mathscr{A}_2) \tag{25.25}$$

$$S_i \geq 0 \quad (i \in V) \tag{25.26}$$

$$x_{ikl} \in \{0,1\} \quad (i \in V;\ k \in \mathscr{R}_i;\ l \in \mathscr{L}_k \cap \mathscr{L}_i) \tag{25.27}$$

$$y_{ij} \in \{0,1\} \quad ((i,j) \in \mathscr{A}_2) \tag{25.28}$$

25.4.1 Feasibility Issues

The particular problem just revisited is $\mathscr{NP}$-hard as pointed out by Correia et al. (2012). Nevertheless, checking the feasibility of an instance is quite simple in this case.

To see this, one just need to observe that the feasibility of an instance of this problem only depends on having enough resources for processing each one of the activities. In fact, as no deadline exists for finishing the project, the activities can be processed one at a time. Therefore, checking the feasibility of the problem reduces to checking whether or not there are resources enough to execute each activity. As pointed out by Correia et al. (2012), at a first glance, one might think that this could be achieved by simply comparing, for each activity i, the values r_{il} with the cardinalities of the set $\mathscr{R}_l$. However, it is easy to find trivial examples showing that this is not the case.

Alternatively, feasibility can be checked by finding a feasible flow in a specific network defined for each activity. Considering a particular activity $i \in V$ define a directed graph $N_i = (V_i, E_i)$ in which:

- The set of nodes V_i contains:
  - a source node s,
  - $\mathscr{R}_i$ (a set of nodes defined by the resources that can be assigned to activity i),
  - $\mathscr{L}_i$ (a set of nodes associated with the skills required by activity i),
  - a sink node t.
- The set of arcs E_i contains:
  - arcs (s, k), $k \in \mathscr{R}_i$ with minimum throughput 0 and capacity 1,
  - arcs (k, l), $k \in \mathscr{R}_i$, $l \in \mathscr{L}_i \cap \mathscr{L}_k$, with minimum throughput 0 and capacity 1,
  - arcs (l, t), $l \in \mathscr{L}_i \cap \mathscr{L}_k$, with minimum throughput and capacity r_{il}.

Note that in order to be able to execute an activity $i \in V$, r_{il} resources are needed for each skill l required by the activity (i.e., for each skill $l \in \mathscr{L}_i$). However, each resource is used at most once because the lower and upper capacities in the intermediate arcs are 0 and 1, respectively.

If a feasible flow between s and t can be found in the above graph, then there are enough resources to execute activity i. If this happens for all activities, then the problem is feasible. If for some activity i there is no feasible flow between s and t in the corresponding graph, then the resources are not enough to execute this activity and consequently, the entire project fails to be executed.

25.4.2 Additional Inequalities

From an integer programming point of view, an important aspect worth investigating regarding a model such as $P1$ is the strength of the bound provided by its linear relaxation. Correia et al. (2012) prove that such bound is in fact quite easy to obtain as it is equal to the length of the longest path between nodes 0 and $n+1$ in the network N. Unfortunately, this bound can be extremely weak. For this reason, they consider the use of several sets of valid inequalities that even when not able to improve this bound may, at least, speed up a general solver when tackling the model $P1$. Some of these inequalities were already introduced in Sect. 25.3.1. However, for the particular problem at hand, additional inequalities can be considered.

25.4.2.1 Time Windows Inequalities

A straightforward set of inequalities are based on the earliest and latest starting time for each activity. They are very popular in mixed-integer linear programming formulations for project scheduling problems and can be applied as follows.

Let d_{ij} be the length of the longest path from node i to node j in the precedence network, $i, j \in V$, $i < j$. As in the previous section, denote by UB a valid upper bound on the optimal project makespan. For each activity $i \in V$ denote by ES_i and LS_i its earliest starting time and its latest starting time, respectively.

In project scheduling, it is well-known that activity i cannot start before time d_{0i}, i.e., $ES_i = d_{0i}$. On the other hand, taking into account that once starting the processing of activity i one has to wait at least $d_{i,n+1}$ until the project is finished, one can set $LS_i = UB - d_{i,n+1}$. The observations above define a time window for the starting time of each activity i:

$$ES_i \le S_i \le LS_i \tag{25.29}$$

which can be added to the formulation proposed in the previous section.

Correia et al. (2012) propose a procedure for obtaining UB.

25.4.2.2 Reduction Tests

Correia et al. (2012) add two sufficient conditions to the ones presented in Sect. 25.3.1 namely the following:

Two activities $i, j \in V$, are incompatible if

C3 $LS_i + p_i < ES_j$
C4 $LS_j + p_j < ES_i$

Consider two activities $i, j \in V$ and assume that they do not satisfy any of the conditions C1 to C4 (i.e., the value of variable y_{ij} in model $P1$ can still be chosen). Assume also that activity i cannot be finished before j. In this case, y_{ij} cannot have value 1 and thus we can set it to 0.

Assume now that activities i and j are incompatible but without any precedence relation existing between them (i.e., the value of variable y_{ij} in model $P1$ can still be chosen). In this case, either i must be executed before j or the other way round. In

Algorithm 25.1: Reduction Test

```
Require: two activities $i$ and $j$
  if $i$ and $j$ are not incompatible and if $ES_i + p_i > LS_j$ then
    $y_{ij} = 0$
  end if
  if $i$ and $j$ are incompatible with no precedence constraint existing between them then
    if $ES_i + p_i > LS_j$ then
      $y_{ji} = 1,\ y_{ij} = 0$
      $y_{js} = 1,\ y_{sj} = 0$, for all $s \in V$ such that $y_{is} = 1$
      $y_{pi} = 1,\ y_{ip} = 0$, for all $p \in V$ such that $y_{pj} = 1$
      $y_{ps} = 1,\ y_{sp} = 0$, for all $p \in V$ such that $y_{pj} = 1$ and for all $s \in V$ such that $y_{is} = 1$
    else if $ES_j + p_j > LS_i$ then
      $y_{ij} = 1,\ y_{ji} = 0$
      $y_{is} = 1,\ y_{si} = 0$, for all $s \in V$ such that $y_{js} = 1$
      $y_{pj} = 1,\ y_{jp} = 0$, for all $p \in V$ such that $y_{pi} = 1$
      $y_{ps} = 1,\ y_{sp} = 0$, for all $p \in V$ such that $y_{pi} = 1$ and for $s \in V$ such that $y_{js} = 1$
    end if
  end if
```

some circumstances it is possible in advance to realize which is the case. Moreover, when we realize that one activity i should be executed before the other activity j, then by transitivity we can possibly fix some other y variables. Therefore, the reduction test proposed by Correia et al. (2012) is properly justified.

As pointed out by Correia et al. (2012) the second part of the reduction test (when i and j are incompatible) is an adaptation to this problem of the well known *pair test* proposed by Carlier and Pinson (1989) for the job-shop scheduling problem.

25.4.2.3 Other Inequalities

As mentioned above, the additional inequalities presented in Sect. 25.3.1 can be applied to enhance $P1$. This is the case, with conditions (25.15) and (25.16).

Finally, Correia et al. (2012) considered constraints (25.18) with $WL'_k = UB$ for all $k \in \mathscr{R}$. As mentioned before, the authors proposed a heuristic procedure for obtaining a value UB.

25.5 A Cost Minimization Multi-Skill Resource-Constrained Project Staffing and Scheduling Problem

Li and Womer (2009) address a unit capacity multi-skill project scheduling problem proposing a model which can be seen as a particularization of P. In the problem addressed by these authors we have:

- The objective function represents the cost associated with the resources used in the project.
- $r_{il} = 1, i \in V, l \in \mathscr{L}_i$.
- $\delta_{ij} \in \{d_{ij}^{min}, d_{ji}^{max}, p_i\}$.

The model proposed is the following:

$$(P2) \quad \text{Min.} \sum_{k \in \mathscr{R}} c_k z_k \tag{25.30}$$

$$\text{s.t.} \quad S_j \geq S_i + \delta_{ij} \qquad (i, j \in V) \tag{25.31}$$

$$S_j \geq S_i + p_i - M\left(1 - y_{ij}\right) \quad (i, j \in V) \tag{25.32}$$

$$\sum_{k \in \mathscr{R}_l} x_{ikl} = 1 \quad (i \in V;\ l \in \mathscr{L}_i) \tag{25.33}$$

$$\sum_{l \in \mathscr{L}_i \cap \mathscr{L}_k} x_{ikl} \leq 1 \quad (k \in \mathscr{R};\ i \in V_k) \tag{25.34}$$

$$x_{ikl} + x_{jkl'} \leq y_{ij} + y_{ji} + 1 \quad (k \in \mathscr{R};\ i, j \in V_k;\ l \in \mathscr{L}_k \cap \mathscr{L}_i;\ l' \in \mathscr{L}_k \cap \mathscr{L}_j) \tag{25.35}$$

$$S_i + p_i \leq \min\{\overline{d}_i, \overline{d}\} \quad (i \in V) \tag{25.36}$$

$$\sum_{i \in V_k} \sum_{l \in \mathscr{L}_k \cap \mathscr{L}_i} p_i x_{ikl} \leq WL_k z_k \quad (k \in \mathscr{R}) \tag{25.37}$$

$$S_i \geq 0 \quad (i \in V) \tag{25.38}$$

$$x_{ikl} \in \{0, 1\} \quad (i \in V;\ k \in \mathscr{R}_i;\ l \in \mathscr{L}_k \cap \mathscr{L}_i) \tag{25.39}$$

$$y_{ij} \in \{0, 1\} \quad (i, j \in V) \tag{25.40}$$

$$z_k \in \{0, 1\} \quad (k \in \mathscr{R}) \tag{25.41}$$

In the above formulation one observes that constraints (25.6) presented in Sect. 25.3 are disaggregated leading to (25.35).

For this problem and using model $P2$, Li and Womer (2009) develop a Benders decomposition approach (see Chap. 27 for further details).

25.6 Extensions

So far we have considered project staffing and scheduling problems in the context of a project scheduling problem. Heimerl and Kolisch (2010) extend the basic setting by considering a multi-project problem. Each project can be seen as one activity with time dependent resource requirements. Several types of resources are considered namely internal to the organization and external. The latter convey outsourcing. Resources are assumed to be heterogeneous as far as the efficiencies are concerned. The objective is to minimize the total cost associated with the resources. This cost includes regular and overtime work for the internal resources and work for the outsourced resources. Resources may work in different projects at the same time.

The problem above is a very general multi-project staffing and scheduling problem. For this problem, Heimerl and Kolisch (2010) propose a MILP model, which they enhance using valid inequalities.

Another extension can be found in Chap. 28 where the basic setting of a multi-skill project staffing and scheduling problem is extended in order to accommodate some needs in an industrial context. In particular, the synchronization of task skills or the ability to preempt some tasks is now considered.

25.7 Conclusions

In this chapter a general MILP modeling framework was presented for project staffing and scheduling problems. A single-project, multi-skill setting was considered. The general modeling framework was particularized to two problems that have been studied in the literature. Several sets of valid inequalities and reduction tests were presented, which aim at enhancing the MILP models for the purpose of using a commercial solver. When the use of a commercial solver fails to solve one instance of this type of problems, specially tailored approaches must be attempted.

References

Alvarez-Valdes R, Tamarit JM (1993) The project scheduling polyhedron: dimension, facets and lifting theorems. Eur J Oper Res 67:204–220

Applegate D, Cook W (1991) A computational study of job-shop scheduling. ORSA J Comp 3:149–156

Baptiste P, Demassey S (2004) Tight LP bounds for resource constrained project scheduling. OR Spectr 26:251–262

Bellenguez O, Néron E (2005) Lower bounds for the multi-skill project scheduling problem with hierarchical levels of skills. In: Practice and theory of automated timetabling V. Lecture notes in computer science, vol 3616. Springer, Berlin, pp 229–243

Carlier J, Néron E (2003) On linear lower bounds for the resource constrained project scheduling problem. Eur J Oper Res 149:314–324

Carlier J, Pinson E (1989) An algorithm for solving the job-shop problem. Manag Sci 35:164–176

Christofides N, Alvarez-Valdés R, Tamarit JM (1987) Project scheduling with resource constraints: a branch and bound approach. Eur J Oper Res 29:262–273

Correia I, Lampreia-Lourenço L, Saldanha-da-Gama F (2012) Project scheduling with flexible resources: formulation and inequalities. OR Spectr 34:635–663

Demassey S (2008) Mathematical programming formulations and lower bounds. In: Artigues C, Demassey S, Néron E (eds) Resource-constrained project scheduling: models, algorithms, extensions and applications. Wiley, Hoboken, pp 49–62

Heimerl C, Kolisch R (2010) Scheduling and staffing multiple projects with a multi-skilled workforce. OR Spectr 32:343–368

Klein R, Scholl A (1999) Computing lower bounds by destructive improvement: an application to resource-constrained project scheduling problem. Eur J Oper Res 112:322–346

Li H, Womer K (2009) Scheduling projects with multi-skilled personal by a hybrid MILP/CP Benders decomposition algorithm. J Sched 12:281–298

Chapter 26
Integrated Column Generation and Lagrangian Relaxation Approach for the Multi-Skill Project Scheduling Problem

Carlos Montoya, Odile Bellenguez-Morineau, Eric Pinson, and David Rivreau

Abstract This chapter introduces a procedure to solve the Multi-Skill Project Scheduling Problem. The problem combines both the classical Resource-Constrained Project Scheduling Problem and the multi-purpose machine model. The aim is to find a schedule that minimizes the completion time (makespan) of a project composed of a set of activities. Precedence relations and resources constraints are considered. In this problem, resources are staff members that master several skills. Thus, a given number of workers must be assigned to perform each skill required by an activity. Practical applications include the construction of buildings, as well as production and software development planning. We present an approach that integrates the utilization of Lagrangian relaxation and column generation for obtaining strong makespan lower bounds. Finally, we present the corresponding obtained results.

Keywords Column Generation • Lagrangian relaxation • Multi-skilled personnel • Project scheduling • Project staffing • Resource constraints

26.1 Introduction

In project scheduling, resource capacity, cost, and resource availabilities play an important role for obtaining a schedule that reaches the goals of a company. Different objectives can be considered in project scheduling such as the maximization of

C. Montoya (✉)
Department of Industrial Engineering, Universidad de los Andes, Bogotá D.C, Colombia
e-mail: ce.montoya@uniandes.edu.co

O. Bellenguez-Morineau
IRCCyN, Ecole des Mines de Nantes, Nantes, France
e-mail: Odile.morineau@mines-nantes.fr

E. Pinson • D. Rivreau
LARIS (EA CNRS 4094), Université Catholique de l'Ouest, Angers, France
e-mail: eric.pinson@uco.fr;david.rivreau@uco.fr

C. Schwindt, J. Zimmermann (eds.), *Handbook on Project Management and Scheduling Vol. 1*, International Handbooks on Information Systems,
DOI 10.1007/978-3-319-05443-8_26

the net present value, the minimization of the resource costs, or the minimization of the project duration (makespan). All of these features are addressed by the resource-constrained project scheduling problems like the RCPSP (Brucker et al. 1999), which are classical scheduling problems that received major attention in the last years.

The RCPSP ($PS \mid prec \mid C_{max}$) deals with a given number of activities that have to be scheduled on a set of resources. It also takes into account precedence relations between activities and limited resource availabilities. The RCPSP considers renewable resources, which can be used whenever they are available (e.g., staff members, machines, and equipment).

The interest in extending the practical applications of the RCPSP encouraged researchers to work on different extensions that capture several variants and features related to specific real life situations (Artigues et al. 2008; Hartmann and Briskorn 2010).

We focus on one particular extension of the RCPSP, which can be classified as a Project Staffing Problem, known as the Multi-Skill Project Scheduling Problem MSPSP ($PSS \mid prec \mid C_{max}$, Bellenguez-Morineau and Néron 2005). It is important to mention that the main features of this problem are also described in Chaps. 25 and 28. Furthermore, we can outline that the MSPSP mixes both the RCPSP and the Multi-Purpose Machine model. The aim is to find a schedule that minimizes the makespan of a project, considering that resources are staff members that master several skills. Practical applications include the construction of buildings, production, and software development planning. In addition it is important to notice that this problem is $\mathcal{NP}$-hard in the strong sense (Artigues et al. 2008).

In this chapter we propose two Lagrangian relaxation models, which are inspired by a column generation (CG) approach proposed in previous work (Montoya et al. 2013). The proposed approach aims at obtaining strong makespan lower bounds. This chapter is organized as follows: In Sect. 26.2 we present a literature review related to the studied problem and to the methods we use to solve it. In Sect. 26.3 we define the problem and give an overview about column generation and subsequently we introduce two master problem formulations. In Sect. 26.4 we propose two Lagrangian relaxation models. In Sect. 26.5, we introduce the procedures used for initializing the CG procedure and for selecting new columns. In Sect. 26.6 we report the computational results. Finally, in Sect. 26.7 we conclude on our work and discuss possible research avenues.

26.2 Literature Review

Before introducing the proposed solution method, we present a literature review related to the studied problem and to the methods we use to solve it.

26.2.1 Problem Background

Despite the fact that the notion of skills plays an important role in the field of personnel assignment (Jiang et al. 2004), it is not often considered in the project scheduling field. Regarding the Multi-Skill Project Scheduling Problem, we refer to the work done by Bellenguez-Morineau and Néron (2005), Bellenguez-Morineau (2008), who developed and implemented different procedures to determine lower and upper bounds for the makespan. More recently, Montoya et al. (2013) proposed a branch-and-price approach for the MSPSP. Other works have been done also on some specific variants for the MSPSP. For example, Cordeau et al. (2010) developed a construction heuristic and an adaptive large neighborhood search heuristic for the Technician and Task Scheduling Problem (TTSP) in a large telecommunications company. The goal in this problem is to assign technicians to tasks with multi-level skill requirements. Here, as it occurs in the MSPSP, the requirements of the tasks (activities) are defined by the presence of a set of resources (technicians) that possess the necessary skills. For solving this last mentioned problem, Fırat and Hurkens (2012) developed more recently a solution methodology that uses a flexible matching model as a core engine based on an underlying mixed-integer programming model.

Additionally, there are other interesting solution methodologies in the literature of project scheduling with multi-skilled human resources. For example, Heimerl and Kolisch (2010) proposed a mixed-integer linear program to solve a multi-project problem where the schedule of each project is fixed. They also considered multi-skilled human workforce with heterogeneous and static efficiencies. Li and Womer (2009) developed a hybrid algorithm based on mixed-integer modeling and constraint programming for solving a project scheduling problem with multi-skilled personnel, taking into consideration an individual workload capacity for each worker. Gutjahr et al. (2008) proposed a greedy heuristic and a hybrid solution methodology using priority-based rules, ant colony optimization, and a genetic algorithm to solve the so-called “Project Selection, Scheduling and Staffing with Learning Problem”. More recently, as was introduced in Chap. 25, Correia et al. (2012) presented a mixed-integer linear programming formulation and several sets of additional inequalities for a variant of the resource-constrained project scheduling problem in which resources are flexible i.e., each resource has several skills.

26.2.2 Column Generation Overview and Background

Introduced independently by Dantzig and Wolfe (1960) and Gilmore and Gomory (1961), column generation (CG) consists in solving alternately a (restricted) master problem (RMP) and a sub-problem resulting from the decomposition of the original problem. The main idea underlying this approach is to select, at each iteration of an iterative process and by means of an oracle related to the column generation sub-problem and using duality information originating from the solving of the current RMP, a candidate variable whose reduced cost is susceptible to improve the objective function associated with the master problem. The related column is added to the current RMP, and we iterate until no more candidate can be found. Notice that such a decomposition is possible due to the exploitation of some specific structure of the problem formulation whose pricing sub-problem leads to an "easier" optimization problem such as shortest path or knapsack problems.

So far column generation (CG) had been used to solve specifically the Multi-Skill Project Scheduling Problem by Montoya et al. (2013). Additionally CG has been used in combination with other optimization techniques for solving project scheduling problems. Particularly, Brucker et al. (1999) implemented a destructive approach for finding tight lower bounds for the RCPSP by using both constraint propagation techniques and CG. Afterwards, authors extended their work for solving the Multi-Mode RCPSP (MRCPSP) (Brucker and Knust 2000). Additionally, Van den Akker et al. (2007) presented a destructive lower bound based on column generation for certain extensions of the RCPSP. In this approach, authors used a simulated annealing algorithm for finding a schedule for each resource, which was enforced by a time-indexed integer programming formulation.

On the other hand, CG has been widely used on the Vehicle Routing Problem (VRP) and several related extensions (Lübbecke and Desrosiers 2005). Some VRP problems consider similar features to those of the MSPSP. For example, Dohn et al. (2009) deal with an assignment problem where a set of teams must be assigned to a set of tasks, restricted by time-windows. As it occurs in the MSPSP, assigned resources must start and finish a given activity simultaneously. Authors developed a branch-and-price procedure and enforce the fulfillment of such a constraint with a branching scheme that limits the starting time of a given activity. Moreover, for a particular extension of the VRP, Ioachim et al. (1999) modeled the synchronization constraint directly in the master problem with the consequence that a large number of columns with a small variation in the starting times (departure times) of the tasks (flights) are generated. To handle such a drawback, they introduced a tolerance in the side constraints to allow asynchronous departure times.

Additionally, column generation has also been used to solve Staff Scheduling Problems (Bard and Purnomo 2005; Beliën and Demeulemeester 2007; Jaumard et al. 1998; Mason and Smith 1998; Mehrotra et al. 2000). This type of problems, as it occurs with the MSPSP, involves the assignment of staff members to perform a set of activities, but they normally intend to minimize a total assignment cost, considering also a predefined time horizon.

Finally, CG has also been used to solve shop scheduling problems (Chen and Powell 1999; Gélinas and Soumis 2005; Van den Akker et al. 2000, 2002). These problems are related to single machine, flexible and job shop scheduling problems, sharing also some common features with the MSPSP. For example, Gélinas and Soumis (2005) deal with precedence constraints for solving the job shop scheduling problem. They handle these constraints in the master problem and modeled the sub-problem as a single machine problem with time constraints. On another side, Van den Akker et al. (2000) used CG based on a time-indexed formulation for solving single machine scheduling problems. They used Dantzig-Wolfe decomposition techniques (Dantzig and Wolfe 1960) to deal with the difficulties related to the size of a time-indexed formulation, given its capacity to obtain strong lower bounds. This approach supports the one considered in this chapter, since our work is also based on a time-indexed perspective.

26.2.3 Combining Lagrangian Relaxation and Column Generation Background

Typically, column generation is used to solve the LP-relaxation of the master problem, but it can also be combined with Lagrangian relaxation as we will discuss in this chapter. According to Wolsey (1998), it is possible to solve the Lagrangian dual either by means of the subgradient method or by solving the linear relaxation of the extensive formulation by using a CG approach. Thereafter, the optimal lower bound of the restricted linear master problem (RMP) and the best Lagrangian dual will have the same value. Both solution methods for the Lagrangian dual have advantages and disadvantages, hence some authors have proposed procedures that try to combine the advantages of both approaches (Barahona and Jensen 1998; Huisman et al. 2005).

As we will show later on, each Lagrangian multiplier vector is linked with the dual variables related to the relaxed constraint. Consequently, this implies that the dual values obtained by solving the RMP can be estimated by the Lagrangian multipliers used in the related Lagrangian dual problem. Thus, instead of solving the RMP with the simplex method by using a solver, we can use a subgradient procedure (Held and Karp 1971) for solving the Lagrangian dual approximately. The associated Lagrangian multipliers can be used for estimating the values of the dual variables related to the constraints of the RMP and for pricing out new columns.

Overall, there are different reasons for using this last mentioned approach. The subgradient method is fast, easy to implement, and does not require a commercial solver. When solving the RMP with a simplex method, we obtain a basic dual solution that corresponds to a vertex of the optimal face of the dual polyhedron. Given that a new column of the RMP may cut that vertex, a dual solution interior (in the center) of the dual face allows stronger dual cuts (i.e., better primal columns). Bixby et al. (1992) and Barnhart et al. (1998) obtained from their research that this may improve the convergence of a column generation algorithm and reduce degeneracy. Jans and Degraeve (2004) and Huisman et al. (2005) provide computational results that indicate that Lagrangian multipliers are beneficial. Finally, it is also shown that during the subgradient phase, possible feasible solutions are generated.

26.3 Problem Description and Column Generation

As was already mentioned, the Multi-Skill Project Scheduling Problem MSPSP is a known project scheduling problem, which is mainly composed by three components: activities, resources, and skills. It considers a set V of n activities. Within this set, we also define two dummy activities 0 and $n + 1$ which represent respectively the beginning and completion of the project. Let $Succ(i)$ denote the set of immediate successors of an activity i whose processing time is denoted by p_i. Additionally, a set $\mathscr{R}$ of K workers and a set $\mathscr{L}$ of L skills are defined for performing these activities. We denote by r_{il} the number of workers with skill l required by activity i. We define $a_{kl} = 1$ if worker k masters skill l, 0 otherwise. Finally, T denotes an upper bound for the total duration of the project or makespan.

After understanding the main features of the MSPSP, we introduce the two master problem formulations related to the two Lagrangian relaxation models exploited in this chapter.

26.3.1 Column Generation Master Problem Formulations

The basic idea underlying the considered CG approach relies on a time-indexed reformulation of the problem. In this mathematical formalization, a column ω (i.e., activity work pattern) represents certain processing attributes (a starting time and set of assigned workers) of an activity i. Hence, a solution to the MSPSP consists in identifying a single column (a starting time and a set of assigned workers) for each activity of the project. An activity work pattern ω is represented by three parameters: (i) ϵ_i^{ω}, which takes the value of 1 if activity i is processed in activity work pattern ω, 0 otherwise; (ii) Υ_i^{ω}, which takes the value of t if activity i starts at time t in activity pattern ω, 0 otherwise; (iii) ζ_{kt}^{ω}, which takes the value of 1 if worker k is assigned on activity work pattern ω at time t, 0 otherwise. We assume that the workers assigned to ω satisfy the skills requirement of the related activity. Additionally, we denote the set of all feasible activity work patterns by Ω.

The decision variables governing the target model are defined by: (i) x_{ω}, which takes the value of 1 if activity work pattern ω is selected, 0 otherwise; (ii) S_i, which represents the starting time of an activity i.

Based on the previous description we present two master problem formulations (MP_1 and MP_2), that lead to the same sub-problem (SP). Moreover, we can state that solving the SP aims to exhibit a feasible selection of workers/skills for processing activity i at time t.

In the next subsections we introduce MP_1 and MP_2, and later on, we explain the sub-problem and the solution method applied to solve it.

26.3.1.1 First Master Problem (MP_1)

The first master problem MP_1 can be stated as follows:

$$Z[MP_1]\text{: Min. } S_{n+1} \tag{26.1}$$

$$\text{s.t.} \quad \sum_{\omega \in \Omega} (x_\omega \cdot \epsilon_i^\omega) = 1 \quad (i \in V) \tag{26.2}$$

$$\sum_{\omega \in \Omega} (x_\omega \cdot \Upsilon_i^\omega) = S_i \quad (i \in V) \tag{26.3}$$

$$\sum_{\omega \in \Omega} (x_\omega \cdot \zeta_{kt}^\omega) \leq 1 \quad (k \in \mathscr{R};\ t = 0, \ldots, T) \tag{26.4}$$

$$S_i + p_i \leq S_j \quad (i \in V;\ j \in Succ(i)) \tag{26.5}$$

$$ES_i \leq S_i \leq LS_i \quad (i \in V) \tag{26.6}$$

$$x_\omega \in \{0, 1\} \quad (\omega \in \Omega) \tag{26.7}$$

The objective is to minimize the makespan (26.1). Constraint set (26.2) ensures that only a unique activity work pattern can be assigned to any task i. Constraints (26.3) recover the associated starting times, while constraint set (26.4) ensures that any operator can carry out at most one activity at a given time. Constraints (26.5) state the precedence relations connecting the activities in V, and constraint set (26.6) ensures that the starting time of each activity must be within a predefined time-window. For this purpose, ES_i (resp. LS_i) denotes in this formulation a lower bound (resp. upper) for the starting date associated with activity i. Such a time-window is for instance simply induced by the precedence graph using recursively Bellman's conditions and a given upper bound (T) for the makespan.

Thereafter, for applying CG, integrality constraints (26.7) are relaxed. The (restricted) master problem (RMP_1) is then obtained by considering a partial pool of activity work patterns $\bar{\Omega}$ ($\bar{\Omega} \subseteq \Omega$). Assuming that an optimal solution of the RMP_1 has been computed with a standard LP solver, let us denote the simplex multipliers associated with constraints (26.2), (26.3), and (26.4) respectively by ($\rho_i, i \in V$), ($\tau_i, i \in V$), and ($\theta_{kt}, k \in \mathscr{R}, t = 0, \ldots, T$). The reduced cost associated with an activity work pattern ω related to the processing of activity i at time t can be stated as follows:

$$rc_{it} = 0 - \rho_i - (\tau_i \cdot t) - \sum_{k \in \mathscr{R}} \sum_{t'=t}^{t'=t+p_i-1} (\theta_{kt'} \cdot \zeta_{kt'}^\omega) = rc_{it}^1 + rc_{it}^2 \tag{26.8}$$

with:

$$rc_{it}^1 = -\rho_i - (\tau_i \cdot t) \tag{26.9}$$

$$rc_{it}^2 = -\sum_{k \in \mathscr{R}} \sum_{t'=t}^{t'=t+p_i-1} (\theta_{kt'} \cdot \zeta_{kt'}^{\omega}) \qquad (26.10)$$

Hence, if $rc_{it} < 0$, then the corresponding column (activity work pattern) can be added to the current pool of columns, iteratively, until no more profitable columns can be found (Gilmore and Gomory 1961).

26.3.1.2 Second Master Problem (MP_2)

This new master problem formulation has a similar structure to the one previously explained. This new mathematical model relies on an alternative way for integrating the precedence relations constraints. Hence, let us consider a particular precedence relation between two activities i and j $(j \in Succ(i))$ which must be processed in the time slot $W_{ij} = ES_i, \ldots, LS_j + p_j$. Mapping this time slot, we can extend the notion of an activity work pattern ω by defining a new parameter $\nu_{ijt'}^{\omega}$ in the following way:

If ω corresponds to the execution of i at a starting time t:

- $\nu_{ijt'}^{\omega} = 1 \quad (t' = ES_i, \ldots, t + p_i - 1)$;
- $\nu_{ijt'}^{\omega} = 0 \quad (t' = t + p_i, \ldots, LS_j + p_j)$.

If ω corresponds to the execution of j at a starting time t:

- $\nu_{ijt'}^{\omega} = 1 \quad (t' = ES_i, \ldots, t - 1)$;
- $\nu_{ijt'}^{\omega} = 0 \quad (t' = t, \ldots, LS_j + p_j)$.

If ω does not correspond to the execution of neither i nor j:

- $\nu_{ijt'}^{\omega} = 0 \quad (t' = ES_i, \ldots, LS_j + p_j)$.

Clearly, two activity work patterns ω_1 and ω_2 should be consistent for the precedence constraint of i and j, if:

$$(\nu_{ijt'}^{\omega_1} \cdot x_{\omega_1}) + (\nu_{ijt'}^{\omega_2} \cdot x_{\omega_2}) \leq 1 \quad (t' \in W_{ij}) \qquad (26.11)$$

Additionally, we also consider a coefficient C_{max}^{ω}, related to each activity work pattern ω. Hence, given that $S(\omega)$ represents the starting time linked to activity work pattern ω, we define C_{max}^{ω} as follows:

- $C_{max}^{\omega} = S(\omega)$ if activity work pattern ω is related to the execution of the dummy activity $n + 1$, 0 otherwise.

