International Series in
Operations Research & Management Science

Vitaly A. Strusevich
Kabir Rustogi

# Scheduling with Times-Changing Effects and Rate-Modifying Activities

Springer

Vitaly A. Strusevich · Kabir Rustogi

# Scheduling with Times-Changing Effects and Rate-Modifying Activities

Springer

Vitaly A. Strusevich
Department of Mathematical Sciences
University of Greenwich
London
UK

Kabir Rustogi
Department of Mathematical Sciences
University of Greenwich
London
UK

and

Delhivery Pvt. Ltd.
New Delhi
India

ISSN 0884-8289 ISSN 2214-7934 (electronic)
International Series in Operations Research & Management Science
ISBN 978-3-319-39572-2 ISBN 978-3-319-39574-6 (eBook)
DOI 10.1007/978-3-319-39574-6

Library of Congress Control Number: 2016942024

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG Switzerland

# Preface

Scheduling theory, born in the middle of the 1950s, has become an established area of operations research, with numerous widely quoted books and influential surveys that cover various stages of development in scheduling or addressing a particular range of its models. Journals are published, and regular conferences are held with scheduling as the main topic. Hundreds of researchers around the world work on further advancing this branch of knowledge, and thousands of students of all levels study its aspects either as a full course or as a part of more general courses related to operations research, operations management, industrial engineering, and logistics.

As with most areas of operations research, scheduling is motivated by practical needs and its achievements are fed back into various areas of industry, service, transport, etc. Whatever the motivation or application, scheduling problems are normally formulated in terms of processing jobs on machines, with a purpose of optimizing a certain objective function. In classical deterministic machine scheduling, it is assumed that for a given job, its processing times on the machines are known and remain unchanged during the planning horizon. Although the classical scheduling models form a solid theoretical background, due to their too ideal nature, their immediate practical applications are very rare. It is not by chance that a major current trend of scheduling calls for studies of more realistic models that combine scheduling decisions with logistics decisions such as batching, transportation, and maintenance.

The models of classical scheduling are too static and do not respond to possible changes of processing conditions. In reality, actual processing times of a job may be affected by the fact that either these conditions get worse, or they may improve, or undergo some changes which affect the processing time in a less predictable, not necessarily monotone way. One aspect addressed in this book is related to the study of such time-changing effects.

The studies of scheduling problems with time-changing effects were originated by O.I. Melnikov and Y.M. Shafransky (both from Minsk, Belarus) in the late 1970s. By the mid-1990s, such studies had become a noticeable part of scheduling research, and currently, the total number of publications exceeds several thousands.

There has been a need for summarizing the obtained results, identifying most influential ones and presenting them from a unified position. Partly, such a purpose was achieved by the book "Time-Dependent Scheduling" written by S. Gawiejnowicz (Poznan, Poland) and published by Springer in 2008. The book, however, gives a systematic treatment of only one particular type of time-changing effect, under which the actual processing times of jobs depend on their start times in a schedule. There are other effects, still not covered in the monographic scheduling literature, including positional effects (the actual processing times of jobs are affected by their position in a sequence on a machine), cumulative effects (times are affected by the total value of some parameter of previously scheduled jobs), or combined effects, where a combination of the "pure" effects mentioned above are applied together.

Another reason that has motivated us to undertake the task of writing this book is a need for studying the possibility of including certain activities in a schedule that alter the processing conditions. As a simple example, imagine process jobs using a cutting tool which gradually loses its sharpness, so it takes longer to process a later scheduled job. The decision-maker may decide to stop and sharpen the tool or replace it by a new tool, and such a maintenance activity may appear to be beneficial for the overall performance. We hope that this example, simple as it is, demonstrates the need for studying the scheduling problems with time-changing effects and rate-modifying activities; the reader will find these words in the title of our book.

Our personal interest in the models that we study in this book does not have a long history. Vitaly Strusevich, among his other scheduling-related studies, was involved in several papers on scheduling with changing times, jointly written with the late V.S. Gordon (Minsk, Belarus), and C.N. Potts and J.D. Whitehead (both from Southampton, UK). Kabir Rustogi, who pursued his PhD at the University of Greenwich, London, UK, wrote his PhD thesis on this topic, with Vitaly Strusevich as the first supervisor. The thesis was awarded the Best PhD Prize of the Operational Research Society (2013). The content of that thesis along with several joint papers, some additionally co authored by H. Kellerer (Graz, Austria), has determined the content of this book to a very large extent.

It would be impossible to include into a one-volume book all relevant major results, known and new, for all scheduling systems and all objectives. We have, therefore, decided to be selective and to limit our consideration only to those models and solution methods that are sufficiently representative to be of interest to the reader and, at the same time, fall within the scope of our own research interests to reflect our own contributions to the area.

In this book, we focus on two-machine environments. Most of the presented results address scheduling problems on a single machine; however, we also consider problems on parallel machines (identical, uniform, or unrelated). Scheduling systems that involve multi stage processing, such as the flow shop, appear to be outside the scope of this book.

We also have decided to limit our consideration to two objective functions: the maximum completion time, known as the makespan, and the total completion time, in both unweighted and weighted forms. We do not discuss scheduling problems with the objective functions related to due dates, such as the maximum lateness or the total tardiness. We also do not include the models that involve assignments of due dates or due windows to jobs.

From a methodological point of view, our main goal has been to systematically present the results related to the computational complexity and approximability of the chosen range of problems. Mainly, we explore how the classical and simple scheduling models can be extended to incorporate more features related to practical needs but still remain polynomially solvable by an appropriate adaptation of the classical solution methods. For those problems that are NP-hard, so that the existence of a polynomial-time solution algorithm is unlikely, we present approximation algorithms and schemes with provable running times and accuracy. Exact methods of guided enumeration, such as branch-and-bound techniques, and heuristic procedures, e.g., local search or evolutionary algorithms, are left beyond the content of this book.

The material of this book contains 20 chapters split into three parts. We have tried to make each chapter to be a self-sufficient document, complete with its individual bibliography. Chapters 1, 6, and 12 are introductory chapters to each part; they are of a bibliographic nature, so that unlike the other chapters, the corresponding literature references are placed within the body of each of these chapters. All other chapters are accompanied with a section entitled "Bibliographic Notes," which is essentially a review that points out the sources of the material in the main body of a chapter and provides references to further reading on the relevant topic.

Part I introduces all required concepts of the classical scheduling theory and delivers reviews of methods and techniques widely used in the remaining part of this book. In all scheduling problems of this part, no time-changing effects are assumed; i.e., the processing times remain constant. Chapter 1 describes the main notions of scheduling theory, introduces notation, and gives a brief informal introduction to issues of computational complexity and approximability. Chapter 2 gives a description of the pairwise interchange argument, presents a matching algorithm for minimizing a linear form over permutations, introduces the concept of a 1-priority rule, and shows how the interchange techniques can be used to solve various classical scheduling problems. Chapter 3 reviews the techniques of solving problems under precedence constraints, in particular of minimizing a priority-generating function under series-parallel precedence constraints. Many scheduling problems reduce to solving problems of Boolean linear and nonlinear programming, and Chap. 4 gives an overview of the relevant issues. It includes algorithms for the linear assignment problem with square and rectangular cost matrices and approximation schemes for the linear knapsack problem and problems related to minimizing a non-separable quadratic function known as the half-product. Chapter 5 discusses the issues of convexity and *V*-shapeness of finite sequences, as well as presents useful facts on combinatorial counting. Thus, Part I produces a toolkit to handle enhanced scheduling models in the forthcoming parts of the book.

Part II is devoted to scheduling problems with time-changing effects. A review of all studied effects, including rationales and illustrative examples, is given in Chap. 6. The remaining chapters of Part II address single machine problems under an effect of a particular type, except the last chapter of the part, Chap. 11, which handles models with parallel machines. Positional effects are studied in Chap. 7. Start-time-dependent effects, both pure and combined with a positional effect, are analyzed in Chap. 8 (for an additive form of the effect) and in Chap. 9 (for a multiplicative form of the effect). Chapter 10 addresses single machine problems with both pure and combined cumulative effects. Most of the results on single machine models under time-changing effects are polynomial-time algorithms based on adaptation of the matching approach, priority rules, and reductions to the linear assignment problem. The respective problems with series-parallel precedence constraints are also analyzed, where appropriate. For the parallel machine models in Chap. 11, the issues of complexity and approximability are additionally considered.