Notice that the proposed mathematical formulations are adapted for most of regular as well as nonregular optimization criteria.

Let us also recall that Ω represents the set of all feasible activity work patterns. The only decision variable governing the target model is x_{ω}, which was already introduced for MP_1. Therefore, the associated mathematical formulation can then be stated as follows:

$$Z[MP_2]\text{: Min. } \sum_{\omega \in \Omega} (C^{\omega}_{max} \cdot x_{\omega}) \tag{26.12}$$

$$\text{s. t. } \sum_{\omega \in \Omega} (x_{\omega} \cdot \epsilon^{\omega}_{i}) = 1 \quad (i \in V) \tag{26.13}$$

$$\sum_{\omega \in \Omega} (x_{\omega} \cdot \zeta^{\omega}_{kt}) \leq 1 \quad (k \in \mathscr{R};\ t = 0, \ldots, T) \tag{26.14}$$

$$\sum_{\omega \in \Omega} (x_{\omega} \cdot \nu^{\omega}_{ijt}) \leq 1 \quad (i \in V; j \in Succ(i);\ t \in W_{ij}) \tag{26.15}$$

$$x_{\omega} \in \{0, 1\} \quad (\omega \in \Omega) \tag{26.16}$$

Notice that this formulation has a structure (set packing problem) with a pure 0-1 coefficients constraint matrix. More precisely, we have that constraint set (26.13) ensures that only a unique activity work pattern can be assigned to any task i. Constraint set (26.14) ensures that any operator can carry out at most one activity at a given time. Constraint (26.15) states the precedence relations connecting the activities in V at given time-point t.

The (restricted) master problem (RMP_2) can simply be obtained by relaxing the binarity constraints relating to decision variables x_{ω} and by considering a partial pool of activity work patterns $\bar{\Omega}$ ($\bar{\Omega} \subseteq \Omega$). Thus, after solving the RMP_2 the corresponding simplex multipliers (dual variables) associated to constraints (26.13), (26.14), (26.15) are denoted respectively by ($\rho_i, i \in V$), ($\theta_{kt}, k \in \mathscr{R}, t = 0, \ldots, T$), and ($\eta_{ijt}, i \in V, j \in Succ(i), t \in W_{ij}$). Subsequently, the reduced cost associated with an activity work pattern ω related to the processing of activity i at time t can be stated as follows:

$$rc_{it} = 0 - \rho_i - \sum_{i \in V} \sum_{j \in Succ(i)} \sum_{t' \in W_{ij}} (\nu^{\omega}_{ijt'} \cdot \eta_{ijt'}) - \sum_{k \in \mathscr{R}} \sum_{t'=t}^{t'=t+p_i-1} (\theta_{kt'} \cdot \zeta^{\omega}_{kt'}) = rc^{1}_{it} + rc^{2}_{it} \tag{26.17}$$

with:

$$rc^{1}_{it} = -\rho_i - \sum_{i \in V} \sum_{j \in Succ(i)} \sum_{t' \in W_{ij}} (\nu^{\omega}_{ijt'} \cdot \eta_{ijt'}) \tag{26.18}$$

$$rc^{2}_{it} = - \sum_{k \in \mathscr{R}} \sum_{t'=t}^{t'=t+p_i-1} (\theta_{kt'} \cdot \zeta^{\omega}_{kt'}) \tag{26.19}$$

As was already mentioned a column is added to the current pool of columns, iteratively, until no more profitable columns can be found (Gilmore and Gomory 1961).

Now, before introducing the combined Lagrangian relaxation and column generation approaches proposed in the present chapter, we give an insight to the sub-problem (SP) and to the applied solution method.

26.3.1.3 Column Generation Sub-Problem (SP)

Assuming that an optimal solution of RMP_1 or RMP_2 has been computed, let us define the quantity c_{kt}, which simply corresponds to the total cost incurred by worker k when assigned to an activity i in the time slot $t, \ldots, t + p_i - 1$.

$$c_{kt} = - \sum_{t'=t}^{t+p_i-1} \theta_{kt'} \tag{26.20}$$

Subsequently, we consider the decision variables $y_k = 1$, if worker k is assigned to perform activity i, 0, otherwise, and $z_{kl} = 1$, if worker k uses skill l to perform activity i, 0 otherwise. Finding an optimal feasible selection of workers/skills for processing activity i at time t that minimizes the reduced cost rc_{it} leads to the following sub-problem $SP(i, t)$:

$$Z[SP(i,t)]\text{: Min. } rc_{it}^2 = \sum_{k \in \mathscr{R}} (c_{kt} \cdot y_k) \tag{26.21}$$

$$\text{s. t.} \quad \sum_{k \in \mathscr{R}} z_{kl} = r_{il} \quad (l \in \mathscr{L}) \tag{26.22}$$

$$y_k = \sum_{l \in \mathscr{L}} z_{kl} \quad (k \in \mathscr{R}) \tag{26.23}$$

$$y_k \in \{0, 1\}, \, z_{kl} \in \{0, 1\} \quad (k \in \mathscr{R}; l \in \mathscr{L}) \tag{26.24}$$

In this formulation, the objective is to minimize the total assignment cost to perform activity i at a time t. Constraint set (26.22) guarantees its requirements fulfillment. Constraint set (26.23) ensures that an assigned worker uses only one skill. Finally, constraint set (26.24) defines the decision variables as binary. Moreover, we can state that solving the SP aims to exhibit a feasible selection of workers/skills for processing activity i at time t.

Once $rc_{it}^2 = -Z[SP(i, t)]$ is computed, we get the targeted reduced cost defined by (26.8) and (26.17) related to the solution of the RMP_1 and RMP_2 respectively. If $rc_{it} < 0$, then the corresponding column is candidate to enter the basis since it leads to a decrease in the objective function value for the corresponding restricted master problem (RMP_1 or RMP_2). Consequently, this activity work pattern can be added to

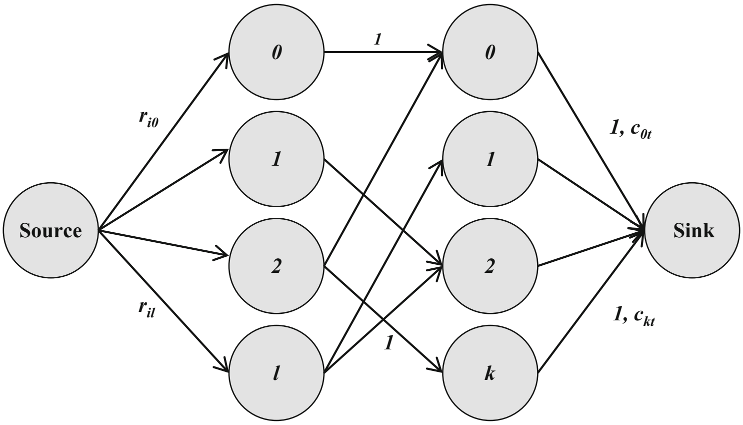


Fig. 26.1 Graph $G(i,t)$ skills assignment for activity i

the current pool of columns according to the parameters defined for each of the two proposed master problem formulations in Sects. 26.3.1.1 and 26.3.1.2.

Of course, an enumeration on each activity for each of its potential starting date ($ES_i \leq t_i \leq LS_i$) is necessary for exhibiting an activity work pattern with global minimal reduced cost. We refer to the next section for more details related to the global solution method.

26.3.1.4 Column Generation Sub-Problem Solution (SP)

Clearly, solving $SP(i,t)$ is equivalent to find an optimal solution to a min-cost max-flow problem on a particular network $G(i,t)$, whose structure is depicted in Fig. 26.1. This graph simply formalizes the assignment of the skills required for performing activity i to the workers mastering at least one of these skills, according to the constraint stating that any worker can use only one skill when performing a given activity. The values on each arc correspond respectively to its flow upper bound and the related unit cost. Notice that each arc $(l, sink)$ with a positive flow corresponds to a selected worker for the processing of activity i ($y_k = 1$). This classical flow problem can be efficiently solved using the algorithm proposed by Busacker and Gowen (1961) in $\mathscr{O}((K+L)^3)$ time complexity.

26.4 Combining Lagrangian Relaxation and Column Generation

One of the main proposal of this chapter relies on combining CG with Lagrangian relaxation, leading to a faster way of solving the (restricted) master problem rather than using an LP solver (Huisman et al. 2005). In the sequel, we present the two Lagrangian relaxation models related to the solution of RMP_1 and RMP_2, respectively.

26.4.1 RMP_1 Based Model for Combining Lagrangian Relaxation and Column Generation

Before introducing the Lagrangian relaxation model related to RMP_1, we include the next additional surrogate constraint to the (restricted) master problem described previously:

$$\sum_{t=0,\ldots,T}\sum_{k\in\mathscr{R}}\sum_{\omega\in\bar{\Omega}}(x_\omega\cdot\zeta_{kt}^{\omega})=\sum_{i\in V}\sum_{l\in\mathscr{L}}(p_i\cdot r_{il}) \tag{26.25}$$

Such a constraint establishes that the accumulated time per resource unit assigned (left term) must be equal to the total amount of time per resource unit required during the whole project duration (right term). Thereafter, let us associate with constraints (26.2), (26.3), and (26.4) the respective Lagrangian multipliers $(\rho_i, i \in V)$, $(\tau_i, i \in V)$ and $(\theta_{kt}, k \in \mathscr{R}, t = 0,\ldots,T)$. The corresponding Lagrangian function can be written as follows:

$$\Gamma(x,t,\rho,\tau,\theta)=S_{n+1}+\sum_{i\in V}\rho_i\cdot(1-\sum_{\omega\in\Omega}(x_\omega\cdot\epsilon_i^{\omega}))$$
$$+\sum_{i\in V}\tau_i\cdot(S_i-\sum_{\omega\in\Omega}(x_\omega\cdot\Upsilon_i^{\omega}))+\sum_{k\in\mathscr{R}}\sum_{t=0,\ldots,T}\theta_{kt}\cdot(1-\sum_{\omega\in\Omega}(x_\omega\cdot\zeta_{kt}^{\omega})) \tag{26.26}$$

$$\Gamma(x,t,\rho,\tau,\theta)=S_{n+1}+\sum_{i\in V}(\tau_i\cdot S_i)+\sum_{\omega\in\Omega}((-\epsilon_i^{\omega}\cdot\rho_i)-(\Upsilon_i^{\omega}\cdot\tau_i)$$
$$-\sum_{k\in\mathscr{R}}\sum_{t=0,\ldots,T}(\theta_{kt}\cdot\zeta_{kt}^{\omega}))\cdot x_\omega+\sum_{i\in V}\rho_i+\sum_{k\in\mathscr{R}}\sum_{t=0,\ldots,T}\theta_{kt} \tag{26.27}$$

For a given distribution (ρ,τ,θ) of Lagrangian multipliers, the associated dual function $L(\rho,\tau,\theta)$ can be computed solving the following independent Lagrangian sub-problems:

$$Z[LSP_1(\tau)]\text{: Min. } S_{n+1}+\sum_{i\in V}(\tau_i\cdot S_i) \tag{26.28}$$

$$\text{s.t.}\quad S_i+p_i\le S_j\quad (i\in V;\ j\in Succ(i)) \tag{26.29}$$

$$ES_i\le S_i\le LS_i\quad (i\in V) \tag{26.30}$$

and

$$Z[LSP_2(\rho,\tau,\theta,x)]\text{: Min. } \sum_{\omega\in\Omega}(C'^{\omega}_{max}\cdot x_\omega) \tag{26.31}$$

$$\text{s.t. } \sum_{\omega\in\Omega}(x_\omega\cdot wl_\omega)=\sum_{i\in V}\sum_{l\in\mathscr{L}}(p_i\cdot r_{il}) \tag{26.32}$$

$$x_\omega\in\{0,1\} \quad (\omega\in\Omega) \tag{26.33}$$

where

$$C'^{\omega}_{max}=-\rho_{i(\omega)}-(S(\omega)\cdot\tau_i)-\sum_{k\in\mathscr{R}}\sum_{t=0,\ldots,T}(\theta_{kt}\cdot\zeta^{\omega}_{kt}) \tag{26.34}$$

$$wl_\omega=\sum_{l\in\mathscr{L}}(p_{i(\omega)}\cdot r_{il}) \tag{26.35}$$

Hence, obviously we have:

$$\begin{aligned} Z[L(\rho,\tau,\theta)] &= Z[LSP_1(\rho,\tau,\theta,t)] \\ +Z[LSP_2(\rho,\tau,\theta,x)] &+ \sum_{i\in V}\rho_i+\sum_{k\in\mathscr{R}}\sum_{t=0,\ldots,T}\theta_{kt} \end{aligned} \tag{26.36}$$

Subsequently, we use a commercial solver for the solution of the first Lagrangian sub-problem ($LSP_1(\tau)$). In addition, it can be noticed that $LSP_2(\rho,\tau,\theta)$, is a classical 0-1 knapsack problem, which can be quite time consuming when the pool of patterns $\bar{\Omega}$ increases. This problem can be solved with a pseudo-polynomial time complexity of $\mathscr{O}(qB)$, where:

$$q=\mid\bar{\Omega}\mid \tag{26.37}$$

$$B=\sum_{i\in V}\sum_{l\in\mathscr{L}}(p_i\cdot r_{il}) \tag{26.38}$$

Nevertheless, we can focus on the linear relaxation of LSP_2 according to the activity work pattern generation process ($x_\omega\geq 0,\quad \omega\in\bar{\Omega}$). First, let us sort the activity work pattern in $\bar{\Omega}$ in such a way that:

$$C'^{\omega_1}_{max}/wl_{\omega_1}\leq C'^{\omega_2}_{max}/wl_{\omega_2}\leq\ldots\leq C'^{\omega_q}_{max}/wl_{\omega_q} \tag{26.39}$$

Now, after sorting activity work patterns according to (26.39), let s be the maximal index such that:

$$\sum_{j=0}^{s}wl_{\omega_j}\leq\sum_{i\in V}\sum_{l\in\mathscr{L}}(p_i\cdot r_{il}) \tag{26.40}$$

An optimal solution to LSP_2 is given by:

$$\bar{x}_{\omega_j} = 1 \quad (j = 0, \ldots, s) \tag{26.41}$$

$$\bar{x}_{\omega_{s+1}} = \frac{(\sum_{i \in V} \sum_{l \in \mathscr{L}} (p_i \cdot r_{il})) - wl_{\omega_j}}{wl_{\omega_{s+1}}} \tag{26.42}$$

$$\bar{x}_{\omega_j} = 0 \quad (j = s + 2, \ldots, q) \tag{26.43}$$

Considering that $Z[L(\rho, \tau, \theta)]$ defines a lower bound on the RMP_1, we can obtain the best possible lower bound by solving the related Lagrangian dual problem ($LDRMP_1$):

$$Z[LDRMP_1] = \mathrm{Max}_{\rho,\tau,\theta} L(\rho, \tau, \theta) \tag{26.44}$$

In the context of combinatorial optimization one efficient way to solve the Lagrangian dual problem is to use a subgradient procedure introduced by Held and Karp (1971), which iteratively updates the Lagrangian multipliers. Nevertheless other methods like volume (Barahona and Anbil 2000), bundle (Fábián 2000), and analytic center cutting plane methods (Goffin and Vial 2002) among others (Bertsekas 1999) can be used for solving the Lagrangian dual problem. We focus particularly on the description of the subgradient method since it is the most diffused one. In addition, besides the fact that it was the first one used in the context of combinatorial optimization, it has, at least, two main advantages: it is easy to code and has minimal memory requirements.

Thereafter, we use the subgradient procedure for estimating the values of the Lagrangian multipliers (ρ, τ, θ). Hence, after solving $LSP_1(\tau)$ and $LSP_2(\rho, \tau, \theta)$ we obtain the starting times vector S and the column assignment vector $\bar{x}$ from the solution of the respective sub-problem. Consequently, we can define a subgradient for $L(\rho, \tau, \theta)$ as follows:

$$\psi_i^1 = 1 - \sum_{\omega \in \Omega} (x_\omega \cdot \epsilon_i^\omega) \quad (i \in V) \tag{26.45}$$

$$\psi_i^2 = S_i - \sum_{\omega \in \Omega} (x_\omega \cdot \Upsilon_i^\omega) \quad (i \in V) \tag{26.46}$$

$$\varphi_{kt} = (1 - \sum_{\omega \in \Omega} (x_\omega \cdot \zeta_{kt}^\omega)) \quad (k \in \mathscr{R};\ t = 0, \ldots, T) \tag{26.47}$$

Now, given an upper bound T for the makespan and a step size *sp*, we can update the current Lagrangian multipliers by:

$$\rho_i = \rho_i + (sp \cdot \psi_i^1) \quad (i \in V) \tag{26.48}$$

$$\tau_i = \tau_i + (sp \cdot \psi_i^2) \quad (i \in V) \tag{26.49}$$

$$\theta_{kt} = \min\{0, \theta_{kt} + (sp \cdot \varphi_{kt})\} \quad (k \in \mathscr{R};\ t = 0, \ldots, T) \tag{26.50}$$

The step size *sp* is defined as follows:

$$sp = vp \cdot \frac{(T - L(\rho, \tau, \theta))}{Norm} \tag{26.51}$$

where *Norm* is given by:

$$Norm = \sum_{i \in V} (\psi_i^1 + \psi_i^2)^2 + \sum_{k \in \mathscr{R}} \sum_{t=0,\ldots,T} \varphi_{kt}{}^2 \tag{26.52}$$

Equation (26.51) represents a known step length rule, which was empirically justified by Held et al. (1974). Moreover, this rule is less expensive in terms of CPU times in comparison to other step length rules with proven convergence (Polyak 1967). Notice that in this equation we consider a parameter *vp*, which is initialized with a value equal to 2 ($vp = 2$), as was proposed by Held and Karp (1971). Additionally it is important to set a limit on the number of iterations, which defines also the accuracy of the solution.

Overall, the general idea is to update the Lagrangian multipliers during a limited number of iterations. Thereafter, we use the last updated multipliers as an estimation of the dual multipliers for pricing out new columns. At the end of each iteration we update the parameter *vp* with a systematic geometric revision: $vp = gp \cdot vp$. Normally the second parameter *gp* ranges between 0.87 and 0.9995, depending on the targeting problem.

Finally, when there are no more columns with a negative reduced cost by using the Lagrangian multipliers, we perform a fixed number of iterations solving the RMP_1 with the simplex method by using the solver for obtaining the values of the dual variables for pricing out new columns. Hence, if after the fixed number of iterations there are still columns with negative reduced costs, we go back to the Lagrangian procedure, otherwise, we stop.

26.4.2 RMP_2 Based Model for Combining Lagrangian Relaxation and Column Generation

The second Lagrangian model proposed is based on the second (restricted) master problem formulation (RMP_2) proposed in Sect. 26.3.1.2. Now, let us associate with constraints (26.13), (26.15), and (26.14) the respective Lagrangian multipliers $(\rho_i, i \in V)$, $(\eta_{ijt}, i \in V, j \in Succ(i), t \in W_{ij})$, and $(\theta_{kt}, k \in \mathscr{R}, t = 0, \ldots, T)$. The corresponding Lagrangian function can be written as follows:

$$\Gamma(x, \rho, \eta, \theta) = \sum_{\omega \in \Omega} (C_{max}^{\omega} \cdot x_{\omega}) + \sum_{i \in V} \rho_i \cdot (1 - \sum_{\omega \in \Omega} (x_{\omega} \cdot \epsilon_i^{\omega}))$$

$$+\sum_{i\in V}\sum_{j\in Succ(i)}\sum_{t\in W_{ij}}\eta_{ijt}\cdot(1-\sum_{\omega\in\Omega}(x_\omega\cdot\nu^{\omega}_{ijt}))$$

$$+\sum_{k\in\mathscr{R}}\sum_{t=0,\dots,T}\theta_{kt}\cdot(1-\sum_{\omega\in\Omega}(x_\omega\cdot\zeta^{\omega}_{kt})) \tag{26.53}$$

$$\begin{aligned}\Gamma(x,\rho,\eta,\theta) = \sum_{\omega\in\Omega}(C^{\omega}_{max}-(\epsilon^{\omega}_{i}\cdot\rho_i)-\sum_{i\in V}\sum_{j\in Succ(i)}\sum_{t\in W_{ij}}(\nu^{\omega}_{ijt}\cdot\eta_{ijt})\\ -\sum_{k\in\mathscr{R}}\sum_{t=0,\dots,T}(\theta_{kt}\cdot\zeta^{\omega}_{kt}))\cdot x_\omega+\sum_{i\in V}\rho_i\\ +\sum_{i\in V}\sum_{j\in Succ(i)}\sum_{t\in W_{ij}}\eta_{ijt}+\sum_{k\in\mathscr{R}}\sum_{t=0,\dots,T}\theta_{kt}\end{aligned} \tag{26.54}$$

For given values (ρ,η,θ) of Lagrangian multipliers, the associated dual function $L(\rho,\eta,\theta)$ can be computed solving the following Lagrangian sub-problem:

$$Z[LSP(\rho,\eta,\theta,x)]\text{: Min. }\sum_{\omega\in\Omega}(C'^{\omega}_{max}\cdot x_\omega) \tag{26.55}$$

$$\text{s.t.}\quad x_\omega\geq 0\quad(\omega\in\Omega) \tag{26.56}$$

where

$$C'^{\omega}_{max}=C^{\omega}_{max}-\epsilon^{\omega}_{i}\cdot\rho_{i(\omega)}-\sum_{i\in V}\sum_{j\in Succ(i)}\sum_{t\in W_{ij}}(\nu^{\omega}_{ijt}\cdot\eta_{ijt})-\sum_{k\in\mathscr{R}}\sum_{t=0,\dots,T}(\theta_{kt}\cdot\zeta^{\omega}_{kt}) \tag{26.57}$$

Hence, obviously we have:

$$\begin{aligned}Z[L(\rho,\eta,\theta)] = Z[LSP(\rho,\eta,\theta)]+\sum_{i\in V}\rho_i\\ +\sum_{i\in V}\sum_{j\in Succ(i)}\sum_{t\in W_{ij}}\eta_{ijt}+\sum_{k\in\mathscr{R}}\sum_{t=0,\dots,T}\theta_{kt}\end{aligned} \tag{26.58}$$

The Lagrangian sub-problem $(LSP(\rho,\eta,\theta))$ can be solved to optimality by setting $\bar{x}_\omega$ equal to 1 if $C'^{\omega}_{max}<0$, or equal to 0 otherwise.

$Z[L(\rho,\eta,\theta)]$ defines a lower bound on the RMP_2, given that each feasible solution for the original problem is also feasible for the Lagrangian function. Hence, we can obtain the best possible lower bound by solving the Lagrangian dual problem $(LDRMP_2)$:

$$Z[LDRMP_2]=\text{Max}_{\rho,\eta,\theta}L(\rho,\eta,\theta) \tag{26.59}$$

As was done for the first proposed Lagrangian model (see Sect. 26.4.1) we use the subgradient procedure for estimating the values of the Lagrangian multipliers (ρ, η, θ). Hence, after solving $LSP(\rho, \eta, \theta)$ we obtain the column assignment vector $\bar{x}$. Consequently, we can define a subgradient for $L(\rho, \eta, \theta)$ as follows:

$$\psi_i = 1 - \sum_{\omega \in \Omega} (x_\omega \cdot \epsilon_i^\omega) \quad (i \in V) \tag{26.60}$$

$$\vartheta_{ijt} = 1 - \sum_{\omega \in \Omega} (x_\omega \cdot \nu_{ijt}^\omega) \quad (i \in V;\ j \in Succ(i);\ t \in W_{ij}) \tag{26.61}$$

$$\varphi_{kt} = 1 - \sum_{\omega \in \Omega} (x_\omega \cdot \zeta_{kt}^\omega) \quad (k \in \mathscr{R};\ t = 0, \ldots, T) \tag{26.62}$$

Now, given an upper bound T for the makespan and a step size sp, we can update the current Lagrangian multipliers by:

$$\rho_i = \rho_i + (sp \cdot \psi_i) \quad (i \in V) \tag{26.63}$$

$$\eta_{ijt} = \min\{0, \eta_{ijt} + (sp \cdot \vartheta_{ijt})\} \quad (i \in V;\ j \in Succ(i);\ t \in W_{ij}) \tag{26.64}$$

$$\theta_{kt} = \min\{0, \theta_{kt} + (sp \cdot \varphi_{kt})\} \quad (k \in \mathscr{R};\ t = 0, \ldots, T) \tag{26.65}$$

Thereafter, we remind that the step size sp is defined as follows:

$$sp = vp \cdot \frac{(T - L(\rho, \eta, \theta))}{Norm} \tag{26.66}$$

where $Norm$ is given by:

$$Norm = \sum_{i \in V} (\psi_i)^2 + \sum_{i \in V} \sum_{j \in Succ(i)} \sum_{t \in W_{ij}} (\vartheta_{ijt})^2 + \sum_{k \in \mathscr{R}} \sum_{t=0,\ldots,T} (\varphi_{kt})^2 \tag{26.67}$$

As was explained in Sect. 26.4.1 we start the subgradient procedure by setting $vp = 2$. At the end of each iteration we update the parameter vp with a systematic geometric revision: $vp = gp \cdot vp$. Normally the second parameter gp ranges between 0.87 and 0.9995, depending on the targeting problem.

The Lagrangian multipliers are updated during a limited number of iterations. Hence, we use the last updated multipliers as an estimation of the dual multipliers for pricing out new columns. Therefore, as was done in Sect. 26.4.1 for solving RMP_1, when there are no more columns with a negative reduced cost by using the Lagrangian multipliers, we perform a fixed number of iterations solving the RMP_2 with the simplex method by using the solver for obtaining the values of the dual variables for pricing out new columns. Hence, if after the fixed number of iterations there are still columns with negative reduced costs, we go back to the Lagrangian procedure, otherwise, we stop.

26.5 Columns Initialization and Selection

Finally, before reporting the obtained computational experiments, we describe how the subset of columns $\bar{\Omega}$ is initialized for the first CG iteration. In addition, we introduce the procedure developed for selecting the columns that should be included in the pool of activity work patterns in each CG iteration.

26.5.1 Columns Initialization

For the first CG iteration, we initialize the subset of columns $\bar{\Omega}$ for solving the RMP($\bar{\Omega}$) according to a schedule obtained by the tabu search (TS) developed by Bellenguez-Morineau (2008). Nevertheless, it is important to state that a reformulation that includes the utilization of slack variables in the models proposed in Sects. 26.3.1.1 and 26.3.1.2 allows solving the RMP without having an initial set of columns $\bar{\Omega}$. Thereafter, preliminary results show that initializing the pool of columns by means of the TS allows us to prove optimality faster and enhance the possibility of keeping a structure of activity work patterns that could lead to an integer feasible schedule.

26.5.2 Columns Selection

As pointed out previously, exhibiting an activity pattern with global minimal reduced cost requires the enumeration of each activity i for each time instant of its starting domain ($ES_i \leq t_i \leq LS_i$). To limit the number of sub-problems solved at each iteration of the CG procedure, we propose filtering procedures ensuring that the considered pairs (i,t) may lead to columns with a negative reduced cost ($rc_{it} < 0$).

First, according to duality properties, simplex multipliers θ_{kt} associated with constraints (26.4) and (26.14) for RMP_1 and RMP_2, respectively, are less than or equal to zero. Consequently, we have necessarily from (26.10) and (26.19) that $rc_{it}^2 \geq 0$. Thus, the column associated with the pair (i,t) might have a negative reduced cost only if $rc_{it}^1 < 0$. Clearly, the sub-problem $SP(i,t)$ has to be solved only if this condition holds.

Then, assuming that lr_{it}^2 is a lower bound on rc_{it}^2, we get $rc_{it} \geq rc_{it}^1 + lr_{it}^2 = lr_{it}$. Obviously, only sub-problems $SP(i,t)$ for which $lr_{it} < 0$ holds have to be considered since they potentially may lead to activity work patterns with a negative reduced cost. For a given pair (i,t), such a lower bound lr_{it}^2 can be computed by summing the $\sum_{l \in \mathscr{L}} r_{il}$ smallest assignment costs c_{kt} among the workers that master at least one of the required skills to perform i.

Notice that intensive computational experiments reveal that adding several patterns to the current pool of columns at each iteration of the CG procedure leads to better average CPU times. In most cases, the potential increasing in CPU time is counterbalanced by a decrease in the number of iterations needed for ensuring the convergence of the CG process.

26.6 Computational Results

Computational experiments were performed using the solver Gurobi Optimizer version 4.6. We selected a subset of the available instances for the MSPSP (Bellenguez-Morineau and Néron 2005) according to their size in terms of number of activities, skills, and number of workers. In general terms, the computational results shown in this section correspond to instances which contain between 20 and 62 activities, 2 and 15 skills, and 2 and 19 workers. We report results for 271 instances, which are divided into three groups:

- Group 1: We studied 110 instances from this group, containing between 20 and 51 activities, between 2 and 8 skills, and between 5 and 14 workers.
- Group 2: We included the results for 71 instances which contain between 32 and 62 activities, 9 and 15 skills, and 5 and 19 workers.
- Group 3: We studied 90 instances which contain between 22 and 32 activities, 3 and 12 skills, and 4 and 15 workers.

In Table 26.1 we compare the performance of the different column generation and Lagrangian relaxation approaches introduced in the previous sections. On one hand, we evaluated the CG approach based on the RMP_1 (see Sect. 26.3.1.1) and using the simplex method for solving the linear program (LP). For notation purposes, we refer to this last approach as CG_1. In addition, we have also $CGLR_1$, in which the LP for the related RMP_1 is solved with the combined Lagrangian relaxation and column generation approach proposed in Sect. 26.4.2. On the other hand, we have CG_2 and $CGLR_2$, which correspond to the utilization of RMP_2, which is based on the master problem reformulation introduced in Sect. 26.3.1.1. Hence, in CG_2 we use only the simplex method for solving the LP, while in $CGLR_2$ we combine the use of Lagrangian relaxation and the simplex method for solving the LP, as it was proposed in Sect. 26.4.2.

Furthermore, in Table 26.1 we compare the results of the four mentioned approaches in terms of the average deviation $\Delta^{\emptyset}_{LB*}$ against the best known lower bounds (LB^*) obtained by Bellenguez-Morineau and Néron (2005). Subsequently, we also compare the average computation times and the average number of generated columns obtained by each tested model for reaching their respective lower bound. It is important to mention that, for enforcing the quality of the lower bound obtained by applying column generation, we calculated a preliminary lower bound based on the principle of the stable set. This bound was proposed by Mingozzi et al. (1998) and adapted to the MSPSP by Bellenguez-Morineau and Néron (2005).