The most advanced scheduling models are studied in Part III. In the most general case, we not only allow time-changing effects to affect the actual durations of the jobs, but also present the decision-maker with a range of rate-modifying activities, including but not limited to maintenance, that alter the processing conditions. A review of the relevant issues, including a generic procedure for solving most general single machine problems of the described range, is given in Chap. 12. Similar to Part II, the remaining chapters of Part III deal with a single machine environment, except the last chapter of the part, Chap. 20, which addresses the parallel machine environment. We start with simple models with compulsory maintenance activities to be introduced in a schedule that do not alter the processing conditions and either are placed in fixed time intervals (in Chap. 14) or start no later than a given deadline (in Chap. 13). For problems considered in these two chapters, along with polynomial-time algorithms, we discuss the issues of complexity and approximability. Scheduling problems with no time-changing effects but with rate-modifying periods (RMPs) to be included in a schedule are studied in Chap. 15. Fully enhanced models that allow both RMP introduction and time-changing effects are addressed in Chap. 16 (for positional effects), in Chap. 17 (for start-time-dependent effects), and in Chap. 18 (for combined effects). The main focus of these four chapters is on the adaptation of the generic procedure described in Chap. 12 for the respective problems. Single machine problems that combine a cumulative effect and a maintenance activity are considered in Chap. 19 with a purpose of developing fully polynomial approximation schemes by adapting the schemes known for relevant Boolean programming problems. For the parallel machine models, in Chap. 20, we present a generic procedure for solving enhanced problems and its adaptations for particular versions of the main model.

We have tried to provide detailed proofs to most of the statements presented, either by adapting the original proofs or by developing new proofs. The reader familiar with the basics of operations research should be able to follow the book without consulting external sources. We perceive that most of the readers will be research students of master or doctoral levels and researchers in operations research, industrial engineering, and logistics. We hope that the practitioners will find the collected results useful, especially on enhanced models that attempt to address the practical needs.

We have benefitted through stimulating discussions with many colleagues who have been ready to encourage us and to share their knowledge and expertise. Apart from our co authors mentioned above, we are grateful to J. Blazewicz (Poznan, Poland), B. Chen and V.I. Deineko (both from Warwick, UK), A.V. Kononov and S.V. Sevastianov (both from Novosibirsk, Russia), M.Y. Kovalyov and Y.M. Shafransky (Minsk, Belarus), D. Shabtay (Be'er Sheva, Israel), N.V. Shakhlevich (Leeds, UK), A. Shioura (Tokyo, Japan), and A.J. Soper (Greenwich, UK).

We are especially grateful to G. Mosheiov (Jerusalem, Israel) and S. Gawiejnowicz (Poznan, Poland), the most prominent researchers who have shaped the area of scheduling with changing times. It would be fair to say that their work has motivated us to turn to this area of scheduling. Vitaly Strusevich visited them both in 2015 during his sabbatical leave and enjoyed their hospitality, most favorable work conditions, and fruitful exchanges of ideas. It has been a privilege to join Professors Gawiejnowicz and Mosheiov in setting up the program of International Workshop on Dynamic Scheduling Problems, a satellite event of EURO 2016 in Poznan. Initiated and organized by S. Gawiejnowicz, the workshop has been the first international conference with a clear focus on scheduling problems with variable parameters and hopefully will open up a series of similar meetings.

We gratefully acknowledge support from the Department of Mathematical Sciences of the University of Greenwich, London, UK, and appreciation of our work by our colleagues there.

We wish to extend our gratitude to our families whose love, attention, and care we have always felt and enjoyed. For both of us, their understanding and help, patience, and reassurance have been invaluable.

London, UK — Vitaly A. Strusevich
New Delhi, India — Kabir Rustogi

# Contents

# About the Authors

**Dr. Vitaly A. Strusevich** is a Professor of Operational Research in the Department of Mathematical Sciences of the University of Greenwich, London, UK. He holds two PhD degrees, one from the Belarusian Academy of Sciences (1982) and the other from Erasmus University, the Netherlands (1991). He was a member of the team of six researchers awarded the State Prize of the Republic of Belarus in Science and Technology (1998), the highest award of the country of his origin. Professor Strusevich has published more than 120 papers in refereed journals and several books, including *Scheduling Theory: Multi-Stage Systems* that appeared in English in 1994. He acted as a guest editor for six special issues, and two most recent are *Journal of Scheduling* and *Annals of Operations Research*. Currently, he is an associate editor of *Omega*. He participated in organizing several conferences on scheduling, planning, and combinatorial optimization, as the main organizer or the program committee member. His main research interests are in combinatorial optimization, including scheduling theory, nonlinear Boolean programming, and optimization under submodular constraints.

**Dr. Kabir Rustogi** is currently working in the logistics industry in New Delhi, India, as an operational research scientist. Formerly, he was a Senior Lecturer of operational research in the Department of Mathematical Sciences of the University of Greenwich, London, UK. He received his PhD from Greenwich in 2013 for his work on machine scheduling with changing processing times, which also won him the prestigious O.R. Society Best PhD Award in the UK. He has published his work in several high-impact journals and is widely cited, even at the early stage of his career. His previous degrees include MSc in operational research from the University of Edinburgh and BTech in engineering physics from the Indian Institute of Technology Delhi.

# List of Figures

# List of Tables

# Part I
# Models and Methods of Classical Scheduling

# Chapter 1
# Models and Concepts of Classical Scheduling

In a typical scheduling system, a number of tasks must be performed, given certain resources for that purpose. It is required to allocate resources to tasks during suitable time slots in order to achieve a required performance. Scheduling problems arise in virtually all areas of human activity:

- **Manufacturing/production**: Components of products are to be made (tasks) on machine tools, possibly assisted by human operators (resources);
- **Computing**: Computational tasks are to be assigned to computers/processors (resources);
- **Transport**: In an airport, landings and takeoffs (tasks) compete for a runway (resource);
- **Construction**: Stages of a construction project (tasks) require heavy machinery (resource) to be hired.
- **Time-tabling**: In an educational institution, classes (tasks) must be taught by teachers (resources) in classrooms (another type of resources) during particular time periods.

This list can be continued. Most of the problems that arise in classical scheduling have common features which allow formulating and handling these problems without a reference to a particular application that has motivated the problem. Traditionally, in classical scheduling, the problems are formulated in terms of jobs to be processed on machines.

In this chapter, we provide a general background of the classical scheduling theory. We define terminology and notation that is used throughout the book and introduce some basic scheduling models that are related to this study. We briefly discuss the classification of the models, as well as the issues of the computational complexity of combinatorial optimization problems, including the concepts of exact and approximation algorithms, and of their running time and performance.

V.A. Strusevich and K. Rustogi, *Scheduling with Times-Changing Effects and Rate-Modifying Activities*, International Series in Operations Research & Management Science 243, DOI 10.1007/978-3-319-39574-6_1

The readers who want to access a wider range of scheduling models are advised to use two very successful books by Brucker (2007) and Pinedo (2016), and each of these books has held five editions in recent years. Among other books that address classical scheduling models are Conway et al. (1967), Tanaev et al. (1984, 1987), Leung (2004), and Baker and Trietsch (2009). Scheduling problems that occur in various application areas are addressed in Blazewicz et al. (2001), Pinedo (2005), and Brucker and Knust (2012). Additional aspects of scheduling theory can be found in survey papers by Graham et al. (1979), Chen et al. (1998), and Potts and Strusevich (2009).

## 1.1 Classical Scheduling Models

Typical models of classical machine scheduling can be described in the following setting. The jobs of set $N = \{1, 2, \ldots, n\}$ have to be processed on $m \geq 1$ machines $M_1, M_2, \ldots, M_m$. If a job $j \in N$ is assigned to machine $M_i$, $1 \leq i \leq m$, then the processing time of job $j$ is equal to $p_{ij}$, where we assume that all $p_{ij}$ values are non-negative integers.

A *schedule* specifies for each machine $M_i$ and each job $j$ one or more time intervals during which job $j$ is processed on machine $M_i$. The actual representation of a schedule may vary for different models. A useful visual representation of a schedule is a Gantt chart.

The main assumptions of classical machine scheduling that are adopted in all models discussed in this book are as follows:

- Each machine processes no more than one job at a time.
- No job is assigned to more than one machine a time.

A *feasible* schedule must satisfy the two assumptions stated above and possibly additional conditions laid out by a particular problem type. Typically, a scheduling problem is specified by three parameters:

- machine environment,
- job characteristics,
- an optimality criterion.

In the forthcoming sections, we consider these parameters in detail.

### *1.1.1 Machine Environment*

A machine environment is defined by the configuration in which the machines are available. Machine configurations can be broadly divided in to two classes: *single-stage* systems and *multi stage* systems.

In a single-stage system, each job requires exactly one *operation*. Such systems involve either a *single* machine or $m$ machines operating in *parallel*. In the case of parallel machines, each machine has the same function. Traditionally, three types of parallel machines are distinguished:

- *Identical parallel machines*: Each processing time is independent of the machine performing a job;
- *Uniform parallel machines*: The machines operate at different speeds;
- *Unrelated parallel machines*: The processing time of a job depends on the machine assignment.

The processing time notation $p_{ij}$ can be simplified for some of the single-stage systems, other than those with unrelated machines. The subscript $i$ is omitted if the processing time of a job $j$ is independent of the machine assignment. Thus, we write

- $p_{ij} = p_j$ either for a single machine or for identical parallel machines;
- $p_{ij} = p_j/s_i$ for uniform parallel machines, where $s_i$ denotes the *speed* of machine $M_i$.

In the main body of this book, we focus on single-stage systems only. However, for completeness, below we give a brief description of multi stage systems as well.

For a multi stage system, the processing of a job consists of several stages, i.e., a job consists of several *operations*. The three main types of classical multi stage systems are characterized by the *processing routes* of the jobs, i.e., by sequences according to which the operations of a job must or may be performed:

- *Flow shop*: All jobs have the same route given by $M_1, M_2, \ldots, M_m$;
- *Open shop*: Each job has to be processed on every machine, but the processing routes are not given in advance and finding them is part of decision making;
- *Job shop*: The most general classical system, where each job has a fixed route that may be different from the routes of other jobs; some machines may be missing in the route, and some machines may occur more than once.

There are also generalizations and combinations of the scheduling systems introduced above, e.g., the systems with some jobs having fixed routes (like in the job shop or flow shop) and some jobs with the routes that are not fixed (like in the open shop). There are also multiprocessor or hybrid variants of multi stage systems, where each stage comprises of several parallel machines.

### 1.1.2 Job Characteristics

In addition to its processing time, a job $j \in N$ may be characterized by the following parameters:

- *Weight*: The weight $w_j$ of a job $j \in N$ reflects the relative importance of the job;

- *Release date*: The release date $r_j$ specifies the time at which a job $j \in N$ becomes available for processing;
- *Due date*: The due date $d_j$ specifies the date at which a job $j \in N$ is expected to complete; completion of a job after its due date is allowed, but a penalty is incurred;
- *Deadline*: The deadline $\bar{d}_j$ specifies the time by which a job $j \in N$ must be completed; unlike the due date, a deadline is a hard constraint.

Without loss of generality, we assume that all these values are non-negative integers.

Due to various practical restrictions, some scheduling models allow *precedence constraints* to be defined over the set of jobs. If job $j$ has precedence over job $k$, then $k$ cannot start its processing until $j$ is completed. Precedence constraints are usually specified by a directed acyclic precedence graph $G$ with vertices $1, \dots, n$. There is a directed path from vertex $j$ to vertex $k$ if and only if job $j$ has precedence over job $k$.

Some scheduling models allow *preemption*: The processing of any operation may be interrupted and resumed at a later time, either on the same or on a different machine.

For a scheduling model, we denote a feasible schedule by $S$. Schedule $S$ must satisfy all the assumptions and the processing requirements of the given model. For a schedule $S$, the *completion* time of job $j \in N$ is denoted by $C_j(S)$.

### 1.1.3 Optimality Criteria

Normally, the purpose of scheduling is to find a feasible schedule that guarantees the best quality with respect to an accepted criterion. Such a criterion is usually represented as a function that depends on the completion times of the jobs, and to solve the problem, it is required to minimize that *objective* function $\Phi(S)$. In most classical scheduling models, the objective function is *regular,* i.e., is a non-decreasing function of the completion times. For a regular function, it is impossible to achieve a smaller value by delaying the completion of a job. For an objective function $\Phi$, a schedule $S^*$ is called *optimal* if the inequality $\Phi(S^*) \le \Phi(S)$ holds for any feasible schedule $S$.

In the most general setting, a job $j \in N$ can be associated with a function $f_j(S) = f_j(C_j(S))$ which defines the cost of completing job $j$ at time $C_j(S)$. Popular forms of function $f_j$, apart from $f_j(S) = C_j(S)$, are the following:

- *Lateness* $L_j(S) = C_j(S) - d_j$, which is the deviation of the completion time of a job from a due date; this quantity can be negative, zero of positive;
- *Tardiness* $T_j(S) = \max\{C_j(S) - d_j, 0\}$, which shows by how much time a job is completed after its due date; this quantity is always non-negative;
- *Unit penalty* $U_j(S) = 1$ if $C_j(S) > d_j$, and $U_j(S) = 0$, otherwise.

If there is no ambiguity regarding the schedule under consideration, we can write $f_j, C_j, L_j, T_j, U_j$, and $f_j$.

**Table 1.1** Standard scheduling objective functions

| Minmax/Minsum | Cost | Notation | Name |
|---|---|---|---|
| $\Phi = \max\{f_j \mid j \in N\}$ | $f_j$ | $f_{\max}$ | Maximum cost |
| | $f_j = C_j$ | $C_{\max}$ | Makespan |
| | $f_j = L_j$ | $L_{\max}$ | Maximum lateness |
| $\Phi = \sum_{j \in N} f_j$ | $f_j$ | $\sum f_j$ | Total cost |
| | $f_j = C_j$ | $\sum C_j$ | Total completion time |
| | $f_j = C_j$ | $\sum w_j C_j$ | Total weighted completion time |
| | $f_j = T_j$ | $\sum T_j$ | Total tardiness |
| | $f_j = w_j T_j$ | $\sum w_j T_j$ | Total weighted tardiness |
| | $f_j = U_j$ | $\sum U_j$ | The number of late jobs |
| | $f_j = w_j U_j$ | $\sum w_j U_j$ | The weighted number of late jobs |

Table 1.1 lists typical regular objective functions that are studied in classical machine scheduling. The functions are classified as minmax, i.e., to minimize the maximum cost, or minsum, i.e., to minimize the total cost (see the first column). The third column presents notation for the objective functions that is used in the three-field classification scheme described below. Column 4 shows commonly accepted names for these functions. Notice that alternative names are also possible, e.g., in this book along with the name "total weighted completion time," we often write "total weighted flow time" and "the sum of the weighted completion times."

We now present meaningful interpretations of some of the listed functions. For simplicity, assume that the underlying scheduling model is a single machine model in which the jobs are available at time zero, i.e., they do not have individual release dates.

Suppose that to complete project $j$ at time $C_j$, the budget of $f_j(C_j)$ must be available (measured in appropriate monetary units). The maximum cost $f_{\max}$ gives us an optimal upper bound on the budget of an individual project, i.e., each project should be given no more than $f_{\max}$ monetary units, and this value is as small as possible. Similarly, total cost $\sum f_j$ gives the smallest cost for conducting the whole range of projects.

The makespan $C_{\max}$ is the time by which all jobs are completed. This is a very popular quality measure, which is studied for various scheduling models in this book.

The maximum lateness $L_{\max}$ can be interpreted as follows: If all due dates are simultaneously changed from $d_j$ to $d'_j = d_j + L_{\max}$, then there exists a schedule with no late jobs with respect to the modified due dates $d'_j$.

The total completion time $\sum C_j$ is another function actively studied in this book. The completion time $C_j$ can be understood as the time that a job spends in the system. If the value $\sum C_j$ is divided by the total number of jobs, the result corresponds to the average flow time, which shows how long on average a job stays in the system. This is an important characteristic of a service system, also used extensively in queueing theory.

If a manufacturer pays a penalty $w_j$ for each order $j$ that is delivered later than an established due date, then the weighted number of late jobs $\sum w_j U_j$ gives the total penalty to be paid. If $w_j$ is the penalty for every day that order $j$ is late, then the total weighted tardiness $\sum w_j T_j$ corresponds to the overall penalty.

Other performance measures are also possible. For example, in the main body of the book, we look at the functions $\sum C_j^z$ and a linear combination of the latter function and the makespan $C_{\max}$.