After analyzing the obtained results we can see that CG_2 and $CGLR_2$ lead to better lower bounds than CG_1 and $CGLR_1$, implying that the solution of

Table 26.1 Comparison between CG approaches proposed

		Group of instances		
		Group 1	Group 2	Group 3
Average deviation against LB^*	CG_1	−10.80 %	−4.96 %	−6.17 %
	$CGLR_1$	−10.80 %	−4.96 %	−6.17 %
	CG_2	−4.31 %	−3.20 %	−3.30 %
	$CGLR_2$	−4.31 %	−3.20 %	−3.30 %
Average CPU time (sec)	CG_1	11.37	9.88	5.10
	$CGLR_1$	7.07	7.97	3.42
	CG_2	216.58	119.82	95.98
	$CGLR_2$	193.15	99.54	87.37
Average number of generated columns	CG_1	738.05	1,181.65	1,817.77
	$CGLR_1$	1,181.19	1,645.07	2,571.80
	CG_2	3,498.45	3,841.40	5,814.11
	$CGLR_2$	9,336.71	10,370.57	11,458.41

RMP_2 indeed enhances a stronger linear relaxation than RMP_1. Nevertheless, the approaches based on the solution of RMP_2 required a considerably larger amount of computation time until obtaining a lower bound. In addition, we can indeed notice that the utilization of Lagrangian relaxation allowed us to accelerate the solution of the respective restricted master problems. Moreover, we can see that the approaches based on the solution of RMP_2 generates more columns (i.e., activity work patterns) per instance than CG_1 and $CGLR_1$. Additionally, we can see that the number of generated columns also increases when using the approach that combines Lagrangian relaxation and the simplex method for solving the LP.

26.7 Conclusions

The main differences between the proposed approaches relies in the master problem formulation and the methods used for solving the related LP. Hence, we were able to conclude that RMP_2 allowed us to obtain better linear relaxations than the ones obtained when using RMP_1. Nevertheless, we could argue that the improvement in the quality of the resulting lower bound after applying the RMP_2 based approaches is not that significant, given the considerable increase in the computation time invested in the solution of each tested instance. Additionally, we were also able to establish that the utilization of the simplex method along with the proposed Lagrangian relaxation models allowed us to decrease the computation time consumed in the solution of each tested instance. Nevertheless, it is important to mention that there are some new perspectives that could be considered regarding to the utilization of column generation for solving the MSPSP. On one hand the generation of certain additional inequalities (cuts) could lead to a stronger linear relaxation when solving the restricted master problem. In addition, regarding the particular performance of the RMP_2 based approaches it could be interesting to take into account certain measures for accelerating the convergence, which could lead to a decrease in the computation times.

References

Artigues C, Demassey S, Néron E, (2008) Resource-constrained project scheduling: models, algorithms, extensions and applications. Wiley, Hoboken

Barahona F, Anbil R (2000) The volume algorithm: producing primal solutions with a subgradient method. Math Program 87:385–399

Barahona F, Jensen D (1998) Plant location with minimum inventory. Math Program 83:101–111

Bard J, Purnomo H (2005) Preference scheduling for nurses using column generation. Eur J Oper Res 164:510–534

Barnhart C, Johnson E, Nemhauser G, Savelsbergh M, Vance P (1998) Branch-and-price: column generation for solving huge integer programs. Oper Res 46:316–329

Beliën J, Demeulemeester E (2007) On the trade-off between staff-decomposed and activity-decomposed column generation for a staff scheduling problem. Ann Oper Res 155:143–166

Bellenguez-Morineau O (2008) Methods to solve multi-skill project scheduling problem. 4OR-Q J Oper Res 6:85–88

Bellenguez-Morineau O, Néron E (2005) Lower bounds for the multi-skill project scheduling problem with hierarchical levels of skills. In: Practice and theory of automated timetabling V. Lecture notes in computer science, vol 3616. Springer, Berlin, pp 229–243

Bertsekas D (1999) Nonlinear programming. Athena Scientific, Belmont

Bixby R, Gregory J, Lustig I, Marsten R, Shanno D (1992) Very large-scale linear programming: a case study in combining interior point and simplex methods. Oper Res 40:885–897

Brucker P, Knust S (2000) A linear programming and constraint propagation-based lower bound for the RCPSP. Eur J Oper Res 127:355–362

Brucker P, Drexl A, Möhring R, Neumann K, Pesch E (1999) Resource-constrained project scheduling: notation, classification, models, and methods. Eur J Oper Res 112:3–41

Busacker R, Gowen P (1961) A procedure for determining a family of minimum-cost-flow patterns. Technical report 15, operations research office, Johns Hopkings University, Baltimore, MD

Chen Z, Powell W (1999) A column generation based decomposition algorithm for a parallel machine just-in-time scheduling problem. Eur J Oper Res 116:220–232

Cordeau J, Laporte G, Pasin F, Ropke S (2010) Scheduling technicians and tasks in a telecommunications company. J Sched 13:1–17

Correia I, Lourenço L, Saldanha-da Gama F (2012) Project scheduling with flexible resources: formulation and inequalities. OR Spectr 34:635–663

Dantzig G, Wolfe P (1960) Decomposition principle for linear programs. Oper Res 8:101–111

Dohn A, Kolind E, Clausen J (2009) The manpower allocation problem with time windows and job-teaming constraints: a branch-and-price approach. Comput Oper Res 36:1145–1157

Fábián C (2000) Bundle-type methods for inexact data. Cent Eur J Oper Res 8:35–55

Fırat M, Hurkens C (2012) An improved MIP-based approach for a multi-skill workforce scheduling problem. J Sched 15:363–380

Gélinas S, Soumis F (2005) Dantzig-Wolfe decomposition for job shop scheduling. In: Desaulniers G, Desrosiers J, Solomon MM (eds) Column generation. Springer, New York, pp 271–302

Gilmore P, Gomory R (1961) A linear programming approach to the cutting-stock problem. Oper Res 9:849–859

Goffin J, Vial J (2002) Convex nondifferentiable optimization: a survey focused on the analytic center cutting plane method. Optim Method Softw 17:805–867

Gutjahr W, Katzensteiner S, Reiter P, Stummer C, Denk M (2008) Competence-driven project portfolio selection, scheduling and staff assignment. Cent Eur J Oper Re 16:281–306

Hartmann S, Briskorn D (2010) A survey of variants and extensions of the resource-constrained project scheduling problem. Eur J Oper Res 207:1–14

Heimerl C, Kolisch R (2010) Scheduling and staffing multiple projects with a multi-skilled workforce. OR Spectr 32:343–368

Held M, Karp R (1971) The traveling-salesman problem and minimum spanning trees: part II. Math Program 1:6–25

Held M, Wolfe P, Crowder H (1974) Validation of subgradient optimization. Math Program 6: 62–88

Huisman D, Jans R, Peeters M, Wagelmans A (2005) Combining column generation and lagrangian relaxation. In: Desaulniers G, Desrosiers J, Solomon MM (eds) Column generation. Springer, New York, pp 247–270

Ioachim I, Desrosiers J, Soumis F, Belanger N (1999) Fleet assignment and routing with schedule synchronization constraints. Eur J Oper Res 119:75–90

Jans R, Degraeve Z (2004) An industrial extension of the discrete lot-sizing and scheduling problem. IIE Trans 36:47–58

Jaumard B, Semet F, Vovor T (1998) A generalized linear programming model for nurse scheduling. Eur J Oper Res 107:1–18

Jiang H, Krishnamoorthy M, Sier D (2004) Staff scheduling and rostering: theory and applications, part I and II. Ann Oper Res 128:1–4

Li H, Womer K (2009) Scheduling projects with multi-skilled personnel by a hybrid MILP/CP benders decomposition algorithm. J Sched 12:281–298

Lübbecke M, Desrosiers J (2005) Selected topics in column generation. Oper Res 53:1007–1023

Mason A, Smith M (1998) A nested column generator for solving rostering problems with integer programming. In: Proceedings of international conference on optimisation: techniques and applications, pp 827–834

Mehrotra A, Murphy K, Trick M (2000) Optimal shift scheduling: a branch-and-price approach. Nav Res Log 47:185–200

Mingozzi A, Maniezzo V, Ricciardelli S, Bianco L (1998) An exact algorithm for the resource-constrained project scheduling problem based on a new mathematical formulation. Manag Sci 44:714–729

Montoya C, Bellenguez-Morineau O, Rivreau D (2013) Branch and price approach for the multi-skill project scheduling problem. Optim Lett. Doi:10.1007/s11590-013-0692-8

Polyak B (1967) A general method of solving extremum problems. Sov Math Doklady 8:593–597

Van den Akker J, Hurkens C, Savelsbergh M (2000) Time-indexed formulations for machine scheduling problems: column generation. INFORMS J Comput 12:111–124

Van den Akker J, Hoogeveen J, de Velde S (2002) Combining column generation and Lagrangian relaxation to solve a single-machine common due date problem. INFORMS J Comput 14: 37–51

Van den Akker J, Diepen G, Hoogeveen J (2007) A column generation based destructive lower bound for resource constrained project scheduling problems. In: Integration of AI and OR techniques in constraint programming for combinatorial optimization problems (CPAIOR 2007). Lecture notes in computer science, vol 4510. Springer, Berlin, pp 376–390

Wolsey L (1998) Integer programming. Wiley, New York

Chapter 27
Benders Decomposition Approach for Project Scheduling with Multi-Purpose Resources

Haitao Li

Abstract Staffing projects often requires both assignment and scheduling decisions to be made, which leads to a computationally demanding large-scale optimization problem. In this chapter, we show that a general class of assignment-type resource-constrained project scheduling problems (RCPSPs) can be handled by a hybrid Benders decomposition (HBD) approach. Our HBD framework extends the classical Benders decomposition method (Benders, Numer Math 4:238–252, 1962) by integrating solution techniques in math programming and constraint programming (CP). Effective cut generation schemes are devised to improve the algorithm performance. Performance of our HBD is demonstrated on a project scheduling problem with multi-skilled workforce. Extensions to the basic HBD framework are discussed.

Keywords Constraint programming • Hybrid Benders decomposition • Multi-skilled personnel • Project scheduling • Project staffing • Resource constraints

27.1 Introduction

In professional service firms, project managers often need to schedule and staff projects with multi-skilled professionals in a time- and cost-effective way. In a shop manufacturing environment, operation managers may need to assign jobs to cross-trained operators and schedule them in the right sequence. R&D projects in the bio-science industry may require research activities to be well-scheduled and assigned with the right technicians to ensure on-time and on-budget project delivery. In the military setting, onboard tasks need to be performed by multi-skilled crew to accomplish a mission.

H. Li (✉)
Department of Logistics and Operations Management, College of Business Administration, University of Missouri, St. Louis, MO, USA.
e-mail: lihait@umsl.edu

C. Schwindt, J. Zimmermann (eds.), *Handbook on Project Management and Scheduling Vol. 1*, International Handbooks on Information Systems,
DOI 10.1007/978-3-319-05443-8_27

The above project scheduling settings share two distinctive features. First, they all involve some *multi-purpose resources* that are flexible to perform different tasks. Examples of multi-purpose resources include multi-skilled personnel due to cross-training (Li and Womer 2006) and flexible machines/equipment that can operate on different jobs (Brucker 2001). Second, they require both assignment and scheduling decisions to be simultaneously made. For instance, one may want to know how to best tradeoff project cost and makespan, and which tasks should be delayed to free up resources for more critical tasks.

Successful completion of projects with such nature requires matching project tasks with the right resources and scheduling them in the right sequence. It is known as multi-skill project scheduling (Chap. 30 of this book and Néron et al. 2006), or project scheduling with multi-purpose resources (PSMPR, Li 2005). In a typical PSMPR setting, a set of project tasks have two aspects of technical constraints: (1) temporal relationships among tasks, e.g. precedence and minimum time lag relationships; and (2) skill/specification requirements for certain resources. The decision-maker's goal is to find a feasible assignment of multi-purpose resources to project tasks and a task sequence, so that some project performance measure is optimized.

The PSMPR is an extension to the single-mode resource-constrained project scheduling problem (RCPSP, Demeulemeester and Herroelen 2002) and belongs to the category of assignment-type RCPSPs (Drexl et al. 1998). Therefore, PSMPR is clearly $\mathscr{N}\mathscr{P}$-complete, as finding a feasible schedule to a single-mode RCPSP is well-known to be $\mathscr{N}\mathscr{P}$-complete (Bartusch et al. 1988).

In this chapter, we focus on a sub-class of PSMPRs where the objective function involves only the assignment decision, but not the scheduling decision. We show that such PSMRPs can be decomposed into an assignment sub-problem and a scheduling sub-problem. Solving the assignment sub-problem along with the objective function provides an upper/lower bound, because the assignment sub-problem is a relaxation of the original problem. Next the obtained assignment solution is checked for feasibility, i.e., whether it can be extended to a feasible solution to the scheduling sub-problem. If it is feasible, an optimal solution to the original problem is found; otherwise, the cause of infeasibility is inferred as "cuts" added to the assignment sub-problem to exclude the assignments that might cause infeasibility. The assignment sub-problem with an augmented set of cut constraints is solved again. This procedure iterates until an optimal solution is found or the original problem is proved to be infeasible.

The main advantage of the above solution framework is that constraints are successively added only when the corresponding cuts are inferred. This is an idea from the classical Benders decomposition method (Benders 1962), which is effective for problems having a large number of constraints, because only a small fraction of them are often binding at an optimal solution (Lasdon 1970).

Another key feature of our algorithm is its ability to incorporate different methods for handling the assignment sub-problem and scheduling sub-problem. For instance, mixed-integer liner programming (MILP, Nemhauser and Wolsey 1988) is usually a good candidate for an assignment problem, but may not be effective for a scheduling problem due to its disjunctive formulation (Balas 1979) with a weak linear relaxation. On the other hand, constraint programming (CP, Marriott and Stuckey 1998), a methodology originated in the artificial intelligence (AI) area, is often able to handle complex scheduling problems well, due to its expressive nature and effective domain reduction techniques for scheduling problems (Baptiste et al. 2001). The complementary strengths of MILP and CP have motivated us to integrate the two in the Benders decomposition framework, giving rise to the so-called hybrid Benders decomposition (HBD) algorithm.

The remainder of the chapter is organized as follows. Section 27.2 describes the basic PSMPR setting and a mathematical programming formulation to model it. Section 27.3 presents the HBD algorithm framework and implementation details. An application example on a multi-skill project scheduling problem is provided in Sect. 27.4 to demonstrate the effectiveness of HBD. Section 27.5 draws conclusions and discusses extensions to the basic HBD framework.

27.2 Optimization Model

This section starts with a description of the basic problem setting of PSMPR, followed by an MILP model formulation. We then highlight several extensions to the basic model.

27.2.1 Problem Description

Consider a project described by a direct graph $G = (V, E)$, where V is a set of activities and E is a set of arcs. Each activity $i \in V$ has a processing time p_i, and must be completed by a deadline $\bar{d}_i$. An arc $(i, j) \in E$ specifies a precedence relationship between activity i and j, i.e. j cannot start until i is completed. Another aspect of technical requirement is given by a three-tuple set (i, l, r_{il}) for $i \in V$ and $l \in \mathscr{L}_i$, where $\mathscr{L}_i$ is the set of skills required by activity i. The set (i, l, r_{il}) means that the execution of activity i requires r_{il} units of resources with skill l. A set $\mathscr{R}$ of multi-purpose resources are available to perform the project. Their skill mix is described by $\mathscr{L}_k$ for $k \in \mathscr{R}$, i.e. resource k possesses a subset $\mathscr{L}_k$ of skills. Symmetrically, the set of resources $\mathscr{R}_l$ that can perform skill l is well-defined, i.e. $\mathscr{R}_l := \{k \in \mathscr{R} | l \in \mathscr{L}_k\}$. A workload capacity WL_k of each resource k cannot be exceeded. A PSMPR attempts to find a feasible assignment and schedule to optimize some project performance metric.

Assumption 1 (Non-preemption). Once an activity $i \in V$ is started it cannot be interrupted.

This is a typical non-preemption assumption in the scheduling literature, which applies in most schedule environment. In the business setting, for example, a project activity is rarely voluntarily interrupted due to disruption cost or learning curve. Military tasks should usually not be interrupted, because an interruption often results in failure of an entire mission.

Assumption 2 (Unary Resource). Each multi-purpose resource $k \in \mathscr{R}$ is assumed to be a *unary resource* that can only operate on one activity/skill at a time

This unary resource assumption applies in most machine scheduling settings, where a machine/equipment/devise/processer can only operate on one job at any point of time. It is also a reasonable assumption for personnel scheduling when the time scale is sufficiently granulate, so that each individual is only able to work on one job at a time.

Assumption 3 (Performance Metric). The project performance metric is a function of only assignment decisions.

Here the decision-maker is mainly concerned about the resource cost required to accomplish a project, which is a function of assignment but not scheduling decision. It can be resource acquisition cost or assignment cost. For a maximization problem, the performance metric can be the reward/value gained through job-resource assignments.

The skill set serves as a link between the activities and resources. A diagram depicting the underlying activity-skill-resource structure of PSMPR can be found in Li and Womer (2006). Note that in general, the skill set can be a set of technical specifications, and the resources may be machines or equipment meeting these specifications. Such flexibility gives PSMPR a wide range of possible applications, as summarized by Table 27.1.

27.2.2 Model Formulation

Our MILP model for PSMPR is built upon the classical big-M formulation for disjunctive scheduling. We define a binary assignment variable $x_{ikl} = 1$ if and only if resource k is assigned to skill l in activity i, for $i \in V$, $l \in \mathscr{L}_i$ and $k \in \mathscr{R}_l$. The start time of activity $i \in V$ is defined as a continuous decision variable $S_i \geq 0$. To properly model the sequencing decisions between a pair of activities $(i, j) \in V \times V$, a binary sequencing decision variable y_{ij} is defined such that $y_{ij} = 1$ if and only if activity i precedes j.

Table 27.1 Diverse applications of PSMPR

Industry	Problem setting	Resources	Objective function
Service	IT and professional service projects	Multi-skilled professionals	Minimize total staffing and assignment costs
Manufacturing	Make-to-order (MTO) jobs and customer orders	Multi-purpose machines/ equipment	Minimize production or supply chain costs
Construction	Construction project scheduling	Multi-purpose equipment and workers	Minimize total construction costs
Bio-science	R&D project portfolio optimization	Multi-purpose technicians and equipment	Maximize total R&D portfolio return
Airline	Service operations scheduling	Technicians and staff	Minimize total operating costs
Health care	Operating room scheduling	Equipment and nurses	Minimize total operating cost
Military	Mission planning and scheduling	Multi-skilled crew	Minimize crew size or maximize readiness

The MILP formulation of the basic version of PSMPR can be written as:

PSMPR_MILP{

$$\text{Min. } f(x) \tag{27.1}$$

s.t.

$$\sum\nolimits_{k\in\mathscr{R}_l} x_{ikl} = r_{il} \qquad (i \in V; l \in \mathscr{L}_i) \tag{27.2}$$

$$\sum\nolimits_{l\in\mathscr{L}_i} x_{ikl} \le 1 \qquad (i \in V; k \in \mathscr{R}) \tag{27.3}$$

$$\sum\nolimits_{i\in V}\sum\nolimits_{l\in\mathscr{L}_i} p_i x_{ikl} \le WL_k \qquad (k \in \mathscr{R}) \tag{27.4}$$

$$S_j - S_i \ge p_i \qquad ((i,j) \in V \times V) \tag{27.5}$$

$$S_i + p_i \le \bar{d}_i \qquad (i \in V) \tag{27.6}$$

$$y_{ij} + y_{ji} \ge x_{ikl} + x_{ikl'} - 1 \qquad (\text{ordered } (il, jl');\ k \in \mathscr{R}) \tag{27.7}$$

$$S_j \ge S_i + p_i - M\left(1 - y_{ij}\right) \qquad ((i,j) \in V \times V) \tag{27.8}$$

$$x_{ikl}, y_{ij} \in \{0,1\}; S_i \ge 0 \tag{27.9}$$

}

The objective function (27.1) minimizes some performance metric as a function $f(\cdot)$ of the vector x of assignment decision variables. It is not affected by the scheduling decisions (Assumption 3). For instance, given the cost of assigning a resource to an activity, one may minimize the total assignment cost for the project;

one may also minimize the resource acquiring cost as in Li and Womer (2009a), or minimize the cardinality of selected resource set (Li and Womer 2009b).

Constraint (27.2) assigns activity i with r_{il} units of resources that have skill l required by i. Constraint (27.3) forbids assigning more than one skill in an activity to a resource unit, because each resource can only perform one skill at a time (Assumption 2). Constraint (27.4) ensures that the total workload assigned to a resource cannot exceed its available capacity. The precedence constraint between activity i and j is satisfied through Constraint (27.5). Constraint (27.6) guarantees that each activity is completed before its deadline. Constraint (27.7) states that if two activities/skills are assigned with the same resource, the two activities cannot overlap, i.e. one sequencing relation must be determined. Constraint (27.8) is the big-M formulation to specify the sequencing relation between activity i and j: if i precedes j $(y_{ij} = 1)$, j can only start after i is completed.

27.2.3 Variations and Extensions

We highlight some variations and extensions of the basic model to expand the modeling capability of PSMPR.

- *General temporal constraint*: The precedence graph $G = (V, E)$ can be generalized to a weighted directed graph $N = (V, E, \delta)$ with δ being the set of arc weights, representing some time lags between a pair of activities (Neumann et al. 2002). For instance, a minimum time lag d_{ij}^{min} between activity i and j requires that j cannot start until at least d_{ij}^{min} time units after i is started. Clearly, when $d_{ij}^{min} = p_i$ such minimum time lag constraint reduces to a precedence constraint.
- *Discrete renewable resource*: Rather than being treated as unary resource, a resource unit can in general be a discrete renewable resource to operate on more than one activity. For instance, a machine center may have identical parallel machines that can be assigned with multiple activities at one time; a consultant may be assigned to multiple tasks per time interval (e.g. week, month, etc.).
- *Unknown skill mix of resources*: The resources' skill mix $\mathscr{L}_k$ or $\mathscr{R}_l$ may not be known, but can be modeled as decision and be optimized. This extension is useful to design a team of workforce for accomplishing a project. Li and Womer (2009b) present an application to optimize the crew size and skill-mix to man a military ship.

27.3 Hybrid Benders Decomposition

The inefficiency of directly solving the MILP formulation (27.1) through (27.9) is mainly due to the big-M formulation in (27.8), which is known to have a loose linear relaxation. In addition, the number of sequencing constraints (27.8) and (27.9) grows rapidly with problem size. Recall that the objective function (27.1) involves only the assignment decision variables. These observations have motivated us to decompose the original problem into an assignment sub-problem and a scheduling sub-problem, which can be handled by appropriate methods separately. In particular, we employ techniques in constraint programming (CP) for dealing with the scheduling sub-problem.

In this section, we start with a brief introduction to constraint programming (CP), then introduce the algorithm framework of hybrid Benders decomposition (HBD) and its implementation details for PSMPR.

27.3.1 Constraint Programming

Constraint programming (CP) originated in the artificial intelligence (AI) field for solving constraint satisfaction problems (CSP, Tsang 1993). It employs reasoning and deduction methods, called constraint propagation or domain reduction, to iteratively reduce the domain of each decision variable. Efficient constraint propagation techniques are available for dealing with scheduling problems (Baptiste et al. 2001). CP also relies on various search procedures, e.g. depth-first (DF) and best-first (BF), to explore the reduced solution domains.

The complementary strengths between CP and mathematical programming have motivated the design of various hybrid algorithms to integrate the two for solving $\mathcal{NP}$-hard combinatorial optimization problems (Hooker 2002). Existing integration schemes include branch-and-infer (Bockmayr and Kasper 1998), branch-and-price (Easton et al. 2004), Lagrangian relaxation (Benoist et al. 2001; Sellmann and Fahle 2003), and Benders decomposition (Eremin and Wallace 2001; Benoist et al. 2002).

27.3.2 HBD Framework

The hybrid Benders decomposition (HBD) was first proposed by Jain and Grossmann (2001) to solve a class of machine scheduling problems, known as open-shop multiple-purpose machine scheduling or OMPM (Brucker 2001). OMPM can be viewed as a special case of PSMPR when temporal constraints are limited to release- and due-dates, but no time dependency among activities is considered.

In the HBD framework, the original problem is decomposed into a relaxed master problem (RMP) containing only assignment decision variables and constraints, and a feasibility sub-problem (SP) modeling the scheduling decision. An optimal assignment solution to the RMP provides a lower bound to the original problem (with a minimization objective function). Then the SP is solved, while fixing the RMP assignment solution, to check whether the RMP solution can be extended to a feasible schedule. If so, an optimal solution is found; otherwise, causes to infeasibility are inferred as *Benders cuts* to be added back to the RMP. The role of Benders cuts is to exclude assignment solutions that will lead to an infeasible schedule. Next the augmented RMP with cuts added is solved again. The process iterates until an optimal solution is found or infeasibility can be proved. There are several keys for the success of the HBD algorithm:

- The RMP without scheduling decision variables and constraints is easier to solve than the original problem.
- There must be effective methods for solving the scheduling SP. Since one only needs to deduce whether the SP is feasible or not, as in a constraint satisfaction problem (CSP), it naturally calls for CP to handle the SP.
- Benders cuts must be inferred to effectively exclude infeasible assignment solutions.

While the idea of successively adding cuts to tighten the RMP shares similarity with the classical Benders decomposition method (BDM), the HBD framework differs from the classical BDM and other CP-based Benders decomposition methods (Eremin and Wallace 2001; Benoist et al. 2002) in the way the cuts are generated. In the classical BDM, both *feasibility cuts* and *optimality cuts* are inferred by solving the *dual* of the sub-problem. We refer to Martin (1999) for an updated treatment on the classical BDM. In the HBD framework, however, only feasibility cuts are generated because solution to the SP does not affect the objective function (Assumption 3). This has facilitated the use of CP to directly model the *primal* SP and to deduce its feasibility.

27.3.3 HBD for PSMPR

The MILP formulation (27.1)–(27.9) is decomposed into an RMP with only assignment decision variables, and a scheduling sub-problem with only the scheduling decision variables. The formulation of RMP at the μ-th iteration of the HBD algorithm can be written as:

RMP_MILP(μ){

$$\text{Min. } f(x) \qquad (27.1)$$

s.t.

$$\sum\nolimits_{k \in \mathscr{R}_l} x_{ikl} = r_{il} \qquad (i \in V; l \in \mathscr{L}_i) \qquad (27.2)$$

$$\sum\nolimits_{l \in \mathscr{L}_i} x_{ikl} \le 1 \qquad (i \in V; k \in \mathscr{R}) \qquad (27.3)$$

$$\sum\nolimits_{i \in V} \sum\nolimits_{l \in \mathscr{L}_i} p_i x_{ikl} \le WL_k \qquad (k \in \mathscr{R}) \qquad (27.4)$$

$$\beta_\lambda(x) \le 1 \qquad (\lambda = 1, \ldots, \mu) \qquad (27.9)$$

}

The objective function (27.1) and Constraints (27.2)–(27.4) are directly taken from the PSMPR_MILP formulation. Constraints (27.9) include cuts generated at each iteration $\lambda = 1, \dots, \mu$. Each cut prevents certain assignments to avoid infeasibility of the scheduling SP. Details on how to generate cuts will be elaborated later in this section. Evidently, without a large number of scheduling decision variables and sequencing constraints, RMP_MILP is easier to solve than PSMPR_MILP.

For a solution x^{μ} to the RMP_MILP(μ) at iteration μ, we construct a scheduling sub-problem SP(x^{μ}) with the assignment decisions fixed at x^{μ}. The goal is to check whether x^{μ} will result in a feasible schedule that satisfies Constraints (27.5)–(27.8). We use CP to model the SP for the following reasons: (1) the expressive nature of CP will eliminate the need for big-M formulation, thus significantly reduce the size of the model; (2) one only needs to find whether SP(x^{μ}) is feasible or not, which is suitable for CP-based methods; (3) we want to take advantage of the available effective constraint propagation techniques for scheduling problems.

To facilitate presentation of the CP model, we use the language constructs in OPL, a modeling language developed by Van Hentenryck (1999). We define an array of *unary resources* named MultipurposeRes as the resource set $\mathscr{R}$. Then the CP formulation of the feasibility SP at iteration n can be written as:

SP_CP(x^{μ}){

$$\text{Activity } i \ \textit{precedes}\, j, \qquad ((i,j) \in V \times V) \tag{27.10}$$

$$S_i + p_i \leq \bar{d}_i, \qquad (i \in V) \tag{27.11}$$

$$x_{ikl}^{\lambda} = 1 \Rightarrow \text{Activity } i \ \textit{requires} \text{ MultipurposeRes}[k], \qquad (\lambda = 1, \dots, \mu; i \in V; l \in \mathscr{L}_i; k \in \mathscr{R}_l) \tag{27.12}$$

}

Note that no objective function is present here, as one only needs to find out if the SP is feasible or not. Constraints (27.10) and (27.11) are equivalent to Constraints (27.5) and (27.6) in PSMPR_MILP, respectively. Constraint (27.12) is the key in our decomposition scheme, which establishes the link between the RMP and SP. Specifically, if resource k has been assigned to skill l in activity i at iteration λ of the algorithm, we enforce that activity i must *require* resource k when scheduling the activities. In this way, we eliminate the need for the large number of Constraints (27.7) and (27.8) in the big-M formulation of PSMPR_MILP, which has significantly reduced model size.

The pseudo code of HBD algorithm to solve PSMPR is presented below.

```
Step 0. Initialization: Set $\mu := 1$, $contin \coloneqq true$
Step 1. Construct the RMP_MILP model for the relaxed master problem.
Step 2. The main HBD iteration. **While** $contin = true$:
    2.1. Solve the RMP_MILP to get an optimal assignment solution $x^{\mu}$
    2.2. **If** RMP_MILP is infeasible, **then**
             The original PSMPR is infeasible. Set $contin \coloneqq false$.
         **Else**
             Proceed to Step 2.3.
    2.3. Construct/update the sub-problem model SP_CP($x^{\mu}$) while fixing assignment
         solutions to $x^{\mu}$.
    2.4. **Solve** a feasibility problem to check whether SP_CP($x^{\mu}$) is feasible.
    2.5. **If** SP_CP($x^{\mu}$) is feasible, **then**
             $x^{\mu}$ is optimal to the original PSMPR. Set $contin \coloneqq false$.
         **Else**
             Proceed to Step 2.6.
    2.6. Infer a valid cut $\beta_{\mu}(x) \leq 1$ and add it back to RMP_MILP.
    2.7. Increment $\mu \coloneqq \mu + 1$.
```

Fig. 27.1 Pseudo code of the HBD algorithm for solving PSMPR

The SP_CP feasibility model in Step 2.4 can be effectively solved by CP based methods for several reasons. First, the original assignment-type RCPSP has now reduced to a single-mode RCPSP with unary resources. Second, effective constraint propagation algorithms (Baptiste et al. 2001) are available for scheduling problems with unary resources in (27.12) and *binary constraints* in (27.10) and (27.11), where exactly two decision variables are involved in each constraint. Finally, CP is often effective for findings feasible solution.

For the RMP_MILP in Step 2.1 to be computationally tractable, we need the following assumption.

Assumption 4 (Convexity of Objective Function). The objective function $f(x)$ is a convex function of the assignment decision variables x, when the integral requirement is relaxed.

Any linear combination of x, such as those considered in Li and Womer (2009a) and Li and Womer (2009b) will satisfy Assumption 4. Convexity of $f(x)$ makes it possible for various MILP methods (Nemhauser and Wolsey 1988) to find and prove optima to RMP_MILP. An important convergence property of the HBD algorithm also follows.

Proposition 27.1 (Finite Convergence of HBD). *If all the cuts added in Constraint (27.9) are valid cuts, the HBD algorithm converges to an optimal solution or proves infeasibility in a finite number of iterations.*

Proof. We consider two cases.