## 1.2 Three-Field Classification Scheme

Looking at the developments in the classical scheduling theory from a historical prospective, the main types of scheduling models were identified in the period of 1950–1970, with the arrival of the last classical machine environment, the open shop, in 1976 (see Gonzalez and Sahni (1976)). Still, by the middle of the 1970s, the area of scheduling lacked a unified terminology and generally accepted notation. See a review on the history of scheduling by Potts and Strusevich (2009) for a wider discussion of these issues.

The face of scheduling was dramatically changed by the survey by Graham et al. (1979), which, arguably, is the most influential paper in the area. Among many other achievements of that seminal paper is a *three-field classification scheme*, which associates a scheduling problem to a three-field descriptor $\alpha|\beta|\gamma$ that captures the most important features of the problem: $\alpha$ represents the machine environment, $\beta$ defines the job characteristics, and $\gamma$ is the optimality criterion.

Below, we explain the use of the three-field classification scheme. Let $\circ$ denote the empty symbol.

The first field takes the form $\alpha = \alpha_1\alpha_2$, where $\alpha_1$ and $\alpha_2$ are interpreted as follows:

- $\alpha_1 \in \{\circ, P, Q, R, F, O, J\}$:
  - $\alpha_1 = \circ$: a single machine;
  - $\alpha_1 = P$: identical parallel machines;
  - $\alpha_1 = Q$: uniform parallel machines;
  - $\alpha_1 = R$: unrelated parallel machines;
  - $\alpha_1 = O$: an open shop;
  - $\alpha_1 = F$: a flow shop;
  - $\alpha_1 = J$: a job shop;
- $\alpha_2 = m$: There is a fixed number $m$ of machines; for a multi machine environment, position $\alpha_2$ can be omitted, and in this case, the number of machines $m$ is seen as variable.

Among characteristics that may appear in the second field $\beta$ are the following:

- $\beta_1 \in \{\circ, r_j\}$:
  - $\beta_1 = \circ$: No release dates are specified;

  - $\beta_1 = r_j$: Jobs have release dates;

- $\beta_2 \in \{\circ, pmtn\}$:
  - $\beta_2 = \circ$: No preemption is allowed;
  - $\beta_2 = pmtn$: Operations of jobs may be preempted;
- $\beta_3 \in \{\circ, prec\}$:
  - $\beta_3 = \circ$: No precedence constraints are specified;
  - $\beta_3 = prec$: Precedence constraints over the set of jobs are defined.

In particular, if field $\beta$ is empty, then the jobs are available for processing from time zero, they are processed with no preemption, and no precedence constraints are imposed.

The third field defines the optimality criterion, which typically involves the minimization of a function listed in Table 1.1.

To illustrate the use of the three-field descriptors, below we present several examples:

- $1|r_j, prec|\sum w_j C_j$ is the problem of scheduling jobs with release dates and precedence constraints on a single machine to minimize the total weighted completion time;
- $R|pmtn|L_{\max}$ is the problem of preemptively scheduling jobs on an arbitrary number of unrelated parallel machines to minimize the maximum lateness;
- $Q3||\sum w_j T_j$ is the problem of scheduling jobs on three uniform machines to minimize the total weighted tardiness.

Notice that the three-field classification scheme is an open structure, so that a certain feature of the problem under consideration can be captured by adding the required information to the relevant field using an appropriate encoding. Examples of such extended problem descriptors are widely used in this book.

## 1.3 Computational Complexity

In this section, we discuss, in a rather informal manner, the issues of computational complexity of problems of combinatorial optimization. The reader is referred to the classical texts by Garey and Johnson (1979) and Papadimitriou (1994) for a comprehensive exposition of computational complexity and related topics.

### *1.3.1 Time Complexity of Algorithms*

As a mathematical discipline, scheduling is a part of combinatorial optimization. A typical feature of a combinatorial optimization problem is that the set of feasible solutions (permutations of a finite set, 0–1 vectors or matrices, subgraphs of a graph, etc.) is finite. Given a problem of combinatorial optimization, it is often quite easy to decide whether the problem admits a feasible solution, and then, an optimal solution can be found (at least, in theory) by checking all feasible solutions and taking the best. However, for scheduling problems of modest dimensions (e.g., with dozens of jobs), full enumeration of all feasible solutions is impossible within reasonable time, even if performed on the fastest computers. Thus, not every method of solving a problem should be seen as suitable, and the focus of research in combinatorial optimization, including scheduling, has shifted toward a search for methods that would deliver a solution fast. With the arrival of the seminal paper by Edmonds (1965) on the matching problem, it became clear what should be called a fast, or, put simply, a good solution algorithm. Edmonds (1965) argues that a "good" algorithm is that whose running time depends *polynomially* on the length of the input (or size) of the problem. A typical *algorithm* is a sequence of instructions such that each instruction is followed by exactly one other instruction. Without going into technicalities, the *running time* (also known as *time complexity*) of an algorithm is determined as the number of elementary arithmetic and logical operations, such as addition, multiplication, checking equality, and passing control to another instruction to be performed in order to achieve the required output.

Since a computer uses the binary representation of the numbers, the length $L$ of the input of an instance of a problem is essentially bounded by the number of input parameters times the length of the binary code of the longest of them. Recall that the number of bits needed to encode an integer $a$ is equal to $\log_2 a$. For example, the length of the input for problem $R||\sum w_j C_j$ of minimizing the weighted sum of the completion times on $m$ unrelated machines is bounded by $L = nm \log\left(\max p_{ij}\right) + n \log\left(\max w_j\right)$. An algorithm that requires $O(L^k)$ time, where $k$ is a constant that does not depend on $L$, is called *polynomial-time* (or simply *polynomial*) algorithm. Recall that for functions $f(x)$ and $g(x)$ with positive values, $f(x) = O(g(x))$ holds if and only if there exist positive $M$ and $x_0$ values, such that $f(x) \leq Mg(x)$, for all $x \geq x_0$. Intuitively, $O(L^k)$ implies that the running time of the corresponding algorithm grows at a similar rate as the polynomial $L^k$.

In scheduling, quite often, an optimal schedule is defined by a permutation of jobs, and such a permutation can be found by sorting the jobs according to certain assigned priorities. See Chap. 2 for a review of the results of this type for the models of classical scheduling. Typically, the running time of such an algorithm is determined by sorting the priorities of $n$ jobs, which can be done in $O(n \log n)$ time.

For most of guided enumeration algorithms, e.g., those based on the branch-and-bound or dynamic programming techniques, it is possible to exhibit an instance of the problem so that the behavior of the method is not much better than full enumeration, and the running time grows at least exponentially with the growth of the size of the problem.

However, some of dynamic programming algorithms exhibit interesting behavior. Consider, for example, the algorithm presented by Lawler and Moore (1969) for solving problem $1||\sum w_j U_j$ of minimizing the weighted number of late jobs on a single machine. This is a dynamic programming algorithm that requires $O(nT)$ time, where $T = \min\left\{\max d_j, \sum p_j\right\}$. If we assume the binary representation of the input parameters, the algorithm requires time that is exponential with respect to the size $L$ of the problem, where $L = n \log\bigl(\max\bigl\{\max p_j, \max w_j, \max d_j\bigr\}\bigr)$. However, assuming the unary representation under which an integer $k$ is encoded as $k$ bits (so that, e.g., 5 becomes encoded as 11111), the running time of $O(nT)$ should be qualified as polynomial. The algorithms that require polynomial time under the unary encoding are called *pseudopolynomial*; they are of some interest too, but less practical than polynomial algorithms, and their behavior strongly depends on the size of input parameters.

Until 1979, it has remained an open question whether solving linear programming problems can be achieved in polynomial time. Since linear programming is the main modeling tool in operations research, finding an answer to this question had been of great importance. The first polynomial-time algorithm for linear programming is due to Khachiyan (1979); several alternative methods have become known since. The fact that linear programming problems are polynomially solvable has given rise to a number of algorithms for finding exact and approximate solutions to scheduling problems. Notice that although the well-known simplex method is quite computationally efficient in practice, it remains an open question whether it can be implemented to run in polynomial time.

### *1.3.2 Hard and Easy Problems*

Although the goal of searching for polynomial-time algorithms was established, researchers in combinatorial optimization were not able to find such algorithms for most problems of practical and theoretical interest. In the beginning of the 1970s, the general feeling of the community, that most of these problems had some "built-in" difficulty, found a solid justification.

The ground breaking paper by Cook (1971) delivered the message: Some problems are "easy," i.e., polynomially solvable and some are indeed "hard" and for the latter problems, the existence of polynomial-time algorithms is unlikely. It is beyond the scope of this book to discuss the main statement of Cook (1971), which was formulated and proved in terms of language recognition on deterministic and non-deterministic Turing machines. For the operations research community, Cook's discovery was interpreted by Karp (1972), where the first entries into the list of computationally hard problems appeared, including a scheduling problem, namely $1||\sum w_j U_j$.

Without going into a formal discussion, the findings of the theory of computational complexity and their implications for scheduling can be stated as below. Our main purpose is to provide enough information to the readers to enable them to follow the computational complexity proofs presented in this book.

Most scheduling problems are *optimization* problems, in which we need to search for a schedule $S^*$ that minimizes a certain objective function $\Phi(S)$. An optimization problem can be associated with a *decision* problem, where it is required to give a yes-or-no answer to the question of whether there exists a schedule $S_0$ such that $\Phi(S_0) \leq y$ holds for a given threshold value $y$. Clearly, if the decision version of the problem can be solved in polynomial time, then its optimization counterpart can be solved by solving polynomially many decision problems, with the threshold values generated, e.g., by binary search.

As mentioned in Sect. 1.3.1, standard or *deterministic* algorithms perform computational instructions one by one, and each instruction is followed by exactly one. *Non-deterministic* algorithms, on the other hand, for each instruction make a "good guess" of the possible instructions to perform next. Decision problems that can be solved (i.e., receive a yes-or-no answer) in polynomial time by deterministic algorithms form the class $\mathcal{P}$; decision problems that can be solved in polynomial time by non-deterministic algorithms form the class $\mathcal{NP}$. It is not known whether these two classes coincide, and the answer to this question is seen as one of the most important open issues of modern mathematics. In fact, the "$\mathcal{P}$ versus $\mathcal{NP}$" problem is one of the seven problems put forward by the Clay Mathematics Institute in 2000. A solution to each of these problems will be awarded a so-called Millennium Prize of one million American dollars. See Cook (2006) for the presentation of the "$\mathcal{P}$ versus $\mathcal{NP}$" problem among other problems nominated for the Millennium Prize.

Although the question regarding classes $\mathcal{P}$ and $\mathcal{NP}$ is not fully solved, a widely accepted conjecture assumes that $\mathcal{P} \neq \mathcal{NP}$. Moreover, numerous problems have been identified that are no easier than any other problem in $\mathcal{NP}$. These problems are called *NP-hard.* If one of the NP-hard problems admits a polynomial-time algorithm then all of them are polynomially solvable (and $\mathcal{P} = \mathcal{NP}$), while if we can prove that for one of them there is no polynomial-time algorithm, then all of them cannot be solved in polynomial time (and $\mathcal{P} \neq \mathcal{NP}$). Informally, if a problem is proved to be NP-hard, this means that it is no easier than any other problem that is perceived as hard, and it is assumed that the existence of a polynomial-time algorithm for its solution is unlikely.

To prove that a (decision) problem $P$ is NP-hard, we need to take an arbitrary instance of a certain decision problem $Q$, known to be NP-hard, and transform it in polynomial time (with respect to the length of input of problem $Q$) to a specific instance of problem $P$ such that the established instance of $P$ has a solution (i.e., the "yes" answer) if and only if problem $Q$ has a solution. Such a transformation is called a *polynomial reduction* of problem $Q$ to problem $P$. Informally, to solve the constructed instance of problem $P$, one has to be able to solve an NP-hard problem $Q$, and therefore, problem $P$ is no easier than problem $Q$, which is known to be hard.

Cook (1971) proves that a certain problem of mathematical logic (3- Satisfiability) is NP-hard by definition, i.e., no easier than any problem in $\mathcal{NP}$. Using that fact, Karp (1972) proves the NP-hardness of about twenty problems related to operations research.

It was observed that some problems, while NP-hard with respect to the standard binary encoding, are pseudopolynomially solvable. These problems have received the name of *NP-hard in the ordinary sense* or *binary NP-hard.* Those problems which remain NP-hard under the unary encoding have been named *NP-hard in the strong sense* or *unary NP-hard.* To prove that a (decision) problem $P$ is NP-hard in the strong sense, we need to take a problem $Q$ known to be NP-hard in the strong sense and to show that $Q$ reduces to $P$ in pseudopolynomial time.

An NP-hard problem (either binary or unary) that itself belongs to class $\mathcal{NP}$ is called *NP-complete* (either binary or unary, respectively).

Below, we present a list of decision problems that are used in this book in the NP-hardness proofs.

PARTITION: Given positive integers $e_1, \dots, e_r$ and the index set $R = \{1, \dots, r\}$ such that $e(R) = \sum_{i \in R} e_i = 2R$, is it possible to partition set $R$ into disjoint subsets $R_1$ and $R_2$ such that $e(R_1) = \sum_{i \in R_1} e_i = E$ and $e(R_2) = \sum_{i \in R_2} e_i = E$?

EVEN-ODD PARTITION: Given positive integers $e_1, \dots, e_{2r}$ and the index set $R = \{1, \dots, 2r\}$ such that $e_i \le e_{i+1}$ for $1 \le i < 2r$ and $e(R) = \sum_{i \in R} e_i = 2R$, is it possible to partition set $R$ into disjoint subsets $R_1$ and $R_2$ such that $e(R_1) = \sum_{i \in R_1} e_i = E$ and $e(R_2) = \sum_{i \in R_2} e_i = E$ and for each $i$, $1 \le i \le r$, each set $R_1$ and $R_2$ contains exactly one element of the pair $\{2i - 1, 2i\}$?

3- PARTITION: Given positive integers $e_1, e_2, ..., e_{3r}$ and the index set $R = \{1, \dots, 3r\}$ such that $\frac{1}{4}E < e_i < \frac{1}{2}E$, $i \in R$, and $e(R) = \sum_{i \in R} e_i = rR$, does there exist a partition of set $R$ into $r$ disjoint subsets $R_k$ such that $e(R_k) = \sum_{i \in R_k} e_i = E$ for each $k$, $1 \le k \le r$?

SUBSET PRODUCT: Given the index set $R = \{1, \dots, r\}$, positive integers $e_1, \dots, e_r$, and an integer $V$, does there exist a subset $R' \subseteq R$ such that $\prod_{j \in R'} e_j = V$?

PRODUCT PARTITION: Given the index set $R = \{1, \dots, r\}$ and positive integers $e_1, \dots, e_r$ such that $\prod_{i \in R} e_i = E^2$, is it possible to partition set $R$ into disjoint subsets $R_1$ and $R_2$ such that $\prod_{i \in R_1} e_i = \prod_{i \in R_2} e_i = E$?

The first two problems are NP-hard in the ordinary sense, while 3-PARTITION is NP-hard in the strong sense.

SUBSET PRODUCT is claimed in Garey and Johnson (1979) to be NP-hard in the strong sense; however, the problem is in fact NP-hard in the ordinary sense; the corresponding corrections are done in Johnson (1981) and Chen (1996). The fact that PRODUCT PARTITION is NP-hard in the strong sense is established in Ng et al. (2010).

## *1.3.3 Implications to Scheduling*

Starting from the works by Cook (1971) and Karp (1972), scheduling theory, with its variety of models, has provided researchers with a wide range of problems that have been studied with the purpose of establishing a clear borderline between the easy problems and the hard ones. These complexity studies have stimulated the appearance of the three-field classification scheme described in Sect. 1.2.

Among scheduling problems proved to be NP-hard by Karp (1972) is problem $P2| |C_{\max}$ of minimizing the makespan on two identical parallel machines; its decision version is essentially equivalent to PARTITION. The first paper that systematically studies the complexity issues of scheduling problems and gives their classification is due to Lenstra et al. (1977). It demonstrates the issues of polynomial reducibility between various scheduling models and gives numerous proofs of NP-hardness based on standard NP-hard problems (PARTITION, 3-PARTITION, CLIQUE, HAMILTONIAN CYCLE, etc.). The paper introduces an early version of the classification scheme for scheduling problems, which takes its final form in Graham et al. (1979).