1. The original PSMPR_MILP is feasible. As the algorithm iteration μ increases, the solution space of master problem RMP_MILP (μ) is reduced. Since the domain of x is finite, there exists an iteration θ when the optimal solution x^θ to RMP_MILP (θ) leads to a feasible schedule S^θ to the sub-problem SP_CP(x^θ). Thus $f\left(x^\theta\right)$ is an upper bound to PSMPR_MILP. Since a valid cut never excludes any feasible solution, the master problem is a relaxation of the original problem. This implies that $f\left(x^\theta\right)$ is also a lower bound to the original problem. Therefore, x^θ is an optimal solution.
2. The original PSMPR_MILP is infeasible. In this case, any optimal solution to the master problem will never lead to a feasible schedule to the sub-problem. Since the domain of x is finite, the master problem will become infeasible after a finite number of iterations. □

27.3.4 Cut Generation

The finite convergence is a nice property of the HBD algorithm. However, it says nothing about the algorithm efficiency. The performance of HBD depends largely on whether effective cuts can be generated. We now introduce several schemes to generate cuts in Step 2.6 of the algorithm.

No-Good Cuts

The idea of no-good cuts, as suggested by Hooker (2000), is to negate the specific assignment solution. That is, if the scheduling sub-problem SP_CP(x^μ) associated with the solution x^μ at iteration μ is infeasible, at least one assignment in x^μ must not be made. A no-good cut thus takes the following form:

$$\sum\nolimits_{i\in V}\sum\nolimits_{l\in\mathscr{L}_i}\sum\nolimits_{k\in\mathscr{R}_l} x_{ikl}\,|(x_{ikl}=1)\leq\sum\nolimits_{i\in V}|\mathscr{L}_i| \tag{27.13}$$

The no-good cut (27.13) is clearly a valid cut, but might be too loose because it only negates x^μ without excluding other potential infeasible solutions. Thus relying only on no-good cuts alone may not be efficient.

Temporal Analysis Based Cuts

More effective cuts can be generated by exploiting the problem structure. Let's start with some reasoning on why an assignment solution might cause the scheduling sub-problem to be infeasible. Consider a pair of activities (i, j) satisfying all the temporal constraints. Their relationship might be described by three possible cases:

(i) i and j must overlap, which is denoted by $i \parallel j$; (ii) i and j are possible to overlap denoted by $i|j$; and (iii) i and j never overlap denoted by $i \sim j$.

Now suppose both i and j are assigned with the same resource k, i.e., $i \rightarrow k$ & $j \rightarrow k$. Recall from Assumption 2 that resource k can only work on one activity at a time. Therefore, for $i \rightarrow k$ & $j \rightarrow k$ to be feasible, i and j must never overlap (Case iii).

However, infeasibility occurs when Case (i) is in fact true. To resolve infeasibility, a *global cut* can be inferred to prevent both i and j being assigned to k, which takes the form:

$$\sum\nolimits_{l \in \mathscr{L}_i} x_{ikl} + \sum\nolimits_{l \in \mathscr{L}_j} x_{ikl} \leq 1, \tag{27.14}$$

where $i \parallel j$. It is a valid cut and is tighter than the no-good cut in (27.13) because it excludes a set of assignment solutions sharing some common features, rather than just a specific assignment solution.

When Case (ii) is true, a *trial cut* can be generated to resolve infeasibility:

$$\sum\nolimits_{l \in \mathscr{L}_i} x_{ikl} + \sum\nolimits_{l \in \mathscr{L}_j} x_{ikl} \leq 1, \tag{27.15}$$

where $i|j$. A trial cut is not a valid cut, because it may exclude a feasible solution.

Our cut generating procedure starts with a temporal analysis to find the time window, i.e. the earliest start and latest start time, of each activity. It then deduces the relationship between every pair of activities to obtain a valid cut set and a non-valid cut set. Only those activated cuts are added into the RMP_MILP in the HBD algorithm of Fig. 27.1. Caution should be made that optimality can no longer be proved whenever a trial cut is added. We refer to Li and Womer (2009a) for details of the cut generating procedure and properties of the cut sets.

27.4 Application Example

This section presents an application of the HBD algorithm for solving an instance of PSMPR with multi-skilled workforce. The problem setting follows that described in Sect. 27.2, with V being a set of tasks in a project, $\mathscr{L}$ being the set of skills required to execute the tasks, and $\mathscr{R}$ representing the set of multi-skilled individuals. r_{il} is assumed to be one, meaning that an activity i requires only one unit of resource for skill $l \in \mathscr{L}_i$. The objective is to minimize the total resource acquiring cost for accomplishing a project. Let c_k denote the acquiring cost of resource $k \in \mathscr{R}$, and define a binary decision variable $z_k = 1$ if and only if resource k is selected/acquired. Then the objective function $f(\cdot)$ takes the form: $\sum_{k \in \mathscr{R}} c_k z_k$. The new decision variable z is linked with x through a revised formulation of Constraint (27.4):

Table 27.2 Performance of HBD algorithm compared with MILP and CP methods

	HBD algorithm	MILP method	CP method
$p_{feas}(\%)$	100.00	100.00	93.65
$p'_{opt}(\%)$	84.13	83.33	54.76
$p_{opt}(\%)$	80.95	69.05	23.02
$t^{\emptyset}_{cpu}(\mathrm{s})$	418.01	1,672.91	2,734.14

$$\sum\nolimits_{i\in V}\sum\nolimits_{l\in\mathscr{L}_i} p_i x_{ikl} \le WL_k z_k \qquad (k\in\mathscr{R}) \tag{27.16}$$

The new Constraint (27.16) states that a resource k can be assigned to activities only when k is selected. We refer to Li and Womer (2009a) for a complete MILP formulation and a numerical example of the application.

Our HBD algorithm with both the global cuts (27.14) and trial cuts (27.15) is implemented to solve test instances with size up to 30 tasks, eight skill types and 90 individuals. Other data are generated in our computational experiment to control restrictiveness of the project network (Thesen 1977), project deadline, average number of skills required by a task, and average number of skills possessed by an individual. We show performance of HBD compared with the pure MILP and CP methods. A time limit of 10 h is imposed for each MILP and CP run. For the HBD, a limit of 10 s is imposed for solving SP_CP, and 600 s for solving RMP_MILP. Table 27.2 summarizes the results: the percentage of instances for which a feasible solution was found (p_{feas}), percentage of instances for which an optimal solution was found and proved (p_{opt}), percentage of instances for which an optimal solution was found but not proved (p'_{opt}), and the average of computational time $t^{\emptyset}_{cpu}$.

It is evident that the HBD algorithm outperforms either the pure MILP method or the pure CP method alone in both solution quality and speed. Both HBD and MILP are able to find feasible solutions to all the test instances, whereas the pure CP method fails to find feasible solutions to some instances in 10 h. HBD finds about the same number of optimal solutions as MILP does, which outperforms CP. Notably, HBD is able to prove more optimality than MILP does. It also spends significantly less time than MILP or CP. Additional analysis and discussions of the computational results are available in Li and Womer (2009a).

A typical PSMPR problem has both assignment and scheduling components. It is difficult for CP alone to find optimal (or near-optimal) solutions by searching through a mixture of large and complex solution space. The branch-and-bound method in MILP also becomes less effective due to loose linear relaxation on the big-M sequencing formulation. Our HBD framework makes it possible for MILP to deal with only the assignment component of PSMPR, while leaving the scheduling component to be handled by CP.

27.5 Conclusions

We study a general class of assignment-type RCPSPs, called project scheduling with multi-purpose resources (PSMPR), where a resource entity is versatile enough to perform tasks requiring different skills or specifications. We show that when the objective function involves only assignment decision variables, PSMPR can be effectively solved by a hybrid Benders decomposition (HBD) algorithm. In the HBD framework, the original PSMPR is decomposed into a relaxed master assignment problem (RMP) and a feasibility scheduling sub-problem (SP), which can be handled by mix-integer linear programming (MILP) and constraint programming (CP) methods, respectively. The two sub-problems are linked by Benders cuts to successively exclude infeasible assignment solutions. Computational results show that the HBD algorithm outperforms either pure MILP or CP approach alone in solution quality and speed.

There are several important extensions to the basic HBD framework presented in this chapter. First, recall that the convexity assumption in Assumption 4 does not exclude the possibility for objective function $f(x)$ to be nonlinear. There are a large suite of mixed-integer nonlinear programming (MINLP) methods that often work quite well for problems with convex objective function (Floudas 1995). An example of such extension is available in Chap. 55 in the second volume of this handbook and Li and Womer (2012), where the RMP is an MINLP with a nonlinear convex objective function. Second, when it becomes computationally too demanding for integer programming to find and prove optima to the RMP, one may seek to find near-optimal RMP solution using some metaheuristics (Glover and Kochenberger 2005) that fit best for the problem at hand.

References

Balas E (1979) Disjunctive programming. Ann Discrete Math 5:3–51

Baptiste P, Le Pape C, Nuijten W (2001) Constraint-based scheduling: applying constraint programming to scheduling problems. Springer, New York

Bartusch M, Mohring RH, Randermacher FJ (1988) Scheduling project networks with resource constraints and time windows. Ann Oper Res 16(1):201–240

Benders JF (1962) Partition procedures for solving mixed variables programming problems. Numer Math 4:238–252

Benoist T, Gaudin E, Rotternbourg B (2002) Constraint programming contribution to Benders decomposition: A case study. In: Principles and practice of constraint programming (CP 2002). Lecture notes in computer science, vol 2470. Springer, Berlin, pp 603–617

Benoist T, Laburthe F, Rottembourg B (2001) Lagrange relaxation and constraint programming collaborative schemes for traveling tournament problems. In: Gerwet C, Wallace M (eds) Proceedings of the third international workshop on integration of AI and OR techniques in constraint programming for combinatorial optimization problems (CPAIOR 2001), pp 15–26

Bockmayr A, Kasper T (1998) A unifying framework for integer and finite domain constraint programming. INFORMS J Comput 10(3):287–300

Brucker P (2001) Scheduling algorithms. Springer, Berlin

Demeulemeester EL, Herroelen WS (2002) Project scheduling: a research handbook. International series in operations research and management science. Kluwer Academic, Boston

Drexl A, Juretzka J, Salewski F, Schirmer A (1998) New modeling concepts and their impact on resource-constrained project scheduling. In: Węglarz J (ed) Project scheduling: recent models, algorithms and applications. Kluwer Academic, Dordrecht, pp 413–432

Easton K, Nemhauser G, Trick M (2004) CP based branch-and-price. In: Milano M (ed) Constraint and integer programming. Operations research/computer science interfaces series, vol 27. Kluwer Academic, Dordrecht, pp 207–231

Eremin A, Wallace M (2001) Hybrid Benders decomposition algorithms in constraint logic programming. In: Walsh T (ed) Principles and practice of constraint programming (CP 2001). Lecture notes of computer science, vol 2239. Springer, Berlin, pp 1–15

Floudas CA (1995) Nonlinear and mixed-integer optimization: fundamentals and applications. Oxford University Press, New York

Glover F, Kochenberger G (2005) Handbook of metaheuristics. Springer, Berlin

Hooker J (2000) Logic-based methods for optimization: combining optimization and constraint satisfaction. Wiley, New York

Hooker J (2002) Logic, optimization and constraint programming. INFORMS J Comput 14(4):285–321

Jain V, Grossmann I (2001) Algorithms for hybrid milp/cp models for a class of optimization problems. INFORMS J Comput 13(4):258–276

Lasdon LS (1970) Optimization theory for large systems. Macmillan, New York

Li H (2005) Project scheduling with multi-purpose resources: models, algorithms and applications. Ph.D. dissertation, The University of Mississippi

Li H, Womer K (2006) Project scheduling with multi-purpose resources: a combined MILP/CP decomposition approach. Int J Oper Quant Manag 12(4):305–325

Li H, Womer K (2009a) Scheduling projects with multi-skilled personnel by a hybrid MILP/CP Benders decomposition algorithm. J Sched 12(3):281–298

Li H, Womer K (2009b) A decomposition approach for shipboard manpower scheduling. Mil Oper Res 14(3):1–24

Li H, Womer K (2012) Optimizing the supply chain configuration for make-to-order manufacturing. Eur J Oper Res 221(1):118–128

Marriott K, Stuckey PJ (1998) Programming with constraints. MIT Press, Cambridge

Martin RK (1999) Large scale linear and integer optimization: a unified approach. Kluwer Academic, Norwell

Nemhauser G, Wolsey L (1988) Integer and combinatorial optimization. Wiley, New York

Néron E, Bellenguez O, Heurtebise M (2006) Decomposition method for solving multi-skill project scheduling problem. In: Proceedings of the tenth international workshop on project management and scheduling, Poznan, pp 265–269

Neumann K, Schwindt C, Zimmermann J (2002) Project scheduling with time windows and scarce resources: temporal and resource-constrained project scheduling with regular and nonregular objective functions. Springer, Berlin

Sellmann M, Fahle T (2003) Constraint programming based Lagrangian relaxation for the automatic recording problem. Ann Oper Res 118:17–33

Thesen A (1977) Measures of the restrictiveness of project networks. Networks 7(3):193–208

Tsang E (1993) Foundations of constraint satisfaction. Academic Press, London

Van Hentenryck P (1999) The OPL optimization programming language. MIT Press, Cambridge

Chapter 28
Mixed-Integer Linear Programming Formulation and Priority-Rule Methods for a Preemptive Project Staffing and Scheduling Problem

Cheikh Dhib, Ameur Soukhal, and Emmanuel Néron

Abstract This chapter presents a generic model for an industrial project scheduling problem. The problem addressed here is an extension of the Resource-Constrained Project Scheduling Problem (RCPSP) and the Multi-Skill Project Scheduling Problem (MSPSP). The main specificities of this problem are the following: We considered both preemptive activities and non-preemptive activities, resource requirements of activities are given in terms of skills, and different durations exist in terms of both activities and skills. This model and its resolution methods are to be used in the Apache Open For Business (OFBiz) open source Enterprise Resource Planning (ERP) system, and must therefore satisfy some industrial constraints. We first propose a general model for this problem. Then, we propose a Mixed Integer Linear Program (MIP) formulation and a heuristic algorithm based on priority rules. The originality of the model lies in the fact that it simultaneously considers skill synchronization, preemption and precedence relationships. Experimental results performed on adapted instances from the PSPLIB benchmark are provided.

Keywords Linear programming • Multi-Skill scheduling • Preemptive scheduling • Priority-Rule heuristics • Project scheduling

28.1 Introduction

Project scheduling problems are among the most studied scheduling problems in the literature. Resource-Constrained Project Scheduling Problem (RCPSP) is the most classic of these problems. In the RCPSP, a set of non-preemptive activities has

C. Dhib (✉) • A. Soukhal • E. Néron
Laboratory of Computer Science, Team Scheduling and Control (ERL CNRS 6305),
University François Rabelais, Tours, France
e-mail: cheikh.dhib@etu.univ-tours.fr; ameur.soukhal@univ-tours.fr;
emmanuel.neron@univ-tours.fr

C. Schwindt, J. Zimmermann (eds.), *Handbook on Project Management and Scheduling Vol. 1*, International Handbooks on Information Systems,
DOI 10.1007/978-3-319-05443-8_28

to be processed. Activities require a given amount of each resource to be processed and are subject to classical end-to-start precedence relationships. Resources are limited. This problem is known to be $\mathcal{NP}$-Hard (Błażewicz et al. 1983) and several state-of-the-art methods dealing with RCPSP can be found in Demeulemeester and Herroelen (1997) and Artigues et al. (2008) (see Chaps. 1, 2, 3, and 4).

Resource modeling has been one fruitful research direction for new project scheduling models. Resources can be renewable, non-renewable, or doubly constrained. Moreover, resource requirements of activities may differ from one mode to another. These types of resources are modeled in the Multi-Mode Resource Constrained Project Scheduling Problem (MRCPSP) (Bouleimen and Lecocq 2003; Sprecher and Drexl 1998; Hartmann and Drexl 1998). Several methods for solving MRCPSP have been proposed, including exact methods such as branch-and-bound, and heuristics (see Chap. 21). Recently, authors have proposed to enlarge the RCPSP model to take into account the notion of skills, i.e., staff members involved in the project realization can contribute only to a given subset of activities. This is known as the Multi-Skill Project Scheduling Problem (Bellenguez-Morineau and Néron 2004b, 2007). Resources considered are human resources, i.e., staff members, each of them able to perform more than one type of activity. This model is useful in several industrial contexts and, for instance, in the context of IT companies where human resources are the most constrained resources. Moreover, availability periods are considered for staff members. This extension can be seen as a special case of Multi-Mode RCPSP, but potentially having a huge number of modes per activity as revealed by Bellenguez-Morineau and Néron (2007). Recently, several studies focus on the notion of skill, in a context of project scheduling (see Correia et al. 2010; Heimerl and Kolisch 2010; Li and Womer 2009; Santos and Tereso 2010; Valls et al. 2009; Walter and Zimmermann 2010; Gürbüz 2010).

In spite of these studies, we find that these models may not be sufficient in an industrial context. They need to be complemented by some specific constraints, such as the synchronization of activity skills or the ability to preempt some activities. These constraints are introduced to meet a particular need of the *Neréide* company, but they can be found at all software service companies and more generally in production service companies. The aim of this study is therefore to propose a multi-skill project scheduling model arising in an industrial context. Therefore, proposed solution methods have to be efficient in terms of both computation time and solution quality. So, the industrial context underlying this paper is slightly different than the ones addressed before. This work is a joint work with a company contributing to an open-source ERP framework. We have defined a model for project scheduling according to their need, which includes some specificity encountered with the context of project scheduling embedded within a generic framework.
The main characteristics of this model are:

- *Activities can be either preemptive or non-preemptive*. This is a key point of the model. On most real-life projects, some activities can be interrupted without any penalties (such as writing documentation, archiving etc.), whereas some activities cannot be interrupted.

- In the case of preemption, *the resource assigned to one skill for one activity must be the one that completes the activity after the preemption*. This constraint is used to limit the preemption effect on the project organization, thus we do not consider non-resumable or semi-resumable activities that would be implied by the fact that a person has to redo a part of the activity.
- *Exactly one resource is required for each skill for one activity*. This is mainly due to the management of the project. Moreover, in the case where several resources can contribute to the same skill, the problem of workload estimation becomes difficult. Notice that this constraint is relevant in the context of small- and medium- size projects.
- *All parts of one activity, each corresponding to one skill required for this activity, must start simultaneously, but can be preempted and restarted at different time points.* This is what we call a synchronization constraint that corresponds to a short period needed to brief the people contributing to this activity.

The problem presented in this chapter is arising in an industrial context while, in Chap. 25, a modeling framework is addressed, where mixed-integer linear programming formulations are presented to solve project staffing and scheduling issues. Dealing with MSPSP problem, other resolution techniques are presented in Chaps. 26 and 27. In Chap. 26, column generation and Lagrangian relaxation are proposed, when Chap. 27 is dedicated to a Benders decomposition approach.

The reminder of this chapter is organized as follows. In Sect. 28.2, a formal description and MIP formulation are presented, then we give a detailed description of the proposed heuristic algorithm in Sect. 28.3. In Sect. 28.4, we present computational results and in Sect. 28.5, we conclude this chapter by pointing out further research directions.

28.2 Problem Description and Model

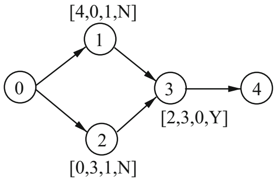
First of all we will propose a conceptual optimization model for the studied problem. Then a MIP formulation will be described. Preliminary results of the use of this formulation where presented in Dhib et al. (2011b), although further results will be presented in Sect. 28.4.2. This formulation is mainly used to establish the problem and to model the constraints we have to consider. Unfortunately, this model cannot be used to obtain relevant lower bounds or to build efficient solutions even for medium-size instances. Despite this negative result, some recent works focusing on MIP formulation for the MSPSP may be a future research direction.

28.2.1 Problem Description

$V = \{0, 1, \ldots, n, n+1\}$ denotes the set of activities. Activities 0 and $n+1$ are the starting and the ending dummy activities, respectively. These activities are subject to precedence relationships. $G = (V, E)$ is the precedence graph with $E = \{(i, j) \in V \times V\}$. If $j \in Succ(i)$, then j cannot start before the end of i. Activities can be either preemptive or non-preemptive: $V^p \subseteq V$ denotes the set of preemptive activities, $V^{np} \subseteq V$ is the set of non-preemptive activities, where $V^p \cup V^{np} = V$.

Let $\mathscr{R} = \{1, \ldots, K\}$ be the set of staff members and $\mathscr{L} = \{1, \ldots, L\}$ be the set of skills. Then, for all $i \in V$, $l \in \mathscr{L}$, p_{il} is the workload of skill l for activity i. According to the industrial considerations, we assume that exactly one person is assigned for one skill requirement. Thus p_{il} becomes a duration. Staff members are able to process a given subset of skills. For all $k \in \mathscr{R}$, $l \in \mathscr{L}$, $s_{kl} = 0$ if resource k does not possess skill l, otherwise $s_{kl} = 1$. Finally, for all $k \in \mathscr{R}$, $t = 1, \ldots, T$, $a_{kt} = 1$ if person k is available on the interval $[t, t + 1[$. We use also a_{klt} which is a binary value equal to 1 if person k masters skill l and is available at t that is deduced from both s_{kl} and a_{kt}. We use p_i, ES_i and d_i to note respectively the activity duration, earliest start time and due date of activity i, where $p_i = \max_{l \in \mathscr{L}}(p_{il})$.

In Fig. 28.1, we present an example of a project with three activities (1, 2, and 3) and three persons (1, 2, and 3). Each person masters one or more skills among the three skills: analysis, web, and database. From the precedence graph of Fig. 28.1, the information [4, 0, 1, N] of node 1 means that activity 1 needs four man-days for skill 1, does not need skill 2 and needs one man-day for skill 3. The last value means that the activity cannot be interrupted. From the skills of persons and the availability table, we deduce that person 2 masters both skills 1 and 3 but does not master skill 2. This person is not available during time-interval [5, 6[. The Gantt diagram of Fig. 28.2 gives a feasible solution. We recall that only activity 3 is allowed to be preempted.



	Skill			
Person	Analysis	Web	DB	Unavailability
1	1	1	1	[4,5[
2	1	0	1	[5,6[
3	1	1	0	

Fig. 28.1 Example

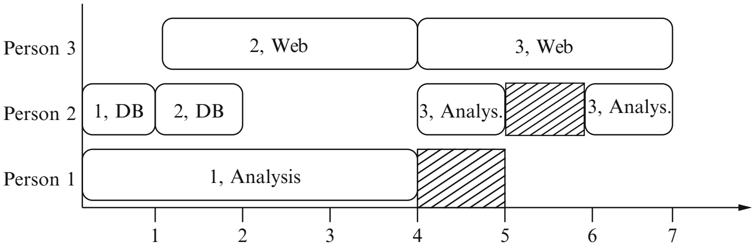


Fig. 28.2 Feasible solution

28.2.2 MIP Formulation

In this section, we focus on the presentation of the proposed Mixed Integer Linear Program (MIP). Notice that, to the best of our knowledge, a few ILP models proposed for the non-preemptive version of the Multi-Skill RCPSP exist (Bellenguez-Morineau and Néron 2004b; Montoya 2012). The main difficulty here lies in modeling simultaneous precedence constraints, skills synchronization and preemption. Notice that preemptive models based on antichains representation have been proposed for modeling preemptive RCPSP (Ballestín et al. 2006; Damay 2005). Moreover, because there are constraints on preemption (i.e., a person who starts an activity must complete this activity), the notion of antichains cannot be trivially adapted to multi-skill RCPSP.

The model presented here is based on a time-indexed formulation. We first formally define time-indexed variables. The MIP formulation is proposed after.

Decision variables:

$$x_{iklt} = \begin{cases} 1 \text{ if resource } k \text{ performs skill } l \text{ for activity } i \text{ during } [t, t+1[\\ 0 \text{ otherwise} \end{cases}$$

$$(i \in V; k \in \mathscr{R}; l \in \mathscr{L}; t = 1, \ldots, T)$$

Auxiliary variables:

- S_i: starting time of activity $i \in V$,
- C_i: completion time of activity $i \in V$,
- C_{il}: completion time of skill $l \in \mathscr{L}$ of activity $i \in V$,
- C_{max}: makespan

Constraints:

Availability of a person and skill constraints: a person cannot perform a skill if he or she does not master it:

$$x_{iklt} \leq a_{klt} \quad (i \in V;\ k \in \mathscr{R};\ l \in \mathscr{L};\ t = 1, \ldots, T) \tag{28.1}$$

Activities must be completed: the sum of each skill part for an activity must be equal to its duration:

$$\sum_{t=0}^{T-1} \sum_{k=1}^{K} x_{iklt} = p_{il} \quad (i \in V;\ l \in \mathscr{L}) \tag{28.2}$$

A resource cannot be assigned to more than one skill at the same time point:

$$\sum_{i=0}^{n+1} \sum_{l=1}^{L} x_{iklt} \leq 1 \quad (k \in \mathscr{R};\ t = 1, \ldots, T) \tag{28.3}$$

Starting and completion dates of activity i can be determined from starting and completion dates of its skill contribution:

$$S_i \geq t - T \cdot \sum_{k=1}^{K} \sum_{q=0}^{t-1} x_{iklq} \quad (i \in V;\ l \in \mathscr{L};\ t = 1, \ldots, T) \tag{28.4}$$

$$S_i \leq t \cdot x_{iklt} + (1 - x_{iklt}) \cdot T \quad (i \in V;\ k \in \mathscr{R};\ l \in \mathscr{L};\ t = 1, \ldots, T) \tag{28.5}$$

$$x_{iklt} \cdot (t + 1) \leq C_{il} \quad (i \in V;\ k \in \mathscr{R};\ l \in \mathscr{L};\ t = 1, \ldots, T) \tag{28.6}$$

$$C_{il} \leq C_i \quad (i \in V;\ l \in \mathscr{L}) \tag{28.7}$$

Precedence constraints:

$$C_i \leq S_j \quad ((i, j) \in E) \tag{28.8}$$

Non-preemption constraint: the starting date added to the duration equals to the completion date of the activity:

$$S_i + p_{il} = C_{il} \quad (i \in V^{np}) \tag{28.9}$$

One person for one skill: at any time point a person can contribute to one skill for one activity at the most and if a person starts a skill, no one else can be assigned to it:

$$x_{iklt} + x_{ik'lt'} \leq 1 \quad (i \in V;\ k \in \mathscr{R};\ k' \in \mathscr{R} :\ k \neq k';\ l \in \mathscr{L};\ t = 1, \ldots, T;\ t' = 1, \ldots, T) \tag{28.10}$$

Makespan:

$$C_{il} \leq C_{max} \quad (i \in V;\ l \in \mathscr{L}) \tag{28.11}$$

Objective function:

$$\text{Min. } C_{max} \tag{28.12}$$

Constraints (28.4) and (28.5) enforce the variables S_i to be equal to the first t so that all activity skills have started. This also models the synchronization constraints. Constraint (28.10) ensures that only one person can be assigned to a given skill of an activity and he or she must finish it. The number of variables and constraints of this model can be reduced. For example, the variables x_{iklt} can be removed for non-preemptive activities, in this case the start date S_i is sufficient. We can also reformulate the constraint (28.10) as follows:

$$x_{iklt} \leq \frac{\sum_{t'=0}^{T-1} x_{iklt'}}{p_{il}} \quad (i \in V;\ k \in \mathscr{R};\ l \in \mathscr{L};\ t = 1, \ldots, T) \tag{28.13}$$

Computational results of this model are presented in Sect. 28.4.2. They show that this MIP can solve only small instances. So, we propose in the next section a heuristic approach to solve instances of medium and large size.

28.3 Resolution Method

In this section, we propose to solve the studied scheduling problem by a heuristic algorithm, namely a list-scheduling algorithm based on the parallel schedule-generation scheme which has been introduced by Kolisch (1996) for solving the classical *RCPSP* problem and has recently been adapted by Bellenguez-Morineau and Néron (2004a) to solve the MSPSP problem. The main adaptation, which we added to the classical parallel scheme, is the possibility, at a time point t, to interrupt an activity which is in progress at t (i.e., started before t and being executed on $[t, t+1]$). For the remainder of this paper, we call this adaptation *Preemptive Adapted Parallel Scheduling Scheme list algorithm* (PAPSS). In the next section, we explain in detail how the *PAPSS* algorithm works.

28.3.1 Preemptive Adapted Parallel Scheduling Scheme List Algorithm (PAPSS)

Computing a solution using this method proceeds as follows: at each time point t, eligible activities, i.e., activities that are ready to be scheduled (release date lower than or equal to t and all predecessors are finished), are sorted according to a given priority rule without violation of precedence relationships constraints. Then, we try to schedule the first activity by solving an assignment problem. If no feasible assignment at t is found, then we move to the next activity in this eligibility list. This way the priority list order is not strictly respected. After visiting all activities in the eligibility list, we move to the first time point where a new event occurred, i.e., a resource becomes available, a new activity becomes eligible. The main algorithm steps are described in Algorithm 28.1. We denote by PAPSS-R1 the execution of the PAPSS algorithm using the priority rule R1.

Algorithm 28.1: Preemptive Adapted Parallel Schedule Scheme Algorithm (*PAPSS-R1*)

```
Require: $V$: set of activities to be scheduled
  $t := 0$
  while $V \neq \phi$ and $t < T$ do
    $\mathscr{D}_t$: set of eligible activities at $t$, sorted according to priority rule $R1$
    $i := 0$
    while $i < |\mathscr{D}_t|$ do
      $feasible := TrySchedule(\mathscr{D}_t(i), t)$
      if $feasible$ then
        $V := V \setminus \mathscr{D}_t(i)$
      end if
      $i := i + 1$
    end while
    $t := nextEvent()$
  end while
```

This algorithm uses two procedures: *TrySchedule*($\mathscr{D}_t(i), t$) which computes the assignment with a minimum cost and *nextEvent*() which returns the next point of time where a new event occurred (resource become available, new activity becomes eligible). The aim of *TrySchedule*($\mathscr{D}_t(i), t$) is to:

1. Compute the assignment with a minimum cost,
2. Schedule the activity $\mathscr{D}_t(i)$ starting from t and using the computed assignment,
3. If the assignment caused a preemption for some activities, then it resumes them as soon as possible.

Priority rules from the literature (see Hartmann and Kolisch 2000), as well as some specific rules are used in the experimental tests. The following priority rules are tested:

- Earliest Release Date (ERD): activities are sorted in non-decreasing order of their release date,
- Earliest Due Date (EDD): activities are sorted in non-decreasing order of their due date,
- Minimum Slack (MINSLK): activities are sorted in non-decreasing order of their slack given by $\max(0, d_i - p_i - ES_i)$,
- Most Number of Successors (MTS): activities are sorted in non-increasing order of the number of their successors,
- Most Number of Skills (MSSKILL): activities are sorted in non-increasing order of their number of needed skills,
- Total man power needed (MAXCHRG): activities are sorted in non-increasing order of the total duration needed given by $\sum_{l=1}^{L} p_{il}$,
- Criticality (CRITICITY): activities are sorted in non-decreasing order of their criticality. The criticality of an activity is the sum of its skills criticalities (see Dhib 2013),

- Greatest rank 1 (GRNK1): activities are sorted in non-increasing order of the greatest rank given by $\sum_{j \in Succ(i)} \sum_{l=1}^{L} p_{jl}$,
- Greatest rank 2 (GRNK2): activities are sorted in non-increasing order of the greatest rank given by $\sum_{j \in Succ(i)} L_j$,
- Greatest rank 3 (GRNK3): activities are sorted in non-increasing order of the greatest rank given by $\sum_{j \in Succ(i)} \max_{l \in \mathscr{L}} p_{jl}$,
- Random priority (RAND): for each activity a random priority rule is associated, the lowest value is the highest priority activity.