Given a scheduling problem of unknown complexity, it is useful to find its place among similar problems, for which the complexity status has been found. We can track how its complexity is affected by various assumptions and additional restrictions. For illustration, take problem $1||\sum w_jC_j$, a single machine problem to minimize the sum of the weighted completion times, which can be solved $O(n \log n)$ time due to Smith (1956) (see also Sect. 2.2). If each job $j \in N$ has an individual release date $r_j$, then the resulting problem is NP-hard in the strong sense, even if all weights $w_j$ are equal (see Lenstra et al. (1977)). If preemption is allowed, problem $1|r_j, pmtn|\sum w_jC_j$ with equal weights becomes solvable in $O(n^2)$ time due to an algorithm by Schrage (1968). Notice that the latter algorithm only operates if the job weights are equal, while problem $1|r_j, pmtn|\sum w_jC_j$ remains NP-hard in the strong sense. Changing the machine environment by increasing the number of machines clearly does not simplify the problem. Let us start again with a polynomially solvable problem $1||\sum w_jC_j$. Considering its enhancement with several parallel machines, Bruno et al. (1974) show that problem $P2||\sum w_jC_j$ with two identical parallel machines is NP-hard in the ordinary sense, while if the weights are equal, there is a polynomial-time algorithm to minimize $\sum C_j$ on any number of unrelated parallel machines (see also Sect. 4.1.2).

Using similar reasoning, a fairly full description of the complexity of the scheduling problems, including the "minimum hard," "maximum easy," and open problems, can be derived. All relevant information can also be found at the Web site http://www.mathematik.uni-osnabrueck.de/research/OR/class/ initiated by Peter Brucker and maintained by Sigrid Knust, University of Osnabrück, Germany.

The NP-hardness of an optimization problem suggests that it is not always possible to find an optimal solution quickly. Thus, a methodological implication can be derived: If the users want the exact optimum, then they should be prepared that finding such a solution will require considerable time and often cannot be found for problems of larger size. Among methods that may deliver an exact optimum are the branch-and-bound techniques, reductions to mathematical programming problems (integer or mixed integer), and, if applicable, dynamic programming methods.

On the other hand, quite often, for practical purposes, it is sufficient to search for a solution that is fairly close to the optimum, which can be found by faster *heuristic* or *approximation* algorithms. The quality of the performance of such algorithms can be judged based on experimental data, provided the algorithm is applied either to known benchmark instances or to randomly generated instances. Another approach is to apply probabilistic analysis and derive conclusions regarding the expected performance, provided that input parameters are drawn from a certain probability distribution.

A popular academic approach to evaluating the behavior of approximation algorithms is the worst-case analysis, which is based on derived bounds on possible errors produced by an algorithm that holds for all instances of the problem. In this book, we discuss approximation algorithms from this point of view. The corresponding definitions and a brief discussion are contained in the next section.

### *1.3.4 Approximation Algorithms*

Below, we present the required definitions for scheduling problems only. Similar definitions for the algorithms applicable to other problem areas, such as integer programming, can be found in Chap. 4.

Consider a scheduling problem in which the objective is to minimize a cost function $\Phi(S) \geq 0$, where $S$ denotes any feasible schedule for the given problem. As earlier, let $S^*$ denote an optimal schedule so that for any feasible schedule $S$, the inequality $\Phi(S^*) \leq \Phi(S)$ holds. Further, assume that there exists a polynomial-time approximation Algorithm H that generates a feasible schedule $S^H$. Algorithm H is called a *$\rho$-approximation algorithm* if the inequality

$$\Phi(S^H) \leq \rho\Phi(S^*) \tag{1.1}$$

holds for all instances of the problem. We refer to $\rho \geq 1$ as a *ratio guarantee* for Algorithm H. If ratio $\rho$ is the smallest possible, then it is called the *worst-case ratio bound* of Algorithm H. A bound $\rho$ is called *tight* if there exists an instance of the problem for which inequality (1.1) holds as equality.

The class of optimization problems for which there exists an approximation algorithm with a ratio guarantee bounded by a constant is often denoted by $\mathcal{APX}$.

A family of $\rho$-approximation algorithms is called a *polynomial-time approximation scheme* (PTAS) if $\rho = 1 + \varepsilon$ for any $\varepsilon > 0$, and the running time of every Algorithm $H_\varepsilon$ in such a family is polynomial with respect to the length of the problem input. Furthermore, if the running time of every Algorithm $H_\varepsilon$ of an PTAS is bounded by a polynomial in the input size and $1/\varepsilon$, then the family is called a *fully polynomial-time approximation scheme* (FPTAS).

For two problems $P$ and $Q$ of class $\mathcal{APX}$, a notion of the PTAS reducibility can be defined, which informally implies that problems $P$ and $Q$ either admit a PTAS or do not admit a PTAS. Similar to the concept of the NP-hardness, problem $P$ is called APX-hard if there is a PTAS reduction from any problem in $\mathcal{APX}$ to problem $P$. Unless $\mathcal{P} = \mathcal{NP}$, an APX-hard problem does not admit a PTAS.

A good review on approximability issues of scheduling problems is contained in Chen et al. (1998). The existence of PTAS for scheduling problems is discussed in Hoogeveen et al. (2001). Challenging open problems in the area are formulated in Schuurman and Woeginger (1999).

## References

Baker KR, Trietsch D (2009) Principles of sequencing and scheduling. Wiley, Chichester

Blazewicz J, Ecker KH, Pesch E, Schmidt G, Weglarz J (2001) Scheduling computer and manufacturing processes, 2nd edn. Springer, Berlin

Brucker P (2007) Scheduling algorithms, 5th edn. Springer, Guildford

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

Bruno JL, Coffman EG Jr, Sethi R (1974) Scheduling independent tasks to reduce mean finishing time. Commun ACM 17:382–387

Chen Z-L (1997) Erratum: parallel machine scheduling with time dependent processing times. Discr Appl Math 75:103

Chen B, Potts CN, Woeginger GJ (1998) A review of machine scheduling: complexity, algorithms and approximability. In: Du D-Z, Pardalos PM (eds) Handbook of combinatorial optimization, vol 3. Kluwer, Dordrecht, pp 21–169

Cook SA (1971) The complexity of theorem-proving procedures. Proceedings of the 3rd annual ACM symposium on theory of computing. ACM, New York, pp 151–158

Cook SA (2006) The P vs NP problem. In: Carlson J, Jaffe A, Wiles A (eds) The Millennium prize problems. Clay Mathematics Institute, Cambridge (MA) and American Mathematical Society, Providence, pp 87–104

Conway RW, Maxwell WL, Miller LW (1967) Theory of scheduling. Addison Wesley, Reading

Edmonds J (1965) Paths, trees, and flowers. Canad J Math 17:449–467

Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. Freeman, San Francisco

Gonzalez T, Sahni S (1976) Open shop scheduling to minimize finish time. J Assoc Comput Mach 23:665–679

Graham RL, Lawler EL, Lenstra JK, Rinnooy Kan AHG (1979) Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann Discr Math 5:287–326

Hoogeveen H, Schuurman P, Woeginger GJ (2001) Non-approximability results for scheduling problems with minsum criteria. INFORMS J Comput 13:157–168

Johnson DS (1981) The NP-completeness column: an ongoing guide. J Algorithms 2:393–405

Karp RM (1972) Reducibility among combinatorial problems. In: Miller RE, Thatcher JM (eds) Complexity of computer computations. Plenum Press, New York, pp 85–103

Khachiyan LG (1979) A polynomial algorithm in linear programming. Sov Math Dokl 20:191–194

Lawler EL, Moore JM (1969) A functional equation and its application to resource allocation and sequencing problems. Manag Sci 16:77–84

Lenstra JK, Rinnooy Kan AHG, Brucker P (1977) Complexity of machine scheduling problems. Ann Discr Math 1:343–362

Leung JY-T (2004) Handbook of scheduling: algorithms, models and performance analysis. Chapman & Hall/CRC, Boca Raton

Ng CT, Barketau MS, Cheng TCE, Kovalyov MY (2010) "Product Partition" and related problems of scheduling and systems reliability: computational complexity and approximation. Eur J Oper Res 207:601–604

Papadimitriou CH (1994) Computational complexity. Addison-Wesley, Reading

Pinedo M (2009) Planning and scheduling in manufacturing and services, 2nd edn. Springer, Berlin

Pinedo M (2016) Scheduling: theory, algorithms, and systems, 5th edn. Springer, Berlin

Potts CN, Strusevich VA (2009) Fifty years of scheduling: a survey of milestones. J Oper Res Soc 60:S41–S68

Schrage L (1968) A proof of the shortest remaining processing time processing discipline. Oper Res 16:687–690

Schuurman P, Woeginger GJ (1999) Polynomial time approximation algorithms for machine scheduling: ten open problems. J Sched 2:203–213

Smith WE (1956) Various optimizers for single stage production. Naval Res Logist Q 3:59–66

Tanaev VS, Gordon VS, Shafransky YM (1984) Scheduling theory. Single-stage systems. Nauka, Moscow (in Russian); Kluwer, Dordrecht, 1994 (in English)

Tanaev VS, Sotskov YN, Stusevich VA (1987) Scheduling theory. Multi-stage systems. Nauka, Moscow (in Russian); Kluwer, Dordrecht, 1994 (in English)

# Chapter 2
# Pairwise Interchange Argument and Priority Rules

In this chapter, we consider a group of methods that are applicable to problems of optimizing functions over a set of all permutations. The correctness of the basic methods is typically proved by the pairwise interchange argument, and the basic methods can be further extended to become applicable to solve more general problems.

In this chapter and throughout the book, for job $j = \pi(r)$ that occupies the $r$th position in the sequence $\pi = (\pi(1), \pi(2), \dots, \pi(n))$ of jobs of set $N = \{1, 2, \dots, n\}$, the completion time in a schedule $S$ that processes the jobs on a single machine in accordance with permutation $\pi$ is denoted either by $C_j(S)$ or by $C_{\pi(r)}$. The corresponding value can be written as

$$C_{\pi(r)} = \sum_{u=1}^{r} p_{\pi(u)}, \ 1 \le r \le n. \tag{2.1}$$

If a schedule $S$ is determined by a permutation $\pi$, instead of writing the objective function as $F(S) = \sum C_j(S)$, we may write $F(\pi) = \sum C_{\pi(r)}$; the same is applied to other functions, e.g., to the weighted sum of the completion times $\sum w_j C_j(S)$.

In Sect. 2.1, we give a proof of the classical result by Hardy et al. (1934) on minimizing a linear form over a set of permutations and show its implications for various scheduling problems, including problem $1||\sum C_j$. The proofs provided are based on the so-called pairwise interchange argument, which is an often used tool in this book. We also introduce important concepts of a priority rule and of a 1-priority that are widely used throughout this book. In Sect. 2.2, a priority rule is derived for problem $1||\sum w_j C_j$ of minimizing the weighted sum of the completion times on a single machine. In Sect. 2.3, the obtained results are extended to problems $Pm||\sum C_j$ and $Qm||\sum C_j$ of minimizing total completion time on parallel (identical or uniform) machines.

V.A. Strusevich and K. Rustogi, *Scheduling with Times-Changing Effects and Rate-Modifying Activities*, International Series in Operations Research & Management Science 243, DOI 10.1007/978-3-319-39574-6_2

## 2.1 Minimizing a Linear Form

Given two arrays $\mathbf{a} = (a_1, a_2, \ldots, a_n)$ and $\mathbf{b} = (b_1, b_2, \ldots, b_n)$, and two arbitrary permutations $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ and $\sigma = (\sigma(1), \sigma(2), \ldots, \sigma(n))$, consider an expression

$$L(\pi, \sigma) = \sum_{j=1}^{n} a_{\pi(j)} b_{\sigma(j)} \tag{2.2}$$

that is often called a *linear form*. The value $L(\pi, \sigma)$ is the sum of products of elements, with one element from array $\mathbf{a}$ and the other from array $\mathbf{b}$. Introduce the problem of minimizing a linear form, i.e., the problem of finding permutations $\varphi = (\varphi(1), \varphi(2), \ldots, \varphi(n))$ and $\psi = (\psi(1), \psi(2), \ldots, \psi(n))$ such that

$$L(\varphi, \psi) \le L(\pi, \sigma).$$

The problem admits a natural graph theoretical interpretation. Let $G = (U, V; E)$ be an undirected complete bipartite graph. The set of vertices consists of two parts $U = \{u_1, u_2, \ldots, u_n\}$ and $V = \{v_1, v_2, \ldots, v_n\}$. The set $E$ of its edges consists of $n^2$ edges $(u_i, v_j)$, where vertex $u_i \in U$ is associated with a number $a_i$, while vertex $v_j \in V$ is associated with a number $b_j$. The weight of an edge $(u_i, v_j) \in E$ is defined as the product $a_i b_j$. A collection $M \subset E$ of $n$ edges is called a (perfect) *matching*, if no two edges in $M$ are adjacent to the same vertex. The matching defined by given permutations $\pi$ and $\sigma$ is the set of edges $\{(u_{\pi(j)}, v_{\sigma(j)}) | 1 \le j \le n\}$. The weight of a matching $M$ is defined as the sum of all weights of edges in $M$. The problem of minimizing the linear form $L(\pi, \sigma)$ is equivalent to finding a matching of the smallest total weight. If $(u_i, v_j) \in M$, we say that the values $a_i$ and $b_j$ *match.*

*Example 2.1* Figure 2.1 shows a complete bipartite graph $G = (U, V; E)$ with three vertices in each part. Let the values of $(a_1, a_2, a_3)$ and $(b_1, b_2, b_3)$ that are associated with the vertices of set $U$ and set $V$, respectively, be as given in Table 2.1. Given permutations $\pi_1 = (1, 2, 3)$ and $\sigma_1 = (1, 2, 3)$, the matching $M_1$ consists of the edges $(u_1, v_1)$, $(u_2, v_2)$, and $(u_3, v_3)$, so that the value of the linear form $L(\pi, \sigma)$ can be computed as

$$L(\pi_1, \sigma_1) = 5 \times 2 + 2 \times 1 + 4 \times 3 = 24,$$

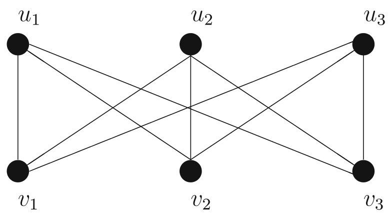


**Fig. 2.1** A complete bipartite graph $G = (U, V; E)$ for Example 2.1

**Table 2.1** The arrays for Example 2.1

| $j$ | 1 | 2 | 3 |
|---|---|---|---|
| $a_j$ | 5 | 2 | 4 |
| $b_j$ | 2 | 1 | 3 |

while for the permutations $\pi_2 = (3, 1, 2)$ and $\sigma_2 = (1, 2, 3)$, the matching $M_2$ consists of the edges $(u_3, v_1)$, $(u_1, v_2)$, and $(u_2, v_3)$, so that the value of the linear form $L(\pi, \sigma)$ can be computed as

$$L(\pi_2, \sigma_2) = 4 \times 2 + 5 \times 1 + 2 \times 3 = 19.$$

The problem of minimizing a linear form can be simplified, so that only one permutation is needed, e.g., $\varphi$, that is defined with respect to a chosen permutation $\psi$. In particular, we may take a permutation $\psi$ such that the sequence $\left(b_{\psi(j)} | 1 \le j \le n\right)$ is non-increasing. It is convenient to assume that the components of array **b** are renumbered in such a way that

$$b_1 \ge b_2 \ge \cdots \ge b_n. \tag{2.3}$$

Under this numbering, a linear form can be written as

$$L(\pi) = \sum_{j=1}^{n} a_{\pi(j)} b_j, \tag{2.4}$$

and to minimize $L(\pi)$, we need to find a permutation $\varphi = (\varphi(1), \varphi(2), \ldots, \varphi(n))$ of the components of array **a**, such that the inequality

$$L(\varphi) = \sum_{j=1}^{n} a_{\varphi(j)} b_j \le L(\pi) = \sum_{j=1}^{n} a_{\pi(j)} b_j \tag{2.5}$$

holds for any permutation $\pi$.

It is clear intuitively that $L(\pi)$ will take smaller values if larger values of $b_j$ are matched to smaller values of $a_j$. The statement below formalizes this observation.