28.3.1.1 Activity Assignment

At each time point t, the most prioritized activity among the eligible activities is selected to be scheduled. Persons to be assigned to the activity i are chosen by solving a minimum-cost assignment problem. Due to the synchronization constraint, required persons for processing skills must be assigned to the activity at the beginning, i.e., we have to assign a different person to each skill.

This problem can be solved using the algorithm of Kuhn (1955). A bipartite graph is built using the skills needed by the activity i and they have to be matched efficiently with available persons at time t. To do this matching, we link a skill node and person node if the following conditions are fulfilled:

- Person k masters the skill l,
- If i is not preemptive, k must have a continuous availability equal to p_{il}. In the case of a preemptive activity, k has to be available on $[t, t+1]$.

The availability does not take into account the running preemptive activities which have lower priority than i. More clearly, we define the continuous availability ($avail(k, i, t)$) of person k on time period t with respect to activity i as follows:
$avail(k, i, t) = 0$ if t is an unavailability period for k, or he/she is executing a preemptive activity which has higher priority than i.
$avail(k, i, t) = 1$ if the following conditions are satisfied:

- $[t, t+1[$ is not unavailability period for resource k,
- He/she does not execute any activity on $[t, t+1]$, except if the activity executed is preemptive and is in progress at t (it started before t). In this case, one of two conditions must be satisfied:
  - The activity being executed has lower priority than i,
  - Both activities have the same priority, but the activity i needs more than one skill.

For each person, we compute a criticality indicator measuring the degree of activity requests for his/her person based on his skills and availability. For details on the computation of the criticality indicators, we refer to Bellenguez-Morineau and Néron (2007) and Dhib (2013). So, for each edge from a skill l to a person k we associate a cost corresponding to the person's criticality. If the person k is assigned to a preemptive activity at the time t, a high value is used to avoid preemption if another assignment is possible.

Experimental results using the priority rules described above are presented in Sect. 28.4.3.

28.4 Computational Results

In this section, we first describe the instances used for experimentation, then we present computational results of the linear model. In Sect. 28.4.3, we present results of the PAPSS algorithm using the different priority rules. In Dhib et al. (2011a) we presented three destructive lower bounds: a flow based lower bound, a preemptive linear model inspired from Carlier and Néron (2000, 2003), and an energetic reasoning based lower bound inspired from Baptiste et al. (1999). As the last lower bound was the best among these lower bounds on the majority of tested instances (less than 10 % from the optimal in average), we use it here to measure the quality of the proposed heuristic.

28.4.1 Instance Description

Instances with a number of activities greater than 20 were generated based on the PSPLIB instances (Kolisch et al. 1997). For each instance of PSPLIB (single mode), we use the same network. Hence, for each instance, the complexity NC of its corresponding network is known, i.e., the number of activities n and an indicator of the density of precedence relationships.

The other parameters characterizing an instance, include:

1. Skill per person ($L^{\o}_{pers}$): the average number of mastered skills per person
2. Skill per activity ($L^{\o}_{act}$): the activities' average requirements for skills. It is similar to the resource factor used for RCPSP
3. Average person availability (p_{avail}): the percentage of randomly generated periods of availability for different resources
4. Preemption Average (p_{pmtn}): the percentage of activities that can be preempted.

For each instance of PSPLIB and each configuration (K, L, $L^{\o}_{pers}$, $L^{\o}_{act}$, p_{avail}, p_{pmtn}) one instance is generated. Used parameters are presented in Sect. 28.4.3. We impose that the maximal charge needed by each activity equals the activity duration as defined in the PSPLIB original instance. During the generation process, we try to obtain a maximum number of feasible instances as follows: each time we try to generate skills needed for a given activity, we solve the assignment problem which takes into consideration a person's skills and availabilities. In fact we choose a permutation of skills for a given activity and we solve the problem of assignment that is based on it. If there is no subset of persons who can perform these skills, then

Table 28.1 Instances of 6 persons, 50 % of preemption, 100 % of availability

n	L	T	p_{opt}	$t_{cpu}(s)$	p_{feas}
10	4	14.10	90 %	5.44	10 %
	5	15.10	100 %	114.66	0 %
16	4	22.00	90 %	48.67	0 %
	5	23.50	60 %	209.17	10 %
20	4	22.00	90 %	55.44	10 %
	5	25.90	20 %	164.00	20 %

Table 28.2 Instances of 6 persons, 50 % of preemption, 75 % of availability

n	L	T	p_{opt}	$t_{cpu}(s)$	p_{feas}
10	4	19.0	100 %	119.70	0 %
	5	24.2	80 %	754.25	0 %
16	4	30.9	50 %	915.20	20 %
	5	34.7	10 %	554.00	0 %
20	4	31.7	30 %	87.33	0 %
	5	39.6	0 %	–	10 %

we randomly reduce by one the skills required to perform the activity. We repeat this procedure until we obtain a valid assignment.

Concerning the instances of 10, 16, and 20 activities, we use two sets of instances; we fix the number of persons to 6, the percentage of preemption to 50, the percentage of skills per person to 100, and the percentage of skills per activity to 50. We use the first 10 instances of 10 activities, 16 activities and 20 activities from the PSPLIB multi-mode data set. We retain the graph topology and for each activity we use the duration in the first mode as the maximum man-day request for this activity. In the first set (Table 28.1), the average availability of persons is 100 % while in the second category (Table 28.2), the percentage is 75 %.
For each set and each group of activities, we generate 10 instances with a number of skills equal to 4, then the same number of instances with a number of skills equal to 5. Because the model is dependent on the planning horizon, we opted to run the heuristic method presented in Sect. 28.3, then we took the returned makespan as the maximum horizon.

A personal computer running Linux 2.6.32, 32-bit operating system, with a 2.2 GHZ Duo core Intel processor and 3 GB of memory was used for all tests.

28.4.2 Integer Linear Model Computational Results

As mentioned above in Sect. 28.2, the model cannot be used except for solving small size instances. Tests of Tables 28.1 and 28.2 use only the data sets of 20 activities maximum. The experiments have been conducted using CPLEX 12.2 academic version concert technology library for *Java*. A runtime limit of 3,600 s is used for each instance.

We report the average horizon T (3rd column) that corresponds to C_{max} value given by heuristic PAPSS. Then, T is used as time horizon; In 4th and 5th columns we report the percentage of optimal solutions and the average time in seconds for the optimal solutions, and in the last column we report the percentage of feasible solutions found which are not optimal.

In Table 28.1, we see that out of six data sets, almost all instances with four skills are optimally solved (90 %, 100 %, 90 %). When the average availability is approximately 75 % (Table 28.2), the MIP optimally solves only one group of instances with ten activities and four skills. Unfortunately, the computation time becomes considerable, because as we decrease the availability average (i.e., increase the unavailability average), the planning horizon needed to obtain feasible solutions also increases. Therefore, the number of variables and constraints in the MIP increases considerably.

The last row of Table 28.2 shows a limitation of this model. When it reaches 20 activities, 6 persons and 5 skills with an average planning horizon equal to 40, MIP is generally not able to find any optimal solution within 1 h of execution time and returns only one feasible solution.

28.4.3 Heuristic Computational Results

In this section, we compare the 11 priority rules described in Sect. 28.3. Because of the high number of parameters in instance definition, we decided to compare priority rules by varying the two parameters p_{pmtn} and p_{avail}.

1. Using different availability rates (p_{avail})
   In Table 28.3, three data sets of 160 instances each are used. We used the first 160 instances of 30 activities from the PSPLIB single mode. For each data set

Table 28.3 Priority rules using different availability rates

Priority rule	Avail = 95 %	Avail = 75 %	Avail = 50 %
CRITICITY	**5.719**	7.944	10.361
EDD	**5.619**	9.156	**8.184**
ERD	6.338	8.269	9.076
GRNK	9.475	9.850	9.076
GRNK1	6.588	9.325	8.747
GRNK2	9.738	9.688	8.823
MAXCHRG	9.950	**4.475**	**8.741**
MAXNBSKIL	**5.200**	**4.194**	**8.589**
MINSLK	11.500	9.706	9.551
MTS	7.169	10.013	9.032
RAND	15.700	**5.750**	11.120

Table 28.4 Priority rules using different preemption average

Priority rule	$p_{pmtn} = 0\,\%$	$p_{pmtn} = 25\,\%$	$p_{pmtn} = 50\,\%$	$p_{pmtn} = 75\,\%$	$p_{pmtn} = 100\,\%$
CRITICITY	4.600	**5.719**	**5.981**	**6.650**	**8.863**
EDD	**1.306**	**5.619**	10.088	14.513	18.431
ERD	4.788	6.338	**7.531**	**9.513**	**12.856**
GRNK	**3.338**	9.475	14.719	24.738	29.325
GRNK1	4.281	6.588	10.450	13.838	16.769
GRNK2	3.575	9.738	14.419	26.706	30.044
MAXCHRG	4.044	9.950	18.344	25.694	32.838
MAXNBSKIL	4.769	**5.200**	**5.469**	**5.956**	**5.156**
MINSLK	**1.663**	11.500	21.869	29.419	34.456
MTS	4.194	7.169	9.631	14.944	18.344
RAND	4.456	15.700	24.225	32.850	39.213

we have: $K = 10$, $L = 5$, $L^{\emptyset}_{pers} = 50$, $L^{\emptyset}_{act} = 25$, and $p_{pmtn} = 25\,\%$. The values used for availability average are 95 %, 75 %, and 50 %. For each data set, we report the average gap from the lower bound for each priority rule ($\triangle^{\emptyset}_{LB}$). For all considered priority rules, the computation time is less than 0.02 s in average.

2. Using different preemption average (p_{pmtn})
   In Table 28.4, 5 data sets of 160 instances each are used. We used the first 160 instances of 30 activities, from the PSPLIB single mode. For each data set, we have: $K = 10$, $L = 5$, $L^{\emptyset}_{pers} = 50$, $L^{\emptyset}_{act} = 25$ and $p_{avail} = 95\,\%$. The values used for preemption average (p_{pmtn}) are 0 %, 25 %, 50 %, 75 %, and 100 %. For each data set we report the average gap from the lower bound (Dhib et al. 2011a) for each priority rule ($\triangle^{\emptyset}_{LB}$).

Analysis: In both tables, the three best priority rules in terms of solution quality are marked in bold. We can see that MAXNBSKIL is among the three best rules in every data set except in case of non-preemption (see the first column of Table 28.4). In the case of non-preemption, we can see that all deviations are very close (maximum gap is less than 5 %). For five data sets, MAXNBSKIL is the best among all priory rules. From Table 28.4, MAXNBSKIL is not really dependent on the preemption average (its gaps were less than 6 % in all data sets). Note that MAXNBSKIL is still not affected when the average availability changes (it is ranked first, first and second in the three data sets in Table 28.3). Another important remark concerns the CRITICITY rule. CRITICITY is among the three best rules for four data sets out of seven. Like MAXNBSKIL, the average person availability p_{avail} has no real impact on CRITICITY rule in terms of rank. However, the gap is proportionally increasing with respect to preemption rate. Unlike MAXNBSKIL, CRITICITY is not one of the best three rules when the availability average decreases (as for the last two data sets in Table 28.3).

From the two Tables 28.3 and 28.4, priority rules GRNK, GRNK1, GRNK2, MINSLK and MTS are generally dominated by the remaining rules.

The priority rule MAXCHRG gives bad results when the preemption average increases, but when the unavailability average increases, the impact is not considerable and its gap is still below a 10 % as shown in Table 28.3.

Concerning the priority rule EDD, the availability percentage has no significant impact. EDD appears three times among three best rules. However, from Table 28.4, we can deduce that the quality of solutions given by EDD depends on the preemption average. The impact of the preemption average is not so significant on the quality of solutions given by the ERD rule compared to EDD.

Finally, we note that the random priority rule gives weaker solutions compared to all other rules.

28.5 Conclusions

In this chapter, we studied a multi-skill project scheduling problem encountered in an industrial context considering specific features such as preemption, synchronization and unavailability periods. A mixed integer linear model (MIP) based on time-indexed variables is proposed. Experimental results show that this MIP is not suitable for real instances.

Finally an adapted *parallel schedule-generation scheme* based on priority rules is studied. The quality of solutions is measured using lower bounds proposed in previous works. The algorithm is implemented and deployed on the open-source framework OFBiz. All add-ons developed to integrate the model and the heuristic code with a web interface on OFBiz can be found at http://addons.neogia.org/. Regarding the speed of these heuristics, we used them within metaheuristic methods proposed in Dhib (2013). Further analysis using different problem characteristics may be interesting to classify the effectiveness of priority rules according to the characteristics of the problem.

References

Artigues C, Demassey S, Néron E (2008) Resource-constrained project scheduling: models, algorithms, extension and applications. Wiley, Hoboken

Ballestín F, Valls V, Quintanilla S (2006) Pre-emption in resource-constrained project scheduling. Eur J Oper Res 189(3):1136–1152

Baptiste P, Le Pape C, Nuijten W (1999) Constrained project scheduling problem, satisfiability tests and time bound adjustments for cumulative scheduling problems. Ann Oper Res 92(0):305–333

Bellenguez-Morineau O, Néron E (2004a) Methods for solving the multi-skill project scheduling problem. In: Proceedings of 9th international workshop on project management and scheduling (PMS'2004). Nancy, France, pp 66–69

Bellenguez-Morineau O, Néron E (2004b) Lower bounds for the multi-skill project scheduling problem with hierarchical levels of skills. In: Proceedings of 5th international conference on the practice and theory of automated timetabling (PATAT'2004). Pittsburgh, PA, USA, pp 429–432

Bellenguez-Morineau O, Néron E (2007) A branch-and-bound method for solving multi-skill scheduling problem. Rairo-Oper Res 41(2):155–170

Błażewicz J, Lenstra JK, Rinnooy Kan AHG (1983) Scheduling subject to resource constraints: classification and complexity. Discrete Appl Math 5(1):11–24

Bouleimen K, Lecocq H (2003) A new efficient simulated annealing algorithm for the resource constrained project scheduling problem and its multiple modes version. Eur J Oper Res 149(2):268–281

Carlier J, Néron E (2000) A new LP-based lower bound for the cumulative scheduling problem. Eur J Oper Res 127(2):363–382

Carlier J, Néron E (2003) On linear lower bounds for the resource constrained project scheduling problem. Eur J Oper Res 149(2):314–324

Correia I, Lourenço L, Saldanha-da-Gama F (2010) Project scheduling with flexible resources: formulation and inequalities. OR Spectr 33(3):1–29

Damay J (2005) Techniques de résolution basées sur la Programmation Linéaire pour l'ordonnancement de projet. Ph.D. dissertation, University Blaise Pascal, Clermont-Ferrand, France

Demeulemeester EL, Herroelen WS (1997) New benchmark results for the resource-constrained project scheduling problem. Manag Sci 43(11):1485–1492

Dhib C (2013) Etude et résolution de problèmes d'ordonnancement de projets multi-compétences: Intégration à un ERP libre. Ph.D. dissertation, University of Tours, Tours, France

Dhib C, Kooli A, Soukhal A, Néron E (2011a) Lower bounds for a multi-skill scheduling problem. In: Proceedings of the international conference on operations research (OR'2011). Zürich, Switzerland, pp 471–476

Dhib C, Soukhal A, Néron E (2011b) A multi-skill project scheduling problem in an industrial context. In: Proceedings of the 4th international conference on industrial engineering and systems management (IESM'2011). Metz, France, pp 316–325

Gürbüz E (2010) A genetic algorithm for biobjective multi-skill project scheduling problem with hierarchical levels of skills. Master thesis, Middle East Technical University, Ankara, Turkey

Hartmann S, Drexl A (1998) Project scheduling with multiple modes: a comparison of exact algorithms. Networks 32(3):283–297

Hartmann S, Kolisch R (2000) Experimental evaluation of state of the art heuristics for the resource constrained project scheduling problem. Eur J Oper Res 127(2):394–407

Heimerl C, Kolisch R (2010) Scheduling and staffing multiple projects with a multi-skilled workforce. OR Spectr 32(2):343–368

Kolisch R (1996) Serial and parallel resource-constrained project scheduling methods revisited: theory and computation. Eur J Oper Res 90(2):320–333

Kolisch R, Sprecher A, Drexl A (1997) PSPLIB: a project scheduling library. Eur J Oper Res 96(1):205–216

Kuhn HW (1955) The Hungarian method for the assignment problem. Nav Res Log Q 2:83–97

Li H, Womer K (2009) Scheduling projects with multi-skilled personnel by a hybrid MILP/CP Benders decomposition algorithm. J Sched 12(3):281–298

Montoya C (2012) New methods for the multi-skill project scheduling problem. Ph.D. dissertation, University of Nantes, Nantes, France

Santos AM, Tereso PA (2010) On the multi-mode, multi-skill resource constrained project scheduling problem (MRCPSP-MS). In: Proceedings of the 2nd international conference on engineering optimization. Lisbon, Portugal

Sprecher A, Drexl A (1998) Multi-mode resource-constrained project scheduling by a simple, general and powerful sequencing algorithm. Eur J Oper Res 107(2):431–450

Valls V, Pérez A, Quintanilla S (2009) Skilled workforce scheduling in service centres. Eur J Oper Res 193(3):791–804

Walter M, Zimmermann J (2010) Staffing projects with multi-skilled workers in matrix organisations. In: Proceedings of the 12th international workshop on project management and scheduling (PMS'2010). Tours, France, pp 387–391

Part IX
Discrete Time-Cost Tradeoff Problems

Chapter 29
The Discrete Time-Cost Tradeoff Problem with Irregular Starting Time Costs

Joseph G. Szmerekovsky and Prahalad Venkateshan

Abstract In this chapter we review the literature on the discrete time-cost tradeoff problem (DTCTP). We then present the four integer programming formulations of a version of DTCTP with irregular starting time costs from Szmerekovsky and Venkateshan (Comp and Oper Res 39(7):1402–1410, 2012). Specifically, the problem is an irregular costs project scheduling problem with time-cost tradeoffs. The empirical tests performed in Szmerekovsky and Venkateshan (Comp and Oper Res 39(7):1402–1410, 2012) are updated using the current version of CPLEX and similar results are found being driven by a reduced number of binary variables, a tighter linear programming relaxation, and the sparsity and embedded network structure of the constraint matrix.

Keywords Discrete time-cost tradeoff • Integer linear programming formulations • Payment schedules • Project scheduling

29.1 Introduction

In many project scheduling problems, there exists a time-cost tradeoff. Estimates of activity times presume a given mode in which that activity is to be executed. With each mode for an activity is associated a duration of the activity and the cost involved in completing the activity in that time. Generally, completing an activity in shorter time incurs higher cost. In the discrete time-cost tradeoff problem, there is a discrete set of activity durations each with its associated cost. Discrete time-cost tradeoff problems are typically classified into three types: (1) minimization of cost subject to a deadline for the completion of the project, (2) minimization of completion time subject to a cost budget, and (3) identification of the complete efficient

J.G. Szmerekovsky (✉)
College of Business, North Dakota State University, Fargo, ND, USA
e-mail: Joseph.Szmerekovsky@ndsu.edu

P. Venkateshan
Indian Institute of Management, Ahmedabad, India
e-mail: prahalad@iimahd.ernet.in

C. Schwindt, J. Zimmermann (eds.), *Handbook on Project Management and Scheduling Vol. 1*, International Handbooks on Information Systems,
DOI 10.1007/978-3-319-05443-8_29

time-cost curve. Though many variations of the discrete time-cost tradeoff problem exist, this chapter considers only $MPS\infty|prec, \bar{d}\,|\sum c_i(p_i)$, the time-constrained deterministic version of the problem. The only generalization considered is in the financial objective. To capture the variety of potential financial objectives we consider the problem with irregular starting time costs, which can be denoted as $MPS\infty|prec, \bar{d}\,|\,f$. Information on the generalized, stochastic, and robust versions of the discrete time-cost tradeoff problem can respectively be found in Chap. 30 as well as Chaps. 36 and 39 in the second volume of this handbook.

The remainder of the chapter is organized as follows. In Sect. 29.2 we provide a review of the literature on the basic discrete time-cost tradeoff problem and discuss alternative financial objectives for the problem, including the general problem of scheduling with irregular starting-time costs. We continue in Sects. 29.3 and 29.4 by reviewing the four formulations discussed in Szmerekovsky and Venkateshan (2012) and updating their empirical experiments using the current version of the CPLEX solver. The chapter concludes with a summary in Sect. 29.5.

29.2 Review of the Basic Discrete Time-Cost Tradeoff Problem

There is a large body of literature on the discrete time-cost tradeoff problem. The variety of time-cost tradeoff problems that have been studied in the literature are reviewed by Brucker et al. (1999), De et al. (1995), and Icmeli et al. (1993). In this section we first provide a review of the literature concerning problem $MPS\infty|prec, \bar{d}\,|\sum c_i(p_i)$ and then review alternative financial objectives which can be captured by $MPS\infty|prec, \bar{d}\,|\,f$.

29.2.1 Literature Review for DTCTP

De et al. (1997) have shown that $MPS\infty|prec, \bar{d}\,|\sum c_i(p_i)$ is $\mathcal{NP}$-hard in the strong sense and so it is unlikely that an efficient algorithm for solving it exists. Therefore, methods for addressing $MPS\infty|prec, \bar{d}\,|\sum c_i(p_i)$ fall into three categories: heuristics, exact algorithms with exponential worst-case complexity, and mathematical programming. We now review the results in these three areas.

Despite the difficulty of the problem little research has focused on heuristics for $MPS\infty|prec, \bar{d}\,|\sum c_i(p_i)$. The first heuristic was an approximation algorithm based on rounding the solution to a natural linear relaxation of the problem developed by Skutella (1998). Deineko and Woeginger (2001) have shown that even the approximation problem is hard. Since then, a column generation-based heuristic has been developed by Akkan et al. (2005) and shown to perform well compared to local search and Lagrangian-based heuristics. More recently, Vanhoucke and Debels

(2007) developed a metaheuristic for a more general version of the discrete time-cost tradeoff problem, but tested it on the basic version as well. They found that though the metaheuristic often provides the optimal solution, it does so less efficiently than the exact procedure of Demeulemeester et al. (1998).

The first exact procedures designed for $MPS\infty|prec,\bar{d}|\sum c_i(p_i)$ were dynamic programs and branch-and-bound procedures with exponential worst case performance proposed by Crowston (1970), Panagiotakopoulos (1977), Harvey and Patterson (1979), and Hindelang and Muth (1979), whose algorithm was later corrected in De et al. (1997). More recently, in order to identify the complete and efficient time-cost curve, Demeulemeester et al. (1996, 1998) developed exact algorithms for $MPS\infty|prec,\bar{d}|\sum c_i(p_i)$. The algorithms of Demeulemeester et al. (1996) use network reductions and the fact that for series-parallel (s/p) networks, $MPS\infty|prec,\bar{d}|\sum c_i(p_i)$ can be solved efficiently, as discussed in De et al. (1995). Hence, exact enumeration and branch-and-bound procedures based on reducing non-s/p networks to s/p networks were developed for solving $MPS\infty|prec,\bar{d}|\sum c_i(p_i)$. The branch-and-bound algorithm of Demeulemeester et al. (1998) uses a convex piecewise linear approximation to the discrete time-cost curve for each activity, which underestimates the true cost. This piecewise linear problem can be solved efficiently and is the basis for the branch-and-bound procedure. Extensive computational testing on randomly generated data has shown that this new procedure tends to substantially outperform the series/parallel based procedure of Demeulemeester et al. (1996) and is potentially the best performing exact procedure to date.

The earliest mathematical programming formulations for $MPS\infty|prec,\bar{d}|\sum c_i(p_i)$ were introduced by Meyer and Shaffer (1965) and Crowston and Thompson (1967), but these suffered from computational limitations. Akkan et al. (2005) develop a binary programming formulation based on network decompositions and analyze it using column generation techniques. Through computational tests the resulting algorithm is shown to provide good quality heuristic solutions and bounds in a reasonable amount of computational time even for hard benchmark instances. More recently, Hadjiconstantinou and Klerides (2010) have investigated a path-based binary programming formulation for $MPS\infty|prec,\bar{d}|\sum c_i(p_i)$. The formulation enforces the deadline through use of a constraint for each path in the project network. The problem is then solved with a cutting plane algorithm based on relaxing the path constraints and then adding them in as cuts as needed. The algorithm is also augmented with speed-up techniques based on identifying promising paths to preemptively add them as deadline constraints before they are needed as cuts. Computational tests were performed demonstrating that the cutting plane algorithm outperforms the branch-and-bound procedure of Demeulemeester et al. (1998) and performs well on the test instances from Akkan et al. (2005) providing optimal solutions to nearly all of them. Hafizoğlu and Azizoğlu (2010) recently have also analyzed a binary programming formulation for the problem. They develop a branch-and-bound procedure based primarily on the linear relaxation of the integer program and the fact that at most two modes for any activity will have positive corresponding variable values in the optimal solution of the linear relaxation. Computational testing has shown their procedure to be able to solve or provide high quality (within 3 % of optimality) heuristic solutions for large problems (150 activities and 10 modes) in a reasonable amount of time (1 h).

29.2.2 Literature Review for DTCTP with Irregular Starting Time Costs

Szmerekovsky and Venkateshan (2012) study $MPS\infty|prec,\bar{d}|f$ with the goal of reconciling time-cost tradeoff problems and problems with financial objective functions. They argue that financial objective functions better represent the corresponding interests of contractors and clients that must be traded off in project scheduling. For example, $MPS\infty|prec,\bar{d}|\sum c_i(p_i)$ requires contractors to incur additional expenses to meet the deadline. Supposedly this is because the client requires the project by the due date. Hence, the deadline alone represents the interests of the client. Clearly this is a gross simplification of the client's interests as, given different costs for different completion dates, the client could be better off with later or earlier project completion dates. Szmerekovsky and Venkateshan (2012) argue that these financial time-cost tradeoffs can be accounted for by considering payment schedules (i.e., time-dependent bonuses and penalties, payment retention, the selection of milestone activities, and periodic payment schemes) and the financial objectives of net present value (NPV) and cash availability (CA).

The NPV objective for project scheduling originally appeared in Russell (1970) and has since received significant attention. A recent review of time-constrained net present value problems can be found in Drezet (2008, Sect. 14.3.1). Resource-constrained, multi-mode, and stochastic versions of the net present value problem are discussed in Chaps. 14 and 23 as well as Chap. 35 in the second volume of this book, respectively. Problems which consider the client's, as opposed to just the contractor's, interests are discussed in Bey et al. (1981), Dayanand and Padman (1998), Ulusoy and Cebelli (2000), Dayanand and Padman (2001), and Szmerekovsky (2005). The CA objective was first described in Goldratt (1997) and has since been discussed in Demeulemeester and Herroelen (2002) and Szmerekovsky and Vairaktarakis (2006).

As observed by Szmerekovsky and Venkateshan (2012) using a general objective function, commonly referred to as an irregular costs function, can capture the variety of payment scheduling methods and objectives previously discussed. Möhring et al. (2001) review the time-constrained version of the problem without time-cost tradeoffs and Grigoriey and Woeginger (2004) establish the complexity of $MPS\infty|prec,\bar{d}|f$ as $\mathcal{NP}$-hard and $\mathcal{APX}$-hard even when the precedence constraints are of bounded height two or form an interval order.

We present the new integer programming formulation for $MPS\infty|prec,\bar{d}|f$ investigated by Szmerekovsky and Venkateshan (2012) in Sect. 29.3. We then update their empirical experiments to the newest version of CPLEX in Sect. 29.4.

29.3 A New Integer Programming Formulation for the DTCTP with Irregular Starting Time Costs

This section considers a new integer programming formulation for $MPS\infty|prec, \bar{d}|f$, the *irregular costs* project scheduling problem with time-cost tradeoffs. We formalize the problem and provide the New Formulation (NF) from Szmerekovsky and Venkateshan (2012). The project is represented by an activity-on-node (AoN) network $G = (V, E)$ with $n + 2$ activities $i \in V$ and arc set E. The activities i are numbered from 0 to $n + 1$ with dummy activity 0 representing the project beginning and dummy activity $n + 1$ standing for the project termination. Associated with each activity i are M_i alternative processing modes such that processing i in mode m, $m = 1, \ldots, M_i$, requires p_{im} time units and $p_{i(m+1)} < p_{im}$ for $m = 1, \ldots, M_i - 1$. That is, usage of a higher mode leads to quicker completion of the activity. For dummy activities, $p_{01} = 0$, $p_{(n+1)1} = 0$, $M_0 = 1$, and $M_{n+1} = 1$. Associated with each arc $(i, j) \in E$ is a time lag δ_{ijm}^{CC} such that if activity i completes at time C_i then activity j cannot finish before time $C_i + \delta_{ijm}^{CC}$ if j is processed in mode m. Specifically, for all computational tests we will have $\delta_{ijm}^{CC} = p_{jm}$. If activity i is processed in mode m and is completed at time t, then the contractor receives an NPV npv_{imt} from processing activity i. Note that npv_{imt} can be positive, indicating a net gain from completing the activity, or negative, indicating a net loss associated with the activity. For dummy activities, $npv_{01t} = 0$ and $npv_{(n+1)1t} = 0$. Furthermore, without loss of generality, we assume that all processing times p_{im} are integers. Finally, let $\bar{d}$ be the integer length of the planning horizon in which the contractor intends to complete the project, so that the project must be scheduled to complete on or before time $\bar{d}$.

Möhring et al. (2001) review integer programming formulations for the irregular costs project scheduling problem when time-cost tradeoffs are not allowed. We now present Szmerekovsky and Venkateshan (2012)'s generalization of the first of these formulations, referred to as the New Formulation (NF). The following variables are used in the formulation:

$$z_{imt} = \begin{cases} 1, & \text{if activity } i \text{ is executed in mode } m \text{ and finishes on or before time } t \\ 0, & \text{otherwise} \end{cases}$$

for $i \in V$, $m = 1, \ldots, M_i$, and $t = 0, \ldots, \bar{d}$.

Note that these variables are similar to those used for Integer Programming Formulation I in Möhring et al. (2001) but they have been generalized to account for multiple processing modes representing time-cost tradeoffs. The nature of the variables z_{imt} requires that the objective coefficients, c_{imt}, are calculated recursively using the npv_{imt} values. This is done as follows:

$$
\begin{aligned}
c_{iM_i\overline{d}} &= npv_{iM_i\overline{d}} \\
c_{imt} &= npv_{imt} - \sum_{h=m+1}^{M_i}\sum_{s=t}^{\overline{d}} c_{ihs} - \sum_{s=t+1}^{\overline{d}} c_{ims} - \sum_{h=1}^{m-1}\sum_{s=t+(p_{ih}-p_{im})}^{\overline{d}} c_{ihs} \\
&\qquad (i \in V;\ t = \overline{d},\ldots,0;\ m = M_i,\ldots,1)
\end{aligned}
$$

We now formulate $MPS\infty|prec, \bar{d}|f$ as follows:

$$\text{Max.} \sum_{i=1}^{n}\sum_{m=1}^{M_i}\sum_{t=0}^{\overline{d}} c_{imt}z_{imt} \tag{29.1}$$

$$\text{s. t.}\quad z_{imt} - z_{im(t-1)} \geq 0 \quad (i \in V : M_i = 1;\ t = 1,\ldots,\overline{d}) \tag{29.2}$$

$$z_{imt} - z_{i(m-1)t} \geq 0 \quad (i \in V : M_i > 1;\ m = 2,\ldots,M_i;\ t = 0,\ldots,\overline{d}) \tag{29.3}$$

$$z_{imt} - z_{i(m+1)(t-[p_{im}-p_{i(m+1)}])} \geq 0 \tag{29.4}$$

$$(i \in V : M_i > 1;\ m = 1,\ldots,M_i - 1;\ t = p_{im} - p_{i(m+1)},\ldots,\overline{d})$$

$$z_{iM_i(t-\delta^{CC}_{ijm})} - z_{jmt} \geq 0 \quad ((i,j) \in E;\ m = 1,\ldots,M_i;\ t = 0,\ldots,\overline{d}) \tag{29.5}$$

$$z_{imt} + z_{iM_i(t-1)} - z_{i(m+1)(t-[p_{im}-p_{i(m+1)}])} \leq 1 \tag{29.6}$$

$$(i \in V;\ m = 1,\ldots,M_i - 1;\ t = p_{im} - p_{i(m+1)},\ldots,\overline{d})$$

$$z_{(n+1)M_{(n+1)}\overline{d}} = 1 \tag{29.7}$$

$$z_{iM_it} \in \{0,1\} \quad (i \in V;\ t = 0,\ldots,\overline{d}) \tag{29.8}$$

The objective (29.1) is to maximize the contractor's NPV. Constraints (29.2)–(29.4) enforce the logical definition of the variables. Here, (29.2) handle activities with a single mode and (29.3)–(29.4) handle activities with two or more modes. Constraints (29.5) enforce the time lags associated with the project network and when $t - \delta^{CC}_{ijm} < 0$, we set $z_{jmt} = 0$. Constraints (29.6) prevent an activity from having more than one completion time and processing mode by guaranteeing that the variables do not produce more than one corner entry. In (29.6) this is done for each z_{imt} by ensuring that if z_{imt} is the corner entry, then all other variables in the preceding time column must be equal to 0 ($z_{iM_i(t-1)} = 0$). If this is not so, i.e., $z_{imt} = 1$ and $z_{iM_i(t-1)} = 1$, then activity i does not finish at time t and $z_{i(m+1)(t-[p_{im}-p_{i(m+1)}])} = 1$. Finally, constraint (29.7) guarantees that activity $n+1$ is completed (all other activities are covered by having activity $n+1$ as a successor), and constraints (29.8) correspond to the binary conditions of the variables.

Notice that all of constraints (29.2)–(29.5) have exactly one $+1$ and one -1 coefficient. Hence, for the special case of a single mode, for each activity there will be no constraints (29.6) and the remaining constraints will constitute a totally unimodular matrix. Therefore any extreme point solution will satisfy the binarity constraints (29.8). This problem is well-known as the maximum closure problem of a directed graph and can be solved as a minimum-cut problem. The maximum closure problem and related problems are discussed in Hochbaum (2004). Further, the above formulation provides an alternative derivation of the ability to solve the irregular cost problem (without crashing) through a minimum-cut computation as is done in Möhring et al. (2003). Finally, notice that the constraints (29.8) require only the variables z_{iM_it} to be binary as once all of these variables are either 0 or 1, the constraints (29.6) will also have exactly one $+1$ and one -1 coefficient so that a totally unimodular matrix will be obtained. Moreover, once the z_{iM_it} variables are fixed, the problem decomposes into $n + 2$ independent problems, one for each $i \in V$.

29.4 Computational Results

In this section, we update the computational experiments of Szmerekovsky and Venkateshan (2012), herein after referred to as the original experiments, to the current version of CPLEX (version 12.4) to reevaluate and compare NF to three other formulations on a randomly generated set of problem instances. We first review the data generation for the experiments.

29.4.1 Problem Generation

The problem generation followed the method of the original experiments with project networks being randomly generated using the procedure of Kolisch and Sprecher (1996) and activity costs being randomly generated using the procedure of Szmerekovsky (2005). The parameters used in the network and cost generation procedures and for generating the rest of the problem data appear in Table 29.1. Most of the parameters in Table 29.1 have already been defined or are self explanatory. However, the number of activities, network density, maximum crashing, and payment scheme parameters are worth commenting on. Regarding network density, the notion of network density used was the ratio of the number of arcs in the network to the total number of arcs possible in a network. Further, the maximum crashing parameter refers to the maximum percentage by which an activity's duration can be shortened compared to p_{i1}. For each integer between the resulting minimum duration and the maximum duration p_{i1} we introduce one execution mode m for activity i. So, if $p_{i1} = 7$ and the maximum amount of crashing is 50 %, then activity i would have $M_i = 4$ modes with $p_{i2} = 6$, $p_{i3} = 5$, and $p_{i4} = 4$. Finally, regarding the payment schemes, LSP refers to the contractor receiving a single payment for the project at project completion and ETI refers to payments at project completion

Table 29.1 Parameter values used for problem generation

Parameter	Values
Number of activities (n)	30, 60, and 90
Network density	25 and 75 %
Maximum activity duration (p_{i1})	Generated from a uniform distribution on the intervals [1,10] and [1,20]
Maximum crashing	25, 50, and 75 %
Discount factor (β)	0.99, 0.995, and 1
Edge cost for edge (i, j) (c_{ij})	Generated from a uniform distribution on the interval [50,1050]
Mode 1 cost for activity i (c_i)	$\sum_{j:(i,j)\in E} c_{ij}$
Percentage of c_i which is a variable cost	50 %
Ratio of mode 1 variable cost to mode M_i variable cost	p_{iM_i}/p_{i1}
Contractor markup	50 %
Payment schemes	Lump-Sum Payment (LSP) and Equal Time Intervals (ETI)

and every five time units over the project execution. When payments are made under the ETI scheme they refer to the work associated with all completed activities, with no payment being made for partially completed activities.

As in the original experiments, for each combination of the number of activities, the network density, the maximum crashing percentage, and the payment scheme 10 problem instances were generated. All problem instances were solved using CPLEX 12.4 without preprocessing and all other options set to default on a 3.3 GHz Intel Core i5 processor having 4 GB of physical memory running Windows 7. All computer code was written in C++ and compiled using Microsoft Visual Studio 2008.

29.4.2 Comparing NF with Standard Formulations

In this section, we compare the computational time required to optimally solve $MPS\infty|prec, \bar{d}|f$ using formulation NF with the same three alternative formulations used in Szmerekovsky and Venkateshan (2012). The first (referred to hereafter as Standard I) is based on that used in Kolisch and Sprecher (1996) to model the discrete time-cost tradeoff problem and uses the following decision variables:

$$y_{imt} = \begin{cases} 1, & \text{if activity } i \text{ finishes at time } t \text{ in mode } m \\ 0, & \text{otherwise} \end{cases}$$

for $i \in V$, $m = 1, \ldots, M_i$ and $t = 0, \ldots, \bar{d}$.

The following integer program, which uses decision variables y_{imt}, represents an exact formulation of problem $MPS\infty|prec, \overline{d}|f$.

$$\text{Max.} \sum_{i=1}^{n} \sum_{m=1}^{M_i} \sum_{t=0}^{\overline{d}} npv_{imt} y_{imt}$$

$$\text{s. t.} \quad \sum_{m=1}^{M_i} \sum_{t=0}^{\overline{d}} y_{imt} = 1 \quad (i \in V) \tag{29.9}$$

$$\sum_{m=1}^{M_i} \sum_{t=0}^{\overline{d}} t y_{imt} \leq \sum_{m=1}^{M_j} \sum_{t=\delta_{ijm}^{CC}}^{\overline{d}} (t - \delta_{ijm}^{CC}) y_{jmt} \quad ((i, j) \in E) \tag{29.10}$$

$$y_{imt} \in \{0, 1\} \quad (i \in V;\ m = 1, \dots, M_i;\ t = 0, \dots, \overline{d})$$

Constraints (29.9) ensure that each activity is performed, and constraints (29.10) enforce the precedence relationships among the activities.

The second formulation, based on that used in Möhring et al. (2001) (referred to hereafter as Standard II), is also tested. The formulation below is a multi-modal generalization of integer programming (IP) formulation II in Möhring et al. (2001). It uses the following decision variables:

$$x_{imt} = \begin{cases} 1, & \text{if activity } i \text{ starts at time } t \text{ in mode } m \\ 0, & \text{otherwise} \end{cases}$$

for $i \in V$, $m = 1, \dots, M_i$ and $t = 0, \dots, \overline{d}$.

The following integer program uses decision variables x_{imt} to formulate problem $MPS\infty|prec, \overline{d}|f$.

$$\text{Max.} \sum_{i=1}^{n} \sum_{m=1}^{M_i} \sum_{t=0}^{\overline{d}} npv'_{imt} x_{imt}$$

$$\text{s. t.} \quad \sum_{m=1}^{M_i} \sum_{t=0}^{\overline{d}-p_{im}} x_{imt} = 1 \quad (i \in V) \tag{29.11}$$

$$\sum_{s=t}^{\overline{d}} x_{ims} + \sum_{m=1}^{M_j} \sum_{s=0:\ s \leq \overline{d}}^{t+p_{im}-1} x_{jms} \leq 1 \tag{29.12}$$

$$((i, j) \in E;\ t = 0, \dots, \overline{d};\ m = 1, \dots, M_i)$$

$$x_{imt} \in \{0, 1\} \quad (i \in V;\ m = 1, \dots, M_i;\ t = 0, \dots, \overline{d})$$

where $npv'_{imt} = npv_{im(t-p_{im})}$.

Constraints (29.11) ensure that each activity is performed. Constraints (29.12) enforce the precedence relationship amongst activities.

The third formulation (referred to hereafter as Standard III) makes use of the same variables y_{imt}, but uses purely discrete properties to describe the set of feasible solutions. The following integer program also provides an exact formulation of problem $MPS\infty|prec, \bar{d}\,|\,f$.

$$\text{Max.} \sum_{i=1}^{n} \sum_{m=1}^{M_i} \sum_{t=0}^{\bar{d}} npv_{imt} y_{imt}$$

$$\text{s. t.} \quad \sum_{m=1}^{M_i} \sum_{t=0}^{\bar{d}} y_{imt} = 1 \quad (i \in V) \tag{29.13}$$

$$\sum_{m=1}^{M_i} \sum_{s=t}^{\bar{d}} y_{ims} + \sum_{m=1}^{M_j} \sum_{s=0}^{t+p_{jm}-1} y_{jms} \le 1 \quad ((i,j) \in E;\ t = 0, \ldots, \bar{d}) \tag{29.14}$$

$$y_{imt} \in \{0, 1\} \quad (i \in V;\ m = 1, \ldots, M_i;\ t = 0, \ldots, \bar{d})$$

In Standard III constraints (29.13) ensure that each activity is performed, and constraints (29.14) enforce the precedence relationships among the activities.

Each instance of $MPS\infty|prec, \bar{d}\,|\,f$ was solved using NF and the three standard formulations. Tables 29.2, 29.3, 29.4, and 29.5 summarize the computational results. The entries in these tables represent the average NPV and computational time over all ten problem instances that are generated for each combination of the number of activities, the network density, the maximum crashing percentage, and the payment scheme. As in the original experiments, a maximum limit of the smallest multiple of 5 min greater than the maximum time taken to solve any problem instance using NF was imposed per problem instance solved using the standard formulations. As in the original experiments, this resulted in the optimal solution not being found for all instances with the Standard I and Standard II formulations. Specifically, with the Standard II formulation not even a feasible solution was obtained for many problems where each activity duration was generated from the interval [1, 20]. When more than 50 % of the problem instances were unsolvable in the 300 s limit, average values are not reported for the Standard II formulation in the tables. More details on the failure of Standard I and Standard II to solve the problem instances are reported in Table 29.8 and discussed subsequently.

Tables 29.2, 29.3, 29.4, and 29.5 report the results of the updated experiments. Specifically, we are able to observe the following:

- The average computational time is increasing with the range of activity durations ([1,10] versus [1,20]) and the maximum crashing amount, with the effect of the range of activity durations being larger.
- The average computational time is increasing with the density of the network.

Table 29.2 Comparison of formulations—LSP, density = 25 %, $n = 30$; underlined entries in the t_{cpu} columns indicate the formulation that resulted in least computational time

			Crashing By								
			25 %			50 %			75 %		
Formulation	Interval	β	LP	IP	t_{cpu}	LP	IP	t_{cpu}	LP	IP	t_{cpu}
Standard I	[1–10]	1	34,260	12,390	$\underline{<1}$	35,277	12,390	$\underline{<1}$	36,046	12,390	$\underline{1}$
		0.995	32,921	6,559	1	34,464	6,895	19	35,461	6,950	43
		0.99	32,148	2,467	1	34,164	3,194	8	35,308	3,360	31
Standard II	[1–10]	1	12,390	12,390	2	12,390	12,390	4	12,390	12,390	6
		0.995	10,962	6,559	50	12,927	6,300	281	13,050	6,366	300
		0.99	10,389	2,467	57	14,483	2,366	300	15,316	2,574	300
Standard III	[1–10]	1	13,190	12,390	2	13,440	12,390	4	13,566	12,390	4
		0.995	7,527	6,559	2	8,187	6,895	4	8,390	6,950	$\underline{5}$
		0.99	3,452	2,467	3	4,540	3,194	4	4,837	3,360	$\underline{6}$
NF	[1–10]	1	12,390	12,390	$\underline{<1}$	12,390	12,390	2	12,390	12,390	6
		0.995	6,559	6,559	$\underline{<1}$	6,895	6,895	$\underline{2}$	6,950	6,950	6
		0.99	2,467	2,467	$\underline{<1}$	3,194	3,194	$\underline{3}$	3,360	3,360	7
Standard I	[1–20]	1	34,477	12,390	$\underline{1}$	35,397	12,390	$\underline{3}$	36,114	12,390	$\underline{4}$
		0.995	32,649	2,963	215	34,398	3,647	280	35,449	3,691	300
		0.99	31,806	−1,559	76	34,085	−522	241	35,376	−272	300
Standard III	[1–20]	1	13,262	12,390	12	13,475	12,390	21	13,584	12,390	24
		0.995	4,049	2,963	14	5,033	3,648	33	5,227	3,700	$\underline{40}$
		0.99	−649	−1,559	12	609	−521	28	1,001	−234	$\underline{36}$
NF	[1–20]	1	12,390	12,390	3	12,390	12,390	17	12,390	12,390	31
		0.995	2,963	2,963	$\underline{4}$	3,648	3,648	$\underline{23}$	3,700	3,700	47
		0.99	−633	−1,559	$\underline{4}$	−64	−521	$\underline{24}$	43	−234	57

- NF provides the tightest LP relaxation with a gap of less than 1 % when the optimal NPV of the project is positive. Further, NF performs competitively with respect to computational time for all problem instances.
- Standard I provides the loosest LP relaxation and is only competitive with respect to computational time when there is no time value of money ($\beta = 1$). It also experiences better performance with the ETI payment scheme when the time between payments and, hence, the impact of the time value of money is less (compared to the LSP payment scheme).
- Standard II provides the third tightest LP relaxation but performs poorest with respect to computational time. Specifically, for larger problem instances with activity durations drawn from [1,20] it failed to solve more than half of the problem instances in the time limit.
- Standard III provides the second tightest LP relaxation and is competitive with respect to computational time especially when activity durations can be reduced by up to 75 % through crashing.

Table 29.3 Comparison of formulations—ETI, density = 25 %, $n = 30$; underlined entries in the t_{cpu} columns indicate the formulation that resulted in least computational time

			Crashing by								
			25 %			50 %			75 %		
Formulation	Interval	β	LP	IP	t_{cpu}	LP	IP	t_{cpu}	LP	IP	t_{cpu}
Standard I	[1–10]	1	12,390	12,390	$\underline{\leq 1}$	12,390	12,390	$\underline{1}$	12,390	12,390	$\underline{1}$
		0.995	11,094	10,136	$\underline{\leq 1}$	1,090	10,133	21	11,095	10,129	$\underline{1}$
		0.99	10,065	8,264	$\underline{\leq 1}$	10,066	8,266	31	10,097	8,248	31
Standard II	[1–10]	1	12,390	12,390	2	12,390	12,390	3	12,390	12,390	6
		0.995	10,487	10,136	12	10,414	10,133	47	10,305	10,129	28
		0.99	9,047	8,264	41	8,935	8,238	86	8,734	8,246	69
Standard III	[1–10]	1	12,390	12,390	1	12,390	12,390	2	12,390	12,390	3
		0.995	10,167	10,136	2	10,170	10,133	3	10,171	10,129	4
		0.99	8,339	8,264	2	8,336	8,266	3	8,338	8,248	$\underline{4}$
NF	[1–10]	1	12,390	12,390	$\underline{\leq 1}$	12,390	12,390	2	12,390	12,390	5
		0.995	10,139	10,136	$\underline{\leq 1}$	10,136	10,133	$\underline{2}$	10,132	10,129	6
		0.99	8,267	8,264	$\underline{\leq 1}$	8,268	8,266	$\underline{2}$	8,252	8,248	5
Standard I	[1–20]	1	12,390	12,390	$\underline{2}$	12,390	12,390	$\underline{3}$	12,390	12,390	$\underline{3}$
		0.995	10,769	8,647	205	10,763	8,633	189	10,793	8,613	128
		0.99	9,733	6,070	202	9,751	6,042	258	9,902	5,978	211
Standard III	[1–20]	1	12,390	12,390	6	12,390	12,390	8	12,390	12,390	13
		0.995	8,691	8,650	14	8,687	8,640	20	8,686	8,613	$\underline{22}$
		0.99	6,193	6,075	13	6,177	6,052	21	6,185	6,010	$\underline{29}$
NF	[1–20]	1	12,390	12,390	3	12,390	12,390	17	12,390	12,390	32
		0.995	8,652	8,650	$\underline{3}$	8,641	8,640	$\underline{16}$	8,615	8,613	30
		0.99	6,076	6,075	$\underline{3}$	6,052	6,052	$\underline{17}$	6,010	6,010	33

The first point is intuitive in that longer activity durations and more crashing will result in more activity modes. Further, the longer activity durations also result in a larger project horizon $(\overline{d})$ having a double impact, which explains the stronger effect. Though the first point is consistent with the results of the original experiments, the second point is not. In the original experiments the effect of network density on average computational time was ambiguous. But, with the updated experiments using the latest version of CPLEX we see a clear trend that network density increases average computational time. This is likely due to a combination of increased number of constraints in the formulations and the associated increased project horizons $(\overline{d})$. The remaining points concerning the performances of the different formulations are consistent with the original experiments. The relative advantages of the different formulations can be explained by considering the sparsity of the formulations.

Tables 29.6 and 29.7 report the average densities and number of nonzeroes for the formulations. Clearly, NF provides the sparsest formulation with the average density of the constraint matrix never exceeding 0.05 % while the other formulations

Table 29.4 Comparison of formulations—LSP, density = 75 %, $n = 30$; underlined entries in the t_{cpu} columns indicate the formulation that resulted in least computational time

			Crashing By								
			25 %			50 %			75 %		
Formulation	Interval	β	LP	IP	t_{cpu}	LP	IP	t_{cpu}	LP	IP	t_{cpu}
Standard I	[1–10]	1	42,743	15,142	$\underline{<1}$	43,713	15,142	$\underline{1}$	44,317	15,142	$\underline{1}$
		0.995	41,274	7,163	2	42,823	7,664	6	43,646	7,672	58
		0.99	40,495	1,801	2	42,510	2,860	53	43,473	3,144	171
Standard II	[1–10]	1	15,142	15,142	3	15,142	15,142	5	15,142	15,142	7
		0.995	14,184	7,163	102	16,597	7,067	300	16,989	6,565	300
		0.99	14,065	1,801	84	19,026	2,062	300	20,444	1,712	300
Standard III	[1–10]	1	16,081	15,142	3	16,351	15,142	5	16,443	15,142	6
		0.995	8,254	7,163	4	9,095	7,664	8	9,241	7,672	$\underline{9}$
		0.99	2,911	1,801	4	4,380	2,860	8	4,771	3,144	$\underline{9}$
NF	[1–10]	1	15,142	15,142	$\underline{<1}$	15,142	15,142	3	15,142	15,142	7
		0.995	7,163	7,163	$\underline{1}$	7,664	7,664	$\underline{4}$	7,672	7,672	$\underline{9}$
		0.99	1,836	1,801	$\underline{<1}$	2,860	2,860	$\underline{4}$	3,145	3,144	$\underline{9}$
Standard I	[1–20]	1	42,823	15,142	$\underline{2}$	43,701	15,142	$\underline{4}$	44,463	15,142	$\underline{6}$
		0.995	40,733	2,207	253	42,532	3,265	300	43,676	3,538	300
		0.99	39,877	−3,245	129	42,219	−1,645	269	43,639	−1,077	300
Standard III	[1–20]	1	16,077	15,142	18	16,331	15,142	28	16,481	15,142	41
		0.995	3,346	2,208	33	4,756	3,286	51	5,212	3,582	$\underline{60}$
		0.99	−2,318	−3,245	18	−583	−1,645	$\underline{34}$	159	−999	$\underline{48}$
NF	[1–20]	1	15,142	15,142	6	15,142	15,142	20	15,142	15,142	39
		0.995	2,208	2,208	$\underline{8}$	3,286	3,286	$\underline{39}$	3,582	3,582	$\underline{60}$
		0.99	−960	−3,245	$\underline{9}$	−271	−1,645	37	−116	−999	89

experience densities which are approximately two orders of magnitude larger. This allows NF to be competitive in terms of the total number of nonzero coefficients despite its large number of constraints with only Standard I having fewer nonzeroes on average and Standard II and Standard III having one or two orders of magnitude more nonzeroes on average. The tables also highlight the reason for the poor performance of Standard II, as its loose LP relaxation can be seen to be paired with a large number of nonzeroes, resulting in poor computational performance. Further, on average Standard III has only approximately one-fifth to one-half the number of nonzeroes of Standard II. When coupled with its poor LP relaxation, this explains the superior solution times with other formulations compared to Standard II as well as the problems encountered when using the Standard II formulation in solving problems where activity durations are drawn from [1,20]. In contrast, the other three formulations show tradeoffs in the tightness of LP relaxations, the number of nonzeroes, and their sparsity so that they remain competitive with each other on various problem instances.

Table 29.5 Comparison of formulations—ETI, density = 75 %, $n = 30$; underlined entries in the t_{cpu} columns indicate the formulation that resulted in least computational time

			Crashing by								
			25 %			50 %			75 %		
Formulation	Interval	β	LP	IP	t_{cpu}	LP	IP	t_{cpu}	LP	IP	t_{cpu}
Standard I	[1–10]	1	15,142	15,142	$\underline{\leq 1}$	15,142	15,142	$\underline{1}$	15,142	15,142	$\underline{1}$
		0.995	13,670	12,331	$\underline{\leq 1}$	13,666	12,350	$\underline{2}$	13,674	12,332	$\underline{3}$
		0.99	12,524	10,039	$\underline{\leq 1}$	12,537	10,093	4	12,587	10,024	12
Standard II	[1–10]	1	15,142	15,142	3	15,142	15,142	5	15,142	15,142	8
		0.995	12,843	12,331	31	12,742	12,350	53	12,599	12,332	53
		0.99	11,115	10,039	52	10,963	10,072	144	10,686	9,759	158
Standard III	[1–10]	1	15,142	15,142	2	15,142	15,142	3	15,142	15,142	4
		0.995	12,370	12,331	3	12,384	12,350	4	12,370	12,332	5
		0.99	10,162	10,039	3	10,171	10,093	5	10,163	10,024	7
NF	[1–10]	1	15,142	15,142	$\underline{\leq 1}$	15,142	15,142	3	15,142	15,142	8
		0.995	12,333	12,331	$\underline{\leq 1}$	12,350	12,350	$\underline{2}$	12,333	12,332	7
		0.99	10,044	10,039	$\underline{\leq 1}$	10,094	10,093	$\underline{2}$	10,026	10,024	$\underline{6}$
Standard I	[1–20]	1	15,142	15,142	$\underline{2}$	15,142	15,142	$\underline{3}$	15,142	15,142	$\underline{4}$
		0.995	13,300	10,542	187	13,316	10,560	189	13,384	10,500	181
		0.99	12,114	7,382	216	12,174	7,369	276	12,292	7,296	192
Standard III	[1–20]	1	15,142	15,142	9	15,142	15,142	12	15,142	15,142	19
		0.995	10,610	10,545	20	10,633	10,561	35	10,651	10,514	50
		0.99	7,546	7,385	18	7,566	7,388	36	7,605	7,316	$\underline{37}$
NF	[1–20]	1	15,142	15,142	6	15,142	15,142	22	15,142	15,142	43
		0.995	10,545	10,545	$\underline{5}$	10,561	10,561	$\underline{18}$	10,514	10,514	$\underline{38}$
		0.99	7,386	7,385	$\underline{5}$	7,389	7,388	$\underline{19}$	7,317	7,316	41

Table 29.6 Comparison of formulations in terms of sparsity—density = 25 %, $n = 30$

		Crashing By					
		25 %		50 %		75 %	
Formulation	Interval	Density	Nonzeros	Density	Nonzeros	Density	Nonzeros
Standard I	[1–10]	4.9 %	14,721	4.91 %	25,807	4.9 %	34,895
	[1–20]	4.91 %	46,289	4.92 %	85,611	4.92 %	121,404
Standard II	[1–10]	2.58 %	561,481	2.19 %	1,471,546	2.03 %	2,506,280
	[1–20]	2.27 %	4,879,556	2.0 %	14,843,152	1.89 %	28,252,590
Standard III	[1–10]	3.38 %	365,292	3.35 %	634,285	3.31 %	850,910
	[1–20]	3.41 %	2,199,423	3.37 %	4,029,268	3.33 %	5,659,280
NF	[1–10]	0.05 %	24,002	0.03 %	50,616	0.02 %	73,282
	[1–20]	0.01 %	89,521	0.01 %	186,146	0.007 %	275,646

Table 29.7 Comparison of formulations in terms of sparsity—density = 75 %, n = 30

		Crashing by					
		25 %		50 %		75 %	
Formulation	Interval	Density	Nonzeros	Density	Nonzeros	Density	Nonzeros
Standard I	[1–10]	5.07 %	19,091	5.07 %	33,339	5.07 %	44,840
	[1–20]	5.09 %	59,214	5.09 %	109,453	5.09 %	154,927
Standard II	[1–10]	2.58 %	814,583	2.22 %	2,114,146	2.07 %	3,565,657
	[1–20]	2.35 %	7,019,835	2.09 %	21,301,330	1.97 %	40,298,770
Standard III	[1–10]	3.37 %	539,213	3.34 %	932,693	3.31 %	1,243,882
	[1–20]	3.41 %	3,223,830	3.37 %	5,893,760	3.34 %	8,253,887
NF	[1–10]	0.05 %	29,422	0.03 %	61,024	0.02 %	87,435
	[1–20]	0.01 %	106,230	0.009 %	219,527	0.006 %	323,636

Recall that due to the maximum time limit imposed when solving the problem instances, Standard I and Standard II formulations did not always obtain an optimal solution. Following the original experiments we provide Table 29.8, which summarizes for each scenario of problem parameters and payment scheme, the number of integer programs that could not be solved within the imposed time limit by the standard formulations. Note that each problem instance is solved for each type of payment scheme (LSP or ETI) with three different crashing percentages. So, summing across the ten different problem instances, we obtain a total of 30 different integer programs for each type of payment scheme that the computational results summarized in Tables 29.2, 29.3, 29.4, and 29.5 were run against. For cases with unsolved problems the gap (ratio of difference in objective function values between that obtained by the respective formulation and the optimal solution) is also reported. A dash in the table indicates that a feasible solution was not obtained in a majority of the corresponding problem instances within the computational time limit. It is observed that a problem with the ETI payment scheme appears to be less difficult to solve for the standard formulations than one with the LSP payment scheme. From Tables 29.2, 29.3, 29.4, and 29.5 it can be seen that this is likely the result of tighter LP relaxations for the problems based on the ETI payment scheme.

We also updated the original tests to determine the limits of problem sizes that can be solved in reasonable time by the best performing formulations—NF and Standard III. Tables 29.9 and 29.10 report the summary of average computational times for problem instances with number of activities $n = 90$, activity durations drawn from the interval [1,20], $\beta = 0.99$, and crashing of 75 % (the most difficult combination of problem parameters). From the tables it is apparent that $n = 90$ represents approximately the limit on the problem size which NF and Standard III can handle reasonably. Between NF and Standard III, NF clearly appears to be the preferred choice.

Table 29.8 Effect of time limit on solution for standard formulations; entries indicate number of unsolved problems, and if non-zero, also the associated gap

Problem parameters	Payment scheme	Number unsolved Standard I	Number unsolved Standard II
Density $= 25\,\%, n = 30$	LSP	0	0
$\beta = 1$, Interval[1, 10]	ETI	0	0
Density $= 25\,\%, n = 30$	LSP	0	18, 9.9 %
$\beta = 0.995$, Interval[1, 10]	ETI	0	1, 0 %
Density $= 25\,\%, n = 30$	LSP	0	21, 11.5 %
$\beta = 0.99$, Interval[1, 10]	ETI	2, 0 %	3, 1 %
Density $= 25\,\%, n = 30$	LSP	0	–
$\beta = 1$, Interval[1, 20]	ETI	0	–
Density $= 25\,\%, n = 30$	LSP	23, 0.1 %	–
$\beta = 0.995$, Interval[1, 20]	ETI	16, 0.1 %	–
Density $= 25\,\%, n = 30$	LSP	18, 8.1 %	–
$\beta = 0.99$, Interval[1, 20]	ETI	17, 5.3 %	–
Density $= 75\,\%, n = 30$	LSP	0	0
$\beta = 1$, Interval[1, 10]	ETI	0	0
Density $= 75\,\%, n = 30$	LSP	1, 0 %	21, 23.4 %
$\beta = 0.995$, Interval[1, 10]	ETI	0	0
Density $= 75\,\%, n = 30$	LSP	3, 0 %	20, 42.7 %
$\beta = 0.99$, Interval[1, 10]	ETI	0	8, 3.6 %
Density $= 75\,\%, n = 30$	LSP	0	–
$\beta = 1$, Interval[1, 20]	ETI	0	–
Density $= 75\,\%, n = 30$	LSP	27, 0.8 %	–
$\beta = 0.995$, Interval[1, 20]	ETI	17, 0.1 %	–
Density $= 75\,\%, n = 30$	LSP	20, 0.3 %	–
$\beta = 0.99$, Interval[1, 20]	ETI	22, 4.5 %	–

Table 29.9 Performance of NF and Standard III models on large and difficult problem instances: $n = 90$, interval [1,20], $\beta = 0.99$, crashing by 75 %, density = 25 %

Formulation	Payment scheme	LP	LP	t_{cpu}
NF	LSP	−1,057	−2,604	1,025
Standard III	LSP	−1,417	−2,604	1,985
NF	ETI	15,911	15,987	198
Standard III	ETI	16,636	15,987	1,106

Table 29.10 Performance of NF and Standard III models on large and difficult problem instances: $n = 90$, interval [1,20], $\beta = 0.99$, crashing by 75 %, density = 75 %

Formulation	Payment scheme	LP	IP	t_{cpu}
NF	LSP	−583	−3,517	1,568
Standard III	LSP	−2,499	−3,517	2,407
NF	ETI	20,132	20,129	279
Standard III	ETI	20,581	20,129	2,482

29.5 Conclusions

This chapter reviewed the literature on the discrete time-cost tradeoff problem (DTCTP). It then explored four formulations for solving problem $MPS\infty|prec, \bar{d}\,|\,f$, which can capture the importance of including financial objective functions in time-cost tradeoff project scheduling. The empirical tests performed in Szmerekovsky and Venkateshan (2012) were updated using the current version of CPLEX. Through empirical testing the results of Szmerekovsky and Venkateshan (2012) were reaffirmed, showing that NF efficiently solves problem instances with up to 90 activities and outperforms Standard I, Standard II, and Standard III. The performance of NF is due to the tightness of the LP relaxation, the sparseness of the constraint matrix, and the reduced number of binary constraints.

References

Akkan C, Drexl A, Kimms A (2005) Network decomposition-based benchmark results for the discrete time-cost tradeoff problem. Eur J Oper Res 165(2):339–358

Bey RB, Doersch RH, Patterson JH (1981) The net present value criterion: its impact on project scheduling. Proj Manag Q 12(2):35–45

Brucker P, Drexl X, Möhring R, Neumann K, Pesch E (1999) Resource-constrained project scheduling: notation, classification, models and methods. Eur J Oper Res 112(1):3–41

Crowston WB (1970) Decision CPM: network reduction and solution. Oper Res Q 21:435–452

Crowston WB, Thompson GL (1967) Decision CPM: a method for simultaneous planning, scheduling, and control of projects. Oper Res 15:407–426

Dayanand N, Padman R (1998) On payment schedules in client-contractor negotiations in projects: an overview of the problem and research issues. In: Węglarz J (ed) Project scheduling: recent models, algorithms and applications. Kluwer Academic, Boston

Dayanand N, Padman R (2001) Project contracts and payment schedules: the client's problem. Manag Sci 47(12):1654–1667

De P, Dunne EJ, Ghosh JB, Wells CE (1995) The discrete time-cost tradeoff problem revisited. Eur J Oper Res 81(2):225–238

De P, Dunne EJ, Ghosh JB, Wells CE (1997) Complexity of the discrete time-cost tradeoff problem for project networks. Oper Res 45(2):302–306

Deinko VG, Woeginger GJ (2001) Hardness of the approximation of the discrete time-cost tradeoff problem. Oper Res Lett 29:207–210

Demeulemeester EL, Herroelen W (2002) Project scheduling: a research handbook. Kluwer Academic, Boston

Demeulemeester EL, Herroelen WS, Elmaghraby SE (1996) New computational results on the discrete time/cost tradeoff problem in project networks. Eur J Oper Res 88(1):50–68

Demeulemeester EL, De Reyck B, Foubert B, Herroelen WS, Vanhoucke M (1998) Optimal procedures for the discrete time/cost tradeoff problem in project networks. J Oper Res Soc 49(11):1153–1163

Drezet L-E (2008) RCPSP with financial costs. In: Artigues C, Demassey S, Néron E (eds) Resource-constrained project scheduling: models, algorithms, extensions and applications, Chap. 14. Wiley, Hoboken

Goldratt EM (1997) Critical chain. North River Press, Great Barrington

Grigoriev A, Woeginger GJ (2004) Project scheduling with irregular costs: complexity, approximability, and algorithms. Acta Inform 41(2/3):83–97

Hadjiconstantinou E, Klerides E (2010) A new path-based cutting plane approach for the discrete time-cost tradeoff problem. Comput Manag Sci 7:313–336

Hafizoğlu AB, Azizoğlu M (2010) Linear programming based approaches for the discrete time/cost trade-off problem in project networks. J Oper Res Soc 61(4):676–685

Harvey RT, Patterson JH (1979) An implicit enumeration algorithm for the time/cost tradeoff problem in project network analysis. Found Contr Eng 4:107–117

Hindelang TJ, Muth JF (1979) A dynamic programming algorithm for decision CPM networks. Oper Res 27:225–241

Hochbaum DS (2004) Selection, provisioning, shared fixed costs, maximum closure, and implications on algorithmic methods today. Manag Sci 50(6):709–723

Icmeli O, Erenguc SS, Zappe CJ (1993) Project scheduling problem: a survey. Int J Oper Prod Man 13(11):80–91

Kolisch R, Sprecher A (1996) PSPLIB: a project scheduling problem library. Eur J Oper Res 96(1):205–216

Meyer WL, Shaffer LR (1965) Extending CPM for multiform project time-cost curves. J Constr Div-ASCE 91:45–65

Möhring RH, Schulz AS, Stork F, Uetz M (2001) On project scheduling with irregular starting time costs. Oper Res Lett 28(4):149–154

Möhring RH, Schulz AS, Stork F, Uetz M (2003) Solving project scheduling problems by minimum cut computations. Manag Sci 49(3):330–350

Panagiotakopoulos D (1977) A CPM time-cost computational algorithm for arbitrary activity cost functions. INFOR 15:183–195

Russell AH (1970) Cash flows in networks. Manag Sci 16(5):357–373

Skutella M (1998) Approximation algorithms for the discrete time-cost tradeoff problem. Math Oper Res 23:992–1020

Szmerekovsky JG (2005) The impact of contractor behavior on the client's payment scheduling problem. Manag Sci 51(4):629–640

Szmerekovsky JG, Vairaktarakis G (2006) Maximizing project cash availability. Nav Res Log 53(4):272–284

Szmerekovsky JG, Venkateshan P (2012) An integer programming formulation for the project scheduling problem with irregular time-cost tradeoffs. Comp and Oper Res 39(7):1402–1410

Ulusoy G, Cebelli S (2000) An equitable approach to the payment scheduling problem in project management. Eur J Oper Res 127(2):262–278

Vanhoucke M, Debels D (2007) The discrete time-cost tradeoff problem: extensions and heuristic procedures. J Sched 10:311–326

Chapter 30
Generalized Discrete Time-Cost Tradeoff Problems

Mario Vanhoucke

Abstract Time-cost tradeoffs have been extensively studied in the literature since the development of the critical path method. Recently, the discrete version of the problem formulation has been extended to various practical assumptions, and solved with both exact and heuristic optimisation procedures, as described in Vanhoucke and Debels (J Sched 10:311–326, 2007).
In this chapter, an overview is given of four variants of the discrete time-cost tradeoff problem and a newly developed electromagnetic meta-heuristic (EM) algorithm to solve these problems is presented. We extend the standard electromagnetic meta-heuristic with problem specific features and investigate the influence of various EM parameters on the solution quality. We test the new meta-heuristic on a benchmark set from the literature and present extensive computational results.

Keywords Discrete time-cost tradeoff • Electromagnetic algorithm • Net present value • Project scheduling • Time-switch constraints • Work continuity constraints

30.1 Introduction

Time-cost tradeoffs in projects have been the subject of research since the development of the critical path method, and have led to problem descriptions under various assumptions. While the early endeavours mainly focused on a linear non-increasing relation between activity duration and cost (Ford and Fulkerson 1962; Fulkerson 1961; Kelley 1961; Kelley and Walker 1959; Siemens 1971; and Elmaghraby and Salem 1984), researchers gradually extended this basic problem

M. Vanhoucke (✉)
Faculty of Economics and Business Administration, Ghent University, Gent, Belgium

Technology and Operations Management Area, Vlerick Business School, Gent, Belgium

Department of Management Science and Innovation, University College London, London, United Kingdom
e-mail: mario.vanhoucke@ugent.be; mario.vanhoucke@vlerick.com; m.vanhoucke@ucl.ac.uk

C. Schwindt, J. Zimmermann (eds.), *Handbook on Project Management and Scheduling Vol. 1*, International Handbooks on Information Systems,
DOI 10.1007/978-3-319-05443-8_30

type to concave (Falk and Horowitz 1972), convex (Elmaghraby and Salem 1982; Kapur 1973; Lamberson and Hocking 1970; Siemens and Gooding 1975) or discrete time-cost relations (Akkan et al. 2005; Azaron et al. 2005; Billstein and Radermacher 1977; Cohen et al. 2007; Crowston 1970; Crowston and Thompson 1967; De et al. 1995, 1997; Demeulemeester and Herroelen 1996; Demeulemeester et al. 1998; Elmaghraby and Kamburowsky 1992; Hazır et al. 2010a,b, 2011; Hindelang and Muth 1979; Ke et al. 2009, 2012; Klerides and Hadjiconstantinou 2010; Mokhtari et al. 2011; Patterson 1979; Pour et al. 2010; Robinson 1975; Skutella 1998; Tareghian and Taheri 2006, 2007; Vanhoucke 2005; Vanhoucke and Debels 2007; Vanhoucke et al. 2002; Wiest and Levy 1977; Sonmez and Bettemir 2012).

The specific problem addressed in this chapter is the discrete time-cost tradeoff problem and involves the selection of a set of execution modes (i.e., time-cost pairs for each activity) in order to achieve a certain objective. The objective of this problem can be threefold. The so-called deadline problem involves the scheduling of project activities in order to minimise the total cost of the project while meeting a given deadline. The budget problem aims at minimising the project duration without exceeding a given budget. A third objective involves the generation of the complete efficient time-cost profile over the set of feasible project durations. Due to its practical relevance, the discrete time-cost tradeoff problem has been the main subject of research, leading to various solution approaches for various variants of the problem. In this chapter, we study the discrete time-cost tradeoff problem (DTCTP) under four different assumptions, and develop a meta-heuristic solution approach for the deadline version of the problem based on the principles of electromagnetic optimisation (Birbil and Fang 2003).

The outline of this chapter is as follows. In Sect. 30.2, we briefly present four versions of the discrete time-cost tradeoff problem and give an overview of previous research efforts in the literature. Section 30.3 explains the building blocks of our electromagnetic meta-heuristic for the four versions of the problem in detail. In Sect. 30.4, we illustrate our novel approach on a problem example. Section 30.5 reports computational experience on a benchmark dataset. Section 30.6 draws overall conclusions and highlights future research avenues.

30.2 Problem Formulation

We assume that a project is represented by an activity-on-arc network $G = (V, E)$, where the set of nodes, V, represents network events and the set of arcs, E, represents the activities of the project. The duration p_{ijm} of an activity $(i, j) \in E$ is a discrete, non-increasing function of the amount of a single nonrenewable resource (money, c_{ijm}) allocated to it. The tuple (p_{ijm}, c_{ijm}) is referred to as a mode, and we assume that each activity has $|\mathscr{M}_{ij}|$ modes with $p_{ij1} < p_{ij2} < \ldots < p_{ij|\mathscr{M}_{ij}|}$ and $c_{ij1} > c_{ij2} > \ldots > c_{ij|\mathscr{M}_{ij}|}$. We assume that the project is subject to a pre-specified project deadline $\overline{d}$. A solution can be represented by a selected set of modes (p_{ijm}, c_{ijm}) (with $m \in \mathscr{M}_{ij}$) for each activity (i, j) such that a certain objective is optimised. In the present chapter, we study four versions of the discrete time-cost tradeoff problem which differ in their objective function or their problem characteristics. In Sect. 30.2.1, we briefly review the specific differences between the four versions of the DTCTP. Section 30.2.2 briefly discusses the relevance of the four problem types in research environments and in practice.

30.2.1 Problem Descriptions

This section gives a summary of the numerous research efforts from literature for the four variants of the discrete time-cost tradeoff problem and presents the classification codes according to the classification schemes of Herroelen et al. (1999) and/or Brucker et al. (1999).

The basic *discrete time-cost tradeoff problem* (DTCTP) involves the scheduling of project activities by selecting a mode m for each activity (i, j) in order to minimise the total cost $\sum_{(i,j)\in E} c_{ijm}$ of the project. The problem can be classified as $1, T|cpm, \delta_n, disc, mu|av$ or $MPS\infty|prec, \overline{d}|\Sigma c_i(p_i)$, and has been solved by various authors as discussed in Sect. 30.1.

The *discrete time-cost tradeoff problem with time-switch constraints* (DTCTP-*tsc*) is very similar to the DTCTP, but assumes that activities are forced to start in a specific time interval and are down in some specified rest interval. Time-switch constraints have been introduced by Yang and Chen (2000) as a logical extension of the analyses and achievements of Chen et al. (1997), and have been incorporated in the DTCTP as a special type of constraints in which each activity follows one of three possible work/rest patterns. Firstly, if an activity follows a *day*-pattern, it can only be executed during day time, from Monday till Friday. Secondly, an activity follows a $d\&n$-pattern if it can be executed during the day or night, from Monday till Friday. Finally, a *dnw*-pattern means that the corresponding activity can be in execution every day or night and also during the weekend. The problem can be classified as $1, T|tsc, cpm, \delta_n, disc, mu|av$, and has been solved to optimality by Vanhoucke et al. (2002) and Vanhoucke (2005).

The *discrete time-cost tradeoff problem with work continuity constraints* (DTCTP-*wc*) also minimises the total cost of the schedule, that consists of the sum of both the direct activity costs (resulting from the selection of a mode for each activity) and work continuity cost for each activity group $E' \subset E$. Work continuity constraints have been defined by El-Rayes and Moselhi (1998) in order to model the timely movement of project resources and hence to maintain continuity of work. The work continuity cost represents the cost of the use of resources during the execution of activity group E'. This cost can be minimised by minimising the time-span between the first activity (start of use of resources) and last activity (release of resources) of the activity group E'. Vanhoucke and Debels (2007) have shown that the DTCTP-*wc* can easily be transformed into the basic DTCTP by adding two extra arcs per work continuity resource group, and hence, the problem can be classified as $1, T|cpm, \delta_n, disc, mu|av$. The problem has been solved to optimality by Reda (1990).

The *discrete time-cost tradeoff problem with net present value optimisation* (DTCTP-*npv*) involves the scheduling of project activities in order to maximise the net present value of the project subject to precedence relations. In addition to the cost c_{ijm} of mode m of activity (i, j), we assume that positive cash flows are associated to the project events (nodes). We use $c_j^{F+} \geq 0$ to denote the positive payment received at the realisation of event j. Note that we assume that, without loss of generality, each cash outflow c_{ijm} occurs at the completion of each activity. This is a reasonable assumption, since it is always possible to calculate a terminal value of each activity's cash flow upon completion by compounding the associated cash flow to the end of the activity. Consequently, the net (positive or negative) cash flow of each event j equals $c_j^F = c_j^{F+} - \sum_{(i,j) \in E_j^-} c_{ijm}$, with E_j^- denoting the set of all incoming arcs of event j. The problem can be classified as $1, T|cpm, \delta_n, disc, mu|npv$ or $MPS\infty|prec, \overline{d}|\Sigma c_i^F \beta^{C_i}$. An exact method for this problem has been presented by Erengüç et al. (1993). A heuristic method is proposed by Vanhoucke and Debels (2007).

30.2.2 Practical Relevance

The development of a meta-heuristic algorithm is based on and inspired by various real-life projects where time-cost tradeoffs are a matter of degree. Most of these applications have been described elsewhere such that a detailed description needs no repetition here.

The use of *time/switch constraints* is straightforward and boils down to the presence of rest and work periods in daily work schedules. Although the specific choice of three work/rest patterns does not exclude more general problem descriptions, it is based on a practical construction project in the field of a water purification company in Belgium (Europe). In this project, a number of filtering machines have to be installed to purify water towers and make use of one or more filtering bags. The more filtering bags are used at the same time, the lower the duration of the particular job but the larger the execution cost. Some of these machines can work without human intervention (a dnw-pattern) or, in other cases, with a human intervention once in a while, such as control operations (since these activities only require one person once in a while, they follow a $d\&n$-pattern). Of course, certain activities of the project require a whole team and can therefore only be executed during the day (day-pattern activities, see Vanhoucke et al. 2002).

The practical relevance of *work continuity constraints* has been extensively described in literature. In Vanhoucke (2006), a literature overview and various practical applications of work continuity constraints in project scheduling have been given, among which a huge and complex tunnel construction project in the Netherlands, Europe (see www.westerscheldetunnel.nl). Optimisation of work continuity could lead to enormous cost savings in the schedule for a large freezing machine needed to bore the links between the two lanes of the tunnel. The scheduling details of this tunnel project are summarised in Vanhoucke (2007).

The use and practical value of *net present value optimisation* has been extensively described in Herroelen et al. (1997). Vanhoucke and Demeulemeester (2003) have illustrated the optimisation of the net present value in a capacity expansion and construction project at a water purification company in Belgium, Europe.

30.3 Electromagnetic Optimisation

In this section, the electromagnetic meta-heuristic procedure to solve the four variants of the DTCTP is explained in detail. The EM approach follows the same generic approach as the original EM algorithm of Birbil and Fang (2003) and can be displayed in pseudo-code as follows:

```
Algorithm EM DTCTP-extensions
   create initial population
   while stop criterion not met
      compute forces
      apply forces
      local search
   endwhile
```

In the remainder of this section, we explain the specific sub-routines of our EM algorithm for the four different versions of the DTCTP.

Create initial population: The algorithm creates an initial population containing σ_{pop} population elements. Each population element, or a so-called solution point x^s ($s = 1, \ldots, \sigma_{pop}$) represents a vector of activity modes that need to be transformed into a project schedule. The EM algorithm randomly assigns values to each element x^s_{ij} ($(i, j) \in E$) of vector x^s between 1 and $|\mathscr{M}_{ij}|$, and transforms the resulting vector into a project schedule using the following schedule-generation schemes: The DTCTP and the DTCTP-*wc* can be efficiently scheduled by determining the earliest completion time of each activity, using the traditional forward pass critical path calculations (problem $cpm|C_{max}$). The DTCTP-*tsc* can be efficiently scheduled by the adapted forward pass critical path calculation method of Yang and Chen (2000) (problem $cpm, tsc|C_{max}$). The DTCTP-*npv* reduces to the well-known max-*npv* problem (problem $cpm|npv$), which can be efficiently solved by the recursive search procedure of Vanhoucke et al. (2001). In order to use this activity-on-node (AoN) procedure, the AoA project network needs to be considered as an AoN network: each event j is then an activity with zero duration and a cash flow c^F_j and the arcs represent the precedence relations with time lags $\delta_{ij} = p_{ijm}$.

Compute forces: This sub-routine calculates charges for each solution point as well as a total force exerted on each solution point by all other solution points, following the principles of Coulomb's law. The charge of each solution point x^s depends on its objective function value $f(x^s)$ in relation to the objection function value of the current best point x^{best} in the population, with better objective function values resulting in higher charges. The charge q^s of solution point x^s is determined according to Eq. (30.1). Note that the differences in objective functions are measured as absolute values in order to cope with both minimisation and maximisation problems. The formula uses $|E|$ as the number of arcs in the project network.

$$q^s = \exp\left(-|E|\frac{|f(x^s) - f(x^{best})|}{\sum_{r=1}^{\sigma_{pop}} |f(x^r) - f(x^{best})|}\right) \tag{30.1}$$

Next, the algorithm calculates a set of force vectors F^s $(s = 1, \dots, \sigma_{pop})$ that are exerted on the corresponding solution point x^s, as follows:

$$F^s = \sum_{r=1:r\neq s}^{\sigma_{pop}} \begin{cases} (x^r - x^s)\frac{q^s q^r}{||x^r - x^s||^2}, \text{ if } f(x^r) < f(x^s) \\ (x^s - x^r)\frac{q^s q^r}{||x^r - x^s||^2}, \text{ otherwise} \end{cases} \tag{30.2}$$

for problems DTCTP, DTCTP-*tsc*, and DTCTP-*wc* and

$$F^s = \sum_{r=1:r\neq s}^{\sigma_{pop}} \begin{cases} (x^r - x^s)\frac{q^s q^r}{||x^r - x^s||^2}, \text{ if } f(x^r) > f(x^s) \\ (x^s - x^r)\frac{q^s q^r}{||x^r - x^s||^2}, \text{ otherwise} \end{cases} \tag{30.3}$$

for problem DTCTP-*npv*.

The general philosophy of Coulomb's law is that the total force exerted on a solution point by all other solution points is inversely proportional to the distance between the solution points and directly proportional to the product of their charges, as shown in Eqs. (30.2) and (30.3). The equations have been modelled such that a point with a relatively good objective function value will attract the other one, whereas a point with the inferior objective value repels the other. The distance between two solution points is measured by the sum of the absolute values of the component-wise differences between the mode number of identical activities. This distance measure is normalised (denoted by the symbol $\|\ \|$) by dividing it by the maximum of all distances between each pair of solution points, in order to lie in the interval [0, 1].

Apply forces: A new population is generated by moving each population element into a direction dictated by the forces. The imposed force is normalised, by dividing it by the maximal force over all dimensions (i.e., the total number of arcs) for the population element, and therefore only identifies the direction of the move. The magnitude of the move is determined by a randomly selected parameter λ generated from a uniform distribution on interval [0, 1] (in analogy with Birbil and Fang 2003) and by the number of modes for each vector element x_{ij}^s $((i, j) \in E)$ of vector x^s.

$$x_{ij}^s = \begin{cases} x_{ij}^s + \lambda \frac{F_{ij}^s}{F_{max}^s}(|\mathscr{M}_{ij}| - x_{ij}^s), \text{ if } F_{ij}^s > 0 \\ x_{ij}^s + \lambda \frac{F_{ij}^s}{F_{max}^s}(x_{ij}^s - 1), \qquad \text{otherwise} \end{cases} \tag{30.4}$$

Since the x_{ij}^s vector elements are discrete numbers, while the EM move assumes a fractional solution space, the calculated x_{ij}^s are rounded up (fractional value above 0.5) or rounded down (fractional value below or equal to 0.5) to prevent that some mode numbers are seldom or never chosen (e.g., without the rounding up mechanism, the mode number $|\mathscr{M}_{ij}|$ will only be chosen (i.e., $x_{ij}^s = |\mathscr{M}_{ij}|$) when $\lambda = 1$ and $F_{ij}^s = F_{max}^s$).

Local search: The generation of new solution points is followed by a local search procedure, which explores the immediate (Euclidian) neighbourhood of individual points. The EM procedure presented in this chapter makes use of two sequential heuristic procedures after the generation of each new population, as follows:

Local search 1: repair function: Any infeasible solution point x^s for which the project duration exceeds the project deadline is subject to a heuristic repair function, which transforms infeasible solution points into feasible ones. The repair function iteratively crashes activity durations until the project duration is smaller than the pre-specified project deadline. The project activities can be selected according to various heuristic rules:

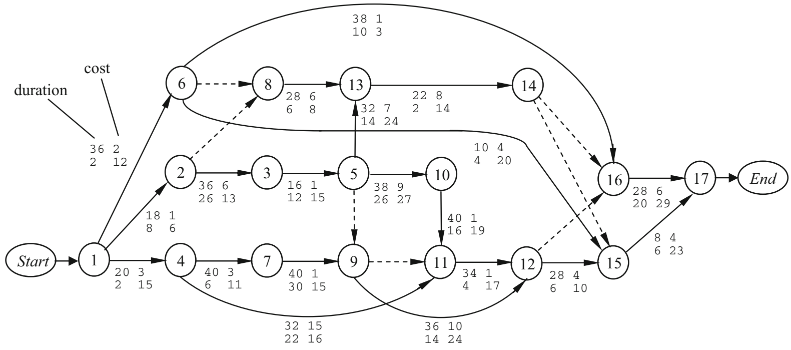
- Random selection of activity/mode combinations (RAN): the repair method iteratively selects project activities at random and crashes their durations to their neighbourhood modes until a feasible solution is obtained.
- Lowest cost per time unit (LCT): all project activities are ranked according to their cost increase per time unit when crashing the respective activity duration to its neighbourhood mode. The repair method selects the activity with the lowest cost increase first, and updates the cost increase ranking each time an activity duration has been crashed.
- Lowest absolute cost difference (LAC): this method is similar to the LCT method, but ranks all project activities according to their lowest absolute cost difference between the activity's current and neighbourhood mode.

Local search 2: improvement search: Feasible solution points can possibly be improved by increasing activity durations (and hence, decreasing the activity cost), resulting in an improved objective function value. A truncated recursive search procedure randomly ranks all activity/mode combinations and searches for improved schedules based on a truncated dynamic programming heuristic of Vanhoucke and Debels (2007). This heuristic search enumerates a subset of all possible combinations to increase activity durations following the ranking of the randomly generated activity/mode list, and is truncated after a very small pre-defined number of backtracking steps. In the present study, the search is truncated after five backtracking steps, to guarantee a very fast and efficient improvement search.

Stop criterion: The length of a search is determined by a pre-defined stop criterion, which is a function of the number of iterations and the size of the population. More precisely, the length of the search is defined as the product of the population size and the maximum number of iterations (i.e., $\sigma_{pop} \times n_{iter}$), which serves as a measure for the estimated number of generated schedules during the complete search. Indeed, since the electromagnetic heuristic completely replaces all population elements at each iteration run, this product serves as a reliable estimate for the total number of generated schedules. This approach also allows the fine-tuning of the population size for identical stop criteria (i.e., varying the σ_{pop} and the n_{iter} parameters while keeping their product at a constant level).

30.4 Illustrative Example

In this section, we show the difference between the four problem types on an example network taken from Vanhoucke et al. (2002) and illustrate the philosophy of the electromagnetic meta-heuristic on a small example schedule population of the project. Each arc has two activity modes, which are displayed near the arcs. Dummy arcs are displayed as dashed arcs. We assume that the project deadline $\overline{d}$ equals 82 days. The project network and additional data are given in Fig. 30.1.



Additional information for the DTCTP-*tsc*

Arc (*i,j*)	Pattern	Arc (*i,j*)	Pattern	Arc (*i,j*)	Pattern
(1,2)	*dnw*	(5,10)	*dnw*	(10,11)	*d&n*
(1,4)	*dnw*	(5,13)	*dnw*	(11,12)	*d&n*
(1,6)	*d&n*	(6,8)	*dummy*	(12,15)	*day*
(2,3)	*d&n*	(6,15)	*d&n*	(12,16)	*dummy*
(2,8)	*dummy*	(6,16)	*dnw*	(13,14)	*day*
(3,5)	*dnw*	(7,9)	*day*	(14,15)	*dummy*
(4,7)	*dnw*	(8,13)	*day*	(14,16)	*dummy*
(4,11)	*dnw*	(9,11)	*dummy*	(15,17)	*day*
(5,9)	*dummy*	(9,12)	*d&n*	(16,17)	*dnw*

Additional information for the DTCTP-*wc*

Activity group $E' = \{(3,5), (5,10), (9,12), (10,11), (11,12)\}$ is subject to work continuity constraints with a cost 6 per time unit of use.

Additional information for the DTCTP-*npv*

Event *j*	$c_j^{F_+}$	**Event *j***	$c_j^{F_+}$	**Event *j***	$c_j^{F_+}$
1	0	7	10	13	6
2	10	8	12	14	5
3	10	9	11	15	20
4	1	10	14	16	15
5	3	11	10	17	20
6	15	12	3		

Fig. 30.1 An AoA project network example (source of network: Vanhoucke et al. 2002)

Table 30.1 Solutions for the example project of Fig. 30.1

	Activity cost	Work continuity		Net present value	Project duration
		E' duration	E' cost		
DTCTP	**127**	[17,68]	306	13.09	82 days
DTCTP-*wc*	153	[27,58]	**186**	13.83	76 days
DTCTP-*npv*	153	[22,72]	300	**18.39**	82 days
DTCTP-*tsc*	**199**	[23,64]	246	−6.36	82 days

We assume, without loss of generality, that the duration of each activity mode has been expressed in work periods of 12 h. The DTCTP versions without time-switch constraints all assume that activities can be executed during all days of the week (no weekends and no holidays), i.e., all days are working days of 12 h. The DTCTP-*tsc*, on the contrary, allows night execution (for the *d*&*n* and *dnw* patterns), and hence, can be used to execute two work periods of an activity. On the other hand, the *day* and *d*&*n* patterns exclude the possibility to execute an activity during the weekend.

Table 30.1 reports the total activity cost, the work continuity cost, the net present value, the duration of the activities of the work continuity group, and the total project duration for four optimal schedules of the example project. The table shows that the objective values of the problem types are indeed optimised (as indicated in bold). The optimal DTCTP and DTCTP-*tsc* schedules have the lowest total activity cost, the DTCTP-*wc* schedule has the lowest total cost (153 + 186), while the DTCTP-*npv* schedule has the highest net present value. Detailed results can be found in the appendix. Note that the objective function values of the DTCTP-*tsc* are separated from the rest of the table, since they cannot be compared with the values of the other problem formulations. The DTCTP-*tsc* incorporates additional time-switch constraints, which are not taken into account by the three other problem types.

Figure 30.2 shows an example translated from our C++ code of an electromagnetic move for the DTCTP on a population of three solution elements x^1, x^2, and x^3 with a total activity cost of 159, 138, and 141, respectively. The charges, the (normalised) distance matrix, and the forces are displayed in the figure. The new solution point x^4 is generated by performing the resulting move on a subset of the activity set (these activities are indicated in grey, e.g., activity (5, 10) is not part of the move) of solution point x^3. Computational results of Sect. 30.5 reveal that this outperforms the complete move on all activities. The general electromagnetic philosophy is conceptually displayed in the figure. Since x^3 lies far from x^1 and has a better objective function, the resulting move is directed away from x^1 with a small magnitude. The opposite is true for x^3 versus x^2. The new solution point x^4 has a project duration larger than the project deadline, and hence, the algorithm randomly decreases some activity durations (displayed by the grey local search area around x^4). The improvement step increases the duration of activity (13,14) within its available slack, leading to the optimal solution for the DTCTP.

Non-dummy activities	Solution points			Forces		New solution point x^4		
	x^1	x^2	x^3	F^3	$\|\|F^3\|\|$	M[1]	R[2]	I[3]
(1,2)	1	1	2	-0.037	-1.000	**1**	1	1
(1,4)	2	2	2	0.000	0.000	2	2	2
(1,6)	2	2	2	0.000	0.000	**2**	2	2
(2,3)	2	2	2	0.000	0.000	**2**	**1**	1
(3,5)	1	2	2	0.000	0.000	2	2	2
(4,7)	2	2	2	0.000	0.000	**2**	2	2
(4,11)	2	2	2	0.000	0.000	2	**1**	2
(5,10)	2	1	2	-0.037	-1.000	2	2	2
(5,13)	2	2	2	0.000	0.000	2	2	2
(6,15)	2	2	2	0.000	0.000	2	2	2
(6,16)	2	2	2	0.000	0.000	**2**	**1**	2
(7,9)	2	2	2	0.000	0.000	2	2	2
(8,13)	2	2	2	0.000	0.000	**2**	**1**	2
(9,12)	2	2	1	0.037	1.000	**2**	2	2
(10,11)	1	2	1	0.037	1.000	**2**	2	2
(11,12)	2	1	1	0.000	0.000	1	1	1
(12,15)	1	1	2	-0.037	-1.000	**1**	1	1
(13,14)	2	2	2	0.000	0.000	**2**	**1**	**2**
(15,17)	2	2	2	0.000	0.000	2	2	2
(16,17)	1	2	2	0.000	0.000	2	2	2
Activity cost:	159	138	141			120	138	127
Duration:	82	82	82			85	82	82

[1] Move: only executed on a sub-part of the activity set (see computational results), indicated in gray (activity (5,10) was not part of the move). λ has been randomly set to 0.9

[2] Repair function: randomly decreases 5 activity durations

[3] Improvement method: increases the duration of activity (13,14) within the available slack

Distances

	x^1	x^2	x^3
x^1	0	5	6
x^2	5	0	5
x^3	6	5	0

Normalized distances

	x^1	x^2	x^3
x^1	0	0.8	1
x^2	0.8	0	0.8
x^3	1	0.8	0

Charges: $q^1 = 0$, $q^2 = 1$ and $q^3 = 0.308$,

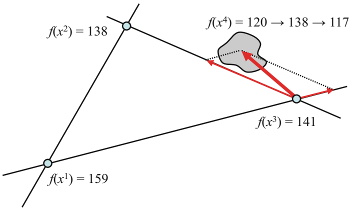


Fig. 30.2 A conceptual representation of the electromagnetic charges and forces calculation

30.5 Computational Results

We have coded the electromagnetic meta-heuristic procedure in Visual C++ version 6.0 to run on a Toshiba personal computer with a Pentium IV 2 GHz processor under Windows XP. In order to evaluate the quality of the heuristic solutions, we compare them with exact solutions for all four problem types as well as another meta-heuristic procedure of Vanhoucke and Debels (2007), which is able to cope with the four versions of the problem under study. The DTCTP and the DTCTP-*wc* instances will be solved to optimality by the procedure of Demeulemeester et al. (1998). The DTCTP-*tsc* instances will be solved by the exact procedure of Vanhoucke (2005). The exact procedure for the DTCTP-*npv* has been linked with the industrial LINDO optimisation library version 5.3 (Schrage 1995) in order to rely on an adapted version of the procedure of Erengüç et al. (1993). The test set used is an extended set from Demeulemeester et al. (1998) and has been previously used by Vanhoucke and Debels (2007). The parameter settings for the different problem instances are summarised along the following lines.

- DTCTP: The number of activities ranges from 10, 20, 30, 40 to 50 activities, the number of modes is fixed at 2, 4, or 6 modes or is randomly chosen between [1,3], [1,7], or [1,11] and the project deadline lies between the minimal and maximal project duration in steps of 25 %, from 0 % (minimum), 25 %, 50 %, 75 % to 100 % (maximum). Each setting contains 30 problem instances, resulting in $30 \times 5 \times 6 \times 5 = 4{,}500$ problem instances.

This dataset is extended with other parameters to test the performance of the solution procedure to solve the DTCTP-*tsc*, DTCTP-*wc*, and DTCTP-*npv*, as follows:

- DTCTP-*tsc*: Each activity belongs to a work/rest pattern, ranging from [0,0,100], [0,33,66], [0,66,33], [0,100,0], [33,0,66], [33,33,33], [33,66,0], [66,0,33], [66,33, 0] to [100,0,0] (where [x,y,z] indicates [%day pattern,%day and night pattern,%day, night and weekend pattern]). The extended set contains 4,500 × 10 = 45,000 problem instances.
- DTCTP-*wc*: We define three sizes of activity groups subject to work continuity constraints containing 25 %, 50 %, or 75 % of the original activities. The work continuity cost has been defined as low, in-between, or high as, respectively, 75 %, 100 %, and 150 % of the average total activity cost of the corresponding activity group. The extended set contains 4,500 × 3 × 3 = 40,500 problem instances.
- DTCTP-*npv*: Each event node has a certain cash inflow value, which is a function of the total cost of all incoming activities, ranging between the minimal value and the maximal value in steps of 25 %, from 0 % (minimum cost), 25 %, 50 %, 75 % to 100 % (maximum cost). The extended set contains 4,500 × 5 = 22,500 problem instances.

We test the quality of our meta-heuristic procedure in two ways. In Sect. 30.5.1, we test the influence of the various EM building blocks on the solution quality of the problem instances. In this section, we restrict our tests on a subset of all data instances, where we have selected only 50-activity networks with the number of modes randomly selected between 1 and 11. In Sect. 30.5.2, we compare our generated solutions with both exact and heuristic solutions as described earlier and test our algorithms on the complete test set as described earlier.

30.5.1 Electromagnetic Heuristic Performance

This section reports results on the influence of the length of the electromagnetic search as well as the contribution of the repair function and the improvement method on the solution quality of the four DTCTP versions. The figure compares four equal stop criteria values while varying the population size and the number of iterations for each stop criterion. The population size varies between 10 and 100 in steps of 10, keeping the $\sigma_{pop} \times n_{iter}$ product constant at a level of 1,000, 2,500, 5,000, or 10,000.

All test results show that a longer search, expressed as increasing $\sigma_{pop} \times n_{iter}$ values, leads to improved solutions, at the expense of a higher CPU time. The division between the population size and the number of iterations for equal $\sigma_{pop} \times n_{iter}$ values is less intuitively clear. The figures reveal an improving start trend for increasing σ_{pop} values, with an optimum at 70, 60, 20, and 30 for the different DTCTP versions. However, Table 30.2 shows that an increase in the population

Table 30.2 CPU times (in seconds) for the DTCTP

	σ_{pop}									
$\sigma_{pop} \times n_{iter}$	10	20	30	40	50	60	70	80	90	100
1,000	0.41	0.52	0.62	0.72	0.86	0.97	1.14	1.30	1.50	1.70
2,500	0.93	1.21	1.39	1.59	1.81	2.02	2.25	2.47	2.78	3.02
5,000	1.75	2.34	2.65	2.99	3.35	3.68	4.04	4.42	4.84	5.19
10,000	3.31	4.55	5.13	5.79	6.42	7.04	7.61	8.26	8.96	9.61

Table 30.3 Influence of the repair function and the improvement method on the solution quality

		DTCTP		DTCTP-*tsc*		DTCTP-*wc*		DTCTP-*npv*	
Improvement		No	Yes	No	Yes	No	Yes	No	Yes
Repair	No	215.90 %	1.43 %	226.20 %	1.14 %	162.57 %	4.05 %	126.24 %	5.28 %
	RAN		0.93 %		0.66 %		1.60 %		1.95 %
	LCT		0.10 %		0.10 %		0.20 %		0.00 %
	LAC		0.00 %		0.00 %		0.00 %		0.11 %

size (and a resulting decrease in the number of iterations) goes hand in hand with a larger CPU time requirement. This observation can mainly be explained by the electromagnetic calculations of distances between solutions, charges and forces. Consider, as an example, a $\sigma_{pop} \times n_{iter}$ value of 1,000. A population size of 10 and a number of iterations equal to 100 means that $10 \times 10 = 100$ distances, forces and charges need to be calculated per iteration, resulting in 100×100 calculations during the complete run. However, when the population size equals 100 and the number of iterations equals 10, 100×100 distances, charges and forces need to be calculated at each iteration, leading to 100,000 calculations during the complete run. We also observed that a higher population size leads to more repairs, which also contributes to the higher computational burden.

Due to this observed tradeoff between the solution quality and the computational burden, we have selected a population size of 30 in the remainder of this study for all DTCTP versions, regardless of the number of iterations.

The contribution of the improvement method and the repair function is analysed in Table 30.3 under a stop criterion of 999 generated schedules (a popsize of 30 and a number of iterations of 33). The table displays the solution quality measured as the average relative deviation from the best found project cost (for the DTCTP, DTCTP-*tsc*, and DTCTP-*wc*) or from the best found net present value (DTCTP-*npv*) in each cell (a cell with 0 % indicates the best found solution method).

The table shows the indispensable contribution of the improvement method on the solution quality. The results without improvement (columns with label "no") are very poor, and the repair function has no influence whatsoever on the solution quality. This can be explained by the specific implementation of the repair function, which decreases activity durations (randomly or controlled) to construct deadline feasible schedules. When this repair function is not immediately followed by the improvement method, very poor feasible solutions will be created. The contribution of the repair function in combination with the improvement method (columns with label "yes") is best for the LAC heuristic to minimise overall project costs and for the LCT heuristic to maximise the net present value.

Although not displayed in a table or a figure, we have found the best results when the forces (Eq. (30.4)) are operated on a sub-part of the schedule (see the example of Fig. 30.2). More precisely, forces are applied on each activity with a probability of only 50 %.

30.5.2 Benchmark Results

In this section, we report computational results on our benchmark set and compare the solutions of each DTCTP version with an exact and a meta-heuristic solution. The exact solutions are found by the solution approaches of Demeulemeester et al. (1998) (for the DTCTP and the DTCTP-*wc*), Vanhoucke (2005) (DTCTP-*tsc*), and Erengüç et al. (1993) (DTCTP-*npv*). We also compare the obtained solutions with the heuristic solutions of Vanhoucke and Debels (2007), who presented, to the best of our knowledge, the only heuristic solution procedure available in the open literature that can solve the four versions of the DTCTP.

Solutions obtained by exact procedures: The results for the exact solution procedure are obtained by imposing a computation time limit of 1 min. After that, the procedure stops and the solution is reported. In doing so, the obtained solution can be classified in one of the following categories: optimal solution, feasible (but not necessarily optimal) solution, or infeasible solution (i.e., no solution found). The columns with label "p_{opt}" are used to denote the percentage of problem instances for which an optimal solution has been found. The columns labelled with "p_{feas}" display the percentage of problem instances for which a feasible (but not guaranteed to be optimal) solution has been found within the pre-specified time limit of 1 min. This means that the procedure already has found one or more feasible solutions, but it is truncated after the pre-specified time. The columns with "p_{unk}" show the percentage of problem instances for which no feasible solution has been found within the pre-specified time limit of 1 min. Each problem instance belongs to one of these three categories, which are used for comparison purposes with the heuristic procedures. The column labelled "$t_{cpu}^{\o}$" contains the average CPU time (in seconds) needed to solve the problem instances.

Meta-heuristic solutions: The results found by the heuristic procedure are compared with the results of one of the three categories. The instances for which an exact solution has been found (i.e., "p_{opt}") are used to compare them with the heuristic solutions as follows. The column labelled with "p_{opt}" displays the percentage of problem instances for which the heuristic has found the optimal solution (only for the problem instances for which the exact methods have found optimal solutions). The column indicated with "$\Delta_{opt}^{\o}$" gives the average percentage of deviation from the optimal solution (only for the problem instances of column

"p_{opt}"). The problem instances with a feasible, though not necessarily an optimal solution (i.e., "p_{feas}") are analysed as follows. On the one hand, the column labelled with "p_{imp}" displays the percentage of problem instances for which the heuristic solution is better than the feasible solution found by the exact procedure (only for the problem instances of column "p_{feas}"). On the other hand, the column labelled with "p_{det}" indicates the percentage of problem instances for which the heuristic solution is worse than the feasible solution found by the exact procedure (only for the problem instances of column "p_{feas}"). The remaining fraction is then the percentage of problem instances with a solution equal to the feasible solution found. Furthermore, the columns labelled with "$\Delta^{\o}_{imp}$" display the average percentage of deviation (improvement) of the heuristic solution (only for the problem instances of column "p_{feas}"), while the columns with label "$\Delta^{\o}_{det}$" refer to the average percentage of deviation (deterioration) of the heuristic solution (only for the problem instances of column "p_{feas}").

Stop criterion: The computational tests haven been performed under various stop criteria. All runs performed by exact solution approaches have been truncated after 60 s CPU time. The meta-heuristic solution approaches make use of two classes of stop criteria: The first class of test runs has been truncated after a pre-specified number of generated schedules (rows "schedule limit"), as discussed previously. More precisely, the electromagnetic procedure has been truncated when the product $\sigma_{pop} \times n_{iter}$ has reached the 1,000 or 5,000 threshold. These solutions will be compared with the meta-heuristic solution procedure of Vanhoucke and Debels (2007) truncated after 1,000 and 5,000 generated schedules. The second class of test runs have been truncated after a pre-specified time limit of 0.1 and 0.5 s (rows "time limit").

The [·] references in Tables 30.4 and 30.5 are used to refer to the solution procedures used for testing. More precisely, the exact solutions have been obtained by [a] (Demeulemeester et al. 1998), [b] (Vanhoucke 2005), and [c] (Erengüç et al. 1993). The heuristic solutions have been obtained by various procedures under different stop criteria, as follows. The solution procedure of Vanhoucke and Debels (2007) is truncated after 1,000 generated schedules [d], after 5,000 generated schedules [e], after 0.1 s [f], or after 0.5 s [g]. The electromagnetic procedure of the current chapter has a $\sigma_{pop} = 30$ and is truncated after $n_{iter} = 1000$ [h], after $n_{iter} = 5000$ [i], after 0.1 s [j], or after 0.5 s [k].

The results of Tables 30.4 and 30.5 can be summarised as follows. Since there is no major difference between the optimal procedures for both problem types (apart from extra arcs), their results are discussed together. The table shows that the exact branch-and-bound procedure of Demeulemeester et al. (1998) can solve all problem instances for the DTCTP and the DTCTP-*wc* to optimality within the time limit of 60 s. Although the electromagnetic procedure is able to generate high-quality solutions within a small time fraction, it has already been concluded by Vanhoucke and Debels (2007) that it is not beneficial to rely on this meta-heuristic procedure to solve instances of this size. These authors have shown that even for tests on larger instances (with up to 200 activities and 50 modes) there is no need to use meta-heuristic procedures. The main reason is that the specific approach used for

Table 30.4 Comparative computational results (DTCTP and DTCPT-*tsc*)

		DTCTP				DTCTP-*tsc*			
Time limit	Exact	[a]				[b]			
	p_{opt}	100 %				89.76 %			
	p_{feas}	0 %				8.08 %			
	p_{unk}	0 %				2.16 %			
	$t^{\varnothing}_{cpu}$	0.081				7.906			
Schedule limit	Heuristic	[d]	[h]	[e]	[i]	[d]	[h]	[e]	[i]
	p_{opt}	79.09 %	94.20 %	86.22 %	96.31 %	86.80 %	97.65 %	92.37 %	98.46 %
	$\Delta^{\varnothing}_{opt}$	0.50 %	0.04 %	0.21 %	0.02 %	0.26 %	0.02 %	0.11 %	0.014 %
	p_{imp}	–	–	–	–	29.40 %	56.22 %	46.31 %	59.71 %
	$\Delta^{\varnothing}_{imp}$	–	–	–	–	0.35 %	0.73 %	0.58 %	0.79 %
	p_{det}	–	–	–	–	50.80 %	11.00 %	25.17 %	6.24 %
	$\Delta^{\varnothing}_{det}$	–	–	–	–	1.20 %	0.08 %	0.34 %	0.04 %
	$t^{\varnothing}_{cpu}$	0.049	0.195	0.244	0.824	0.079	0.273	0.388	1.158
Time limit	Heuristic	[f]	[j]	[g]	[k]	[f]	[j]	[g]	[k]
	p_{opt}	80.51 %	85.09 %	86.67 %	93.98 %	86.74 %	90.10 %	92.51 %	97.54 %
	$\Delta^{\varnothing}_{opt}$	0.856 %	0.607 %	0.239 %	0.073 %	0.380 %	0.305 %	0.115 %	0.023 %
	p_{imp}	–	–	–	–	14.91 %	18.45 %	38.78 %	51.95 %
	$\Delta^{\varnothing}_{imp}$	–	–	–	–	0.172 %	0.205 %	0.482 %	0.670 %
	p_{det}	–	–	–	–	73.79 %	67.05 %	36.30 %	15.92 %
	$\Delta^{\varnothing}_{det}$	–	–	–	–	5.171 %	3.359 %	0.653 %	0.160 %
	$t^{\varnothing}_{cpu}$	0.1	0.1	0.5	0.5	0.1	0.1	0.5	0.5

the exact branch-and-bound algorithm of Demeulemeester et al. (1998) results very quickly in truncated (heuristic) solutions that are very close to the optimal solution. Moreover, thanks to the use of an efficient lower bound calculation of Ford and Fulkerson (1962), many nodes can be evaluated in the branch-and-bound tree within a limited amount of CPU-time, and hence, the meta-heuristic procedure has no computational advantage on that aspect. Note that, for obvious reasons, the rows "p_{feas}" and "p_{unk}", the rows "p_{det}" and "p_{imp}", and their corresponding rows "$\Delta^{\varnothing}_{det}$" and "$\Delta^{\varnothing}_{imp}$" are empty.

The table shows that meta-heuristic algorithms are good alternatives to exact algorithm for both the DTCTP-*tsc* and the DTCTP-*npv*. The procedure of Vanhoucke and Debels (2007) and the newly developed electromagnetic procedure outperform the exact algorithms, both in terms of CPU time and solution quality. The "schedule limit" results also show that the electromagnetic search outperforms the Vanhoucke and Debels (2007) procedure, but at a higher CPU time expense (due to the calculation of charges, forces, and distances). For this reason, we have tested and compared their results within a time limit of 0.1 and 0.5 s. These results show that the electromagnetic procedure performs best.

Table 30.5 Comparative computational results (DTCTP and DTCPT-*tsc*)

		DTCTP-*wc*				DTCTP-*npv*			
Time limit	Exact	[a]				[b]			
	p_{opt}	100 %				49.70 %			
	p_{feas}	0 %				50.30 %			
	p_{unk}	0 %				–			
	$t^{\emptyset}_{cpu}$	0.105				32.94			
Schedule limit	Heuristic	[d]	[h]	[e]	[i]	[d]	[h]	[e]	[i]
	p_{opt}	58.70 %	84.13 %	69.25 %	89.11 %	90.79 %	91.68 %	91.92 %	92.84 %
	$\Delta^{\emptyset}_{opt}$	98.60 %	0.11 %	0.50 %	0.07 %	2.53 %	1.50 %	2.19 %	1.45 %
	p_{imp}	–	–	–	–	80.73 %	83.52 %	83.53 %	84.65 %
	$\Delta^{\emptyset}_{imp}$	–	–	–	–	107.12 %	115.84 %	113.07 %	116.86 %
	p_{det}	–	–	–	–	19.16 %	16.33 %	16.3 %	15.19 %
	$\Delta^{\emptyset}_{det}$	–	–	–	–	6.57 %	1.43 %	3.63 %	1.22 %
	$t^{\emptyset}_{cpu}$	0.075	0.33	0.363	1.465	0.570	0.606	2.869	2.281
Time limit	Heuristic	[f]	[j]	[g]	[k]	[f]	[j]	[g]	[k]
	p_{opt}	60.86 %	67.21 %	71.10 %	83.17 %	85.62 %	84.85 %	90.43 %	91.01 %
	$\Delta^{\emptyset}_{opt}$	1.78 %	1.06 %	0.48 %	0.15 %	28.52 %	4.50 %	11.91 %	2.56 %
	p_{imp}	–	–	–	–	64.83 %	72.69 %	76.97 %	80.69 %
	$\Delta^{\emptyset}_{imp}$	–	–	–	–	82.31 %	97.49 %	101.60 %	111.30 %
	p_{det}	–	–	–	–	35.11 %	27.17 %	22.90 %	19.17 %
	$\Delta^{\emptyset}_{det}$	–	–	–	–	60.27 %	10.96 %	25.13 %	5.28 %
	$t^{\emptyset}_{cpu}$	0.1	0.1	0.5	0.5	0.1	0.1	0.5	0.5

30.6 Conclusions

This chapter studied four variants for the well-known discrete time-cost tradeoff problem, and developed an electromagnetic meta-heuristic to solve the problem types. The heuristic relies on the law of Coulomb and iteratively calculates charges and forces on population elements following the principles of Birbil and Fang (2003). The generation of a schedule has been extended by a dual local search method. The first local search method repairs infeasible solutions by crashing project activities, while the second local search randomly increases activity durations of feasible project solutions within the available activity slack.

The computational results are promising and show that solutions are comparable and often better than a previously developed procedure of Vanhoucke and Debels (2007). Lower CPU time and higher solution quality have been obtained by running tests on a large dataset truncated after a pre-specified number of generated solutions or after a certain CPU time limit.

Our future research intentions lie in the development of dedicated solution approaches for the DTCTP-*tsc* and DTCTP-*npv*. While the meta-heuristic procedures presented in this chapter are rather general search procedures that can deal with all four problem types, dedicated algorithms that exploit problem specific information should allow to test and compare results on larger problem instances, for which the exact algorithm fails to provide a feasible solution.

Acknowledgements We acknowledge the financial support of the "Bijzonder Onderzoeksfonds" (BOF) for the project under contract number BOF12GOA021.

Appendix

In this appendix, more detailed results for the example project are given than summarised in Table 30.1. Table 30.6 displays the optimal mode selection and the corresponding activity costs for the four versions of the DTCTP. Grey shaded cells refer to activities that belong to the work continuity group E'. Note that some of the

Table 30.6 Optimal mode selection and the corresponding activity cost

Non-	Input		Selected mode and corresponding activity cost							
dummy	modes		DTCTP		DTCTP-*tsc*		DTCTP-*wc*		DTCTP-*npv*	
arcs	$m = 1$	$m = 2$	mode	cost	mode	cost	mode	cost	mode	cost
(1,2)	6	1	1	6	1	6	2	1	1	6
(1,4)	15	3	2	3	1	15	2	3	2	3
(1,6)	12	2	2	2	2	2	2	2	1	12
(2,3)	13	6	1	13	1	13	2	6	2	6
(2,8)	0	0	1	0	1	0	2	0	1	0
(3,5)	15	1	2	1	2	1	2	1	2	1
(4,7)	11	3	2	3	1	11	1	11	2	3
(4,11)	16	15	2	15	2	15	2	15	2	15
(5,9)	0	0	1	0	1	0	2	0	1	0
(5,10)	27	9	2	9	2	9	1	27	2	9
(5,13)	24	7	2	7	2	7	2	7	2	7
(6,8)	0	0	1	0	1	0	2	0	1	0
(6,15)	20	4	2	4	2	4	2	4	2	4
(6,16)	3	1	2	1	2	1	2	1	2	1
(7,9)	15	1	2	1	1	15	2	1	2	1
(8,13)	8	6	2	6	2	6	2	6	2	6
(9,11)	0	0	1	0	1	0	2	0	1	0
(9,12)	24	10	2	10	1	24	2	10	2	10
(10,11)	19	1	2	1	1	19	1	19	2	1
(11,12)	17	1	1	17	1	17	1	17	1	17
(12,15)	10	4	1	10	1	10	2	4	1	10
(12,16)	0	0	1	0	1	0	2	0	1	0
(13,14)	14	8	2	8	1	14	2	8	2	8
(14,15)	0	0	1	0	1	0	2	0	1	0
(14,16)	0	0	1	0	1	0	2	0	1	0
(15,17)	23	4	2	4	2	4	2	4	2	4
(16,17)	29	6	2	6	2	6	2	6	1	29
Total activity cost				127		199		153		153

Table 30.7 The event occurrence times and the net present value and work continuity cost

		Net present value and work continuity information							
event		DTCTP		DTCTP-*tsc*		DTCTP-*wc*		DTCTP-*npv*	
nodes	C_j^+	S_j	c_j^F	S_j	c_j^F	S_j	c_j^F	S_j	c_j^F
1	0	0	0	0	0	0	0	0	0
2	10	4	4	4	4	9	9	4	4
3	10	17	−3	23	−3	27	4	22	4
4	1	10	−2	1	−14	10	−2	10	−2
5	3	25	2	31	2	35	2	30	2
6	15	18	13	24	13	18	13	1	3
7	10	30	7	4	−1	13	−1	30	7
8	12	18	12	24	12	18	12	4	12
9	11	50	10	46	−4	35	10	50	10
10	14	44	5	50	5	48	−13	49	5
11	10	64	−6	60	−24	56	−24	70	−6
12	3	68	−24	64	−38	58	−24	72	−24
13	6	41	−7	64	−7	51	−7	61	−7
14	5	52	−3	66	−9	62	−3	72	−3
15	20	71	6	72	6	72	12	75	6
16	15	68	14	66	14	62	14	72	14
17	20	82	10	82	10	76	10	82	−13
net present value			13.09		−6.36		13.83		18.39
work continuity cost		306		246		186		300	

activities are dummy activities, as can be seen by the zero values for the $m = 1$ and $m = 2$ columns.

Table 30.7 displays the occurrence time of each event and the net present value and work continuity cost for the optimal mode selection of each of the four DTCTP variants. The grey cells refer to the start (3) and end (12) event nodes for the work continuity group. The columns with label "S_j" provide the occurrence time of each event node j.

References

Akkan C, Drexl A, Kimms A (2005) Network decomposition-based benchmark results for the discrete time-cost tradeoff problem. Eur J Oper Res 165:339–358

Azaron A, Perkgoz C, Sakawa M (2005) A genetic algorithm approach for the time-cost trade-off in PERT networks. Appl Math Comput 168:1317–1339

Billstein N, Radermacher F (1977) Time-cost optimization. Method Oper Res 27:274–294

Birbil S, Fang SC (2003) An electromagnetism-like mechanism for global optimization. J Global Optim 25:263–282

Brucker P, Drexl A, Möhring R, Neumann K, Pesch E. (1999) Resource-constrained project scheduling: notation, classification, models, and methods. Eur J Oper Res 112:3–41

Chen Y, Rinks D, Tang K (1997) Critical path in an activity network with time constraints. Eur J Oper Res 100:122–133

Cohen I, Golany B, Shtub A (2007) The stochastic time-cost tradeoff problem: a robust optimization approach. Networks 49:175–188

Crowston W (1970) Network reduction and solution. Oper Res Q 21:435–450

Crowston W, Thompson G (1967) Decision CPM: a method for simultaneous planning, scheduling and control of projects. Oper Res 15:407–426

De P, Dunne E, Ghosh J, Wells C (1995) The discrete time-cost tradeoff problem revisited. Eur J Oper Res 81:225–238

De P, Dunne E, Ghosh J, Wells C (1997) Complexity of the discrete time/cost trade-off problem for project networks. Oper Res 45:302–306

Demeulemeester E, Herroelen W (1996) An efficient optimal solution for the preemptive resource-constrained project scheduling problem. Eur J Oper Res 90:334–348

Demeulemeester E, De Reyck B, Foubert B, Herroelen W, Vanhoucke M (1998) New computational results on the discrete time/cost trade-off problem in project networks. J Oper Res Soc 49:1153–1163

Elmaghraby S, Kamburowsky R (1992) The analysis of activity networks under generalized precedence relations. Manag Sci 38:1245–1263

Elmaghraby S, Salem A (1982) Optimal project compression under quadratic cost functions. Appl Manag Sci 2:1–39

Elmaghraby S, Salem A (1984) Optimal linear approximation in project compression. IIE Trans 16:339–347

El-Rayes K, Moselhi O (1998) Resource-driven scheduling of repetitive activities. Constr Manag Econ 16:433–446

Erengüç S, Tufekci S, Zappe C (1993) Solving time/cost trade-off problems with discounted cash flows using generalized Benders decomposition. Nav Res Log 40:25–50

Falk J, Horowitz J (1972) Critical path problems with concave cost-time curves. Manag Sci 19:446–455

Ford L, Fulkerson D (1962) Flows in networks. Princeton University Press, Princeton

Fulkerson D (1961) A network flow computation for project cost curves. Manag Sci 7:167–178

Hazır O, Haouari M, Erel E (2010a) Discrete time/cost trade-off problem: a decomposition-based solution algorithm for the budget version. Comput Oper Res 37:649–655

Hazır O, Haouari M, Erel E (2010b) Robust scheduling and robustness measures for the discrete time/cost trade-off problem. Eur J Oper Res 207:633–643

Hazır O, Erel E, Günalay Y (2011) Robust optimization models for the discrete time/cost trade-off problem. Int J Prod Econ 130:87–95

Herroelen W, Van Dommelen P, Demeulemeester E (1997) Project network models with discounted cash flows: a guided tour through recent developments. Eur J Oper Res 100:97–121

Herroelen W, Demeulemeester E, De Reyck B (1999) A classification scheme for project scheduling problems. In: Węglarz J (ed) Project scheduling: recent models, algorithms and applications. Kluwer Academic, Dordrecht, pp 1–26

Hindelang T, Muth J (1979) A dynamic programming algorithm for decision CPM networks. Oper Res 27:225–241

Kapur K (1973) An algorithm for the project cost/duration analysis problem with quadratic and convex cost functions. IIE Trans 5:314–322

Ke H, Ma W, Ni Y (2009) Optimization models and a GA-based algorithm for stochastic time-cost trade-off problem. Appl Math Comput 215:308–313

Ke H, Ma W, Ni Y (2012) Modeling stochastic project time-cost trade-offs with time-dependent activity durations. Appl Math Comput 218:9462–9469

Kelley J (1961) Critical path planning and scheduling: mathematical basis. Oper Res 9:296–320

Kelley J, Walker M (1959) Critical path planning and scheduling: an introduction. Mauchly Associates, Ambler

Klerides E, Hadjiconstantinou E (2010) A decomposition-based stochastic programming approach for the project scheduling problem under time/cost trade-off settings and uncertain durations. Comput Oper Res 37:2131–2140

Lamberson L, Hocking R (1970) Optimum time compression in project scheduling. Manag Sci 16:597–606

Mokhtari H, Kazemzadeh R, Salmasnia A (2011) Time-cost tradeoff analysis in project management: an ant system approach. IEEE Trans Eng Manag 58:36–43

Patterson J (1979) An implicit enumeration algorithm for the time/cost trade-off problem in project network analysis. Found Control Eng 6:107–117

Pour N, Modarres M, Aryanejad M, Moghadam R (2010) The discrete time-cost-quality trade-off problem using a novel hybrid genetic algorithm. Appl Math Sci 42:2081–2094

Reda R (1990) Repetitive project modeling. J Constr Eng M ASCE 116:316–330

Robinson D (1975) A dynamic programming solution to cost/time trade-off for CPM. Manag Sci 22:158–166

Schrage L (1995) LINDO: optimization software for linear programming. LINDO Systems, Chicago

Siemens N (1971) A simple CPM time/cost trade-off algorithm. Manag Sci 17:354–363

Siemens N, Gooding C (1975) Reducing project duration at minimum cost: a time/cost trade-off algorithm. Omega Int J Manag S 3:569–581

Skutella M (1998) Approximation algorithms for the discrete time-cost tradeoff problem. Math Oper Res 23:909–929

Sonmez R, Bettemir O (2012) A hybrid genetic algorithm for the discrete time-cost trade-off problem. Expert Syst Appl 39:1428–1434

Tareghian H, Taheri S (2006) On the discrete time, cost and quality trade-off problem. Appl Math Comput 181:1305–1312

Tareghian H, Taheri S (2007) A solution procedure for the discrete time, cost and quality tradeoff problem using electromagnetic scatter search. Appl Math Comput 190:1136–1145

Vanhoucke M (2005) New computational results for the discrete time/cost trade-off problem with time-switch constraints. Eur J Oper Res 165:359–374

Vanhoucke M (2006) Work continuity constraints in project scheduling. J Constr Eng Manag ASCE 132:14–25

Vanhoucke M (2007) Work continuity optimization for the Westerscheldetunnel project in the Netherlands. Tijdschrift voor Econ Manag 52:435–449

Vanhoucke M, Debels D (2007) The discrete time/cost trade-off problem: extensions and heuristic procedures. J Sched 10:311–326

Vanhoucke M, Demeulemeester E (2003) The application of project scheduling techniques in a real-life environment. Proj Manag J 34:30–42

Vanhoucke M, Demeulemeester E, Herroelen W (2001) On maximizing the net present value of a project under renewable resource constraints. Manag Sci 47:1113–1121

Vanhoucke M, Demeulemeester E, Herroelen W (2002) Discrete time/cost trade-offs in project scheduling with time-switch constraints. J Oper Res Soc 53:741–751

Wiest J, Levy F (1977) A management guide to PERT/CPM: with GERT/PDM/DCPM and other networks. Prentice Hall, Englewood Cliffs

Yang HH, Chen YL (2000) Finding the critical path in an activity network with time-switch constraints. Eur J Oper Res 120:603–613

Index

C. Schwindt, J. Zimmermann (eds.), *Handbook on Project Management and Scheduling Vol. 1*, International Handbooks on Information Systems,
DOI 10.1007/978-3-319-05443-8