**Theorem 2.1** *Provided that (2.3) holds, permutation $\varphi = (\varphi(1), \varphi(2), \ldots, \varphi(n))$ such that*

$$a_{\varphi(1)} \le a_{\varphi(2)} \le \cdots \le a_{\varphi(n)} \tag{2.6}$$

*satisfies (2.5), i.e., minimizes the linear form $L(\pi)$.*

*Proof* The proof given below utilizes the so-called *pairwise interchange argument*. Assuming that there exists an optimal sequence that does not satisfy a certain rule, a permutation that delivers a smaller value of the objective function can be obtained by interchanging the two adjacent elements that do not obey that rule.

In our case, suppose that $\varphi = (\varphi(1), \varphi(2), \ldots, \varphi(n))$ is a permutation that satisfies (2.5), but does not follow the rule (2.6). Then, there exists an index $k < n$ such that

$$a_{\varphi(1)} \le \cdots \le a_{\varphi(k-1)} \le a_{\varphi(k)};\ a_{\varphi(k)} > a_{\varphi(k+1)}. \tag{2.7}$$

Consider the permutation $\varphi' = (\varphi'(1), \varphi'(2), \ldots, \varphi'(n))$, obtained from $\varphi$ by swapping the elements $\varphi(k)$ and $\varphi(k+1)$, i.e.,

$$\begin{aligned} &\varphi'(j) = \varphi(j),\ 1 \le j \le k-1;\ k+1 \le j \le n; \\ &\varphi'(k) = \varphi(k+1),\ \varphi'(k+1) = \varphi(k). \end{aligned} \tag{2.8}$$

Define $\Delta := L(\varphi) - L(\varphi')$. Due to optimality of $\varphi$, we must have $\Delta \le 0$. However,

$$\begin{aligned} \Delta = &\left(\sum_{j=1}^{k-1} a_{\varphi(j)} b_j + a_{\varphi(k)} b_k + a_{\varphi(k+1)} b_{k+1} + \sum_{j=k+1}^{n} a_{\varphi(j)} b_j\right) \\ &-\left(\sum_{j=1}^{k-1} a_{\varphi(j)} b_j + a_{\varphi(k+1)} b_k + a_{\varphi(k)} b_{k+1} + \sum_{j=k+1}^{n} a_{\varphi(j)} b_j\right) \\ &= \left(a_{\varphi(k)} b_k + a_{\varphi(k+1)} b_{k+1}\right) - \left(a_{\varphi(k+1)} b_k + a_{\varphi(k)} b_{k+1}\right) \\ &= b_k\left(a_{\varphi(k)} - a_{\varphi(k+1)}\right) - b_{k+1}\left(a_{\varphi(k)} - a_{\varphi(k+1)}\right) = (b_k - b_{k+1})\left(a_{\varphi(k)} - a_{\varphi(k+1)}\right). \end{aligned}$$

Since $b_k \ge b_{k+1}$ due to (2.3) and $a_{\varphi(k)} > a_{\varphi(k+1)}$ due to (2.7), we deduce that $\Delta > 0$.

Thus, permutation $\varphi$ cannot be a solution that minimizes the linear form $L(\pi)$. Repeating this argument as many times as required, we conclude that for an optimal permutation $\varphi$, the condition (2.6) must hold. □

Thus, we can describe the following algorithm for solving the problem of minimizing a linear form. The permutation found by the algorithm matches larger components of array **b** to smaller components of array **a**.

**Algorithm Match**

INPUT: Two (unsorted) arrays $\mathbf{a} = (a_1, a_2, \ldots, a_n)$ and $\mathbf{b} = (b_1, b_2, \ldots, b_n)$
OUTPUT: A permutation $\varphi = (\varphi(1), \varphi(2), \ldots, \varphi(n))$ that satisfies (2.5)

**Step 1**. If required, renumber the components of array **b** so that (2.3) holds.
**Step 2**. Output a permutation $\varphi$ such that (2.6) holds.

Algorithm Match reduces to two sorting procedures and therefore requires $O(n \log n)$ time. Simple as it is, the algorithm still plays an important role in optimization over permutations, including numerous scheduling applications discussed in this book.

Applying Algorithm Match to the data in Example 2.1, we first renumber the items so that (2.3) holds, i.e., item 3 with the largest $b$-value becomes item 1, item 1 becomes item 2, and item 2 becomes item 3. With respect to this new numbering,

$$b_1 = 3 > b_2 = 2 > b_3 = 1,$$

and Algorithm Match outputs permutation $\varphi = (3, 1, 2)$, so that

$$a_{\varphi(1)} = a_3 = 2 < a_{\varphi(2)} = a_1 = 4 < a_{\varphi(3)} = a_2 = 5,$$

and the minimum value of the linear form (the smallest weight of a matching in graph shown in Fig. 2.1) is equal to

$$2 \times 3 + 4 \times 2 + 5 \times 1 = 19,$$

i.e., smaller values in one array are matched to larger values in the other array.

### 2.1.1 Minimizing Total Completion Time on a Single Machine

Consider a single machine problem, in which the jobs of set $N = \{1, 2, \dots, n\}$ have to be processed on a single machine. The processing time of job $j \in N$ is equal to $p_j$. It is required to minimize total completion time, i.e., the sum of the completion times. According to the three-field scheduling notation described in Sect. 1.2, the problem can be denoted by $1||\sum C_j$. Clearly, an optimal schedule for the problem is defined by a permutation of jobs.

Suppose that the jobs are sequenced in accordance with some permutation $\pi = (\pi(1), \pi(2), \dots, \pi(n))$. In accordance with (2.1), the total completion time can be written as

$$\begin{aligned} F(\pi) = \sum_{j=1}^{n} C_{\pi(j)} = \sum_{j=1}^{n}\sum_{i=1}^{j} p_{\pi(i)} &= p_{\pi(1)} + \left(p_{\pi(1)} + p_{\pi(2)}\right) + \left(p_{\pi(1)} + p_{\pi(2)} + p_{\pi(3)}\right) \\ &+ \cdots + \left(p_{\pi(1)} + p_{\pi(2)} + \cdots + p_{\pi(n-1)}\right) + \left(p_{\pi(1)} + p_{\pi(2)} + \cdots + p_{\pi(n-1)} + p_{\pi(n)}\right). \end{aligned}$$

It can be seen that the contribution of job $\pi(1)$ to the objective function is $np_{\pi(1)}$, that of job $\pi(2)$ is $(n-1)p_{\pi(1)}$, etc., so that the $j$th job in the sequence contributes $(n-j+1)p_{\pi(j)}$, $1 \le j \le n$. Thus, we can rewrite

$$F(\pi) = \sum_{j=1}^{n} (n - j + 1) p_{\pi(j)}. \tag{2.9}$$

Clearly, $F(\pi)$ is a linear form similar to (2.4) with

$$a_j = p_j,\ b_j = (n - j + 1),\ 1 \le j \le n,$$

so that the $b$-values form the decreasing sequence $(n, n-1, \ldots, 1)$. As follows from Theorem 2.1, a permutation $\varphi$ that minimizes $F(\pi)$ can be found by sequencing the jobs in non-decreasing order of their processing times.

This means that function $F(\pi)$ of the form (2.9) can be minimized by applying a priority rule. This is very important concept that is widely used in this book.

**Definition 2.1** For a scheduling problem of sequencing the jobs of set $N = \{1, 2, \ldots, n\}$, an objective function $\Phi(\pi)$ to be minimized over a set of permutations is said to admit a solution by a *priority rule* if job $j$ can be associated with a priority (or, more precisely, a 1*-priority*) $\omega(j)$, and an optimal permutation can be found by sorting the jobs in accordance with these 1-priorities.

Unless confusion arises, we will normally refer to the values of $\omega(j)$ as 1-priorities and not priorities. The term "1-priority" stresses that the values $\omega(j)$ are associated with individual jobs. We will use the term "priority" for problems with precedence constraints (see, e.g., Chap. 3), where priorities will be associated with subsequences of jobs, rather than with individual jobs.

Two priority rules are often used in scheduling and are of special importance in this book.

**Definition 2.2** The jobs of set $N = \{1, 2, \ldots, n\}$ are said to follow the *SPT* rule (*shortest processing time*) if the jobs are renumbered so that

$$p_1 \le p_2 \le \cdots \le p_n, \tag{2.10}$$

and to follow the *LPT* rule (*longest processing time*) if the jobs are renumbered so that

$$p_1 \ge p_2 \ge \cdots \ge p_n. \tag{2.11}$$

Thus, the following has been proved.

**Theorem 2.2** *For problem $1| |\sum C_j$, an optimal permutation can be found in $O(n \log n)$ time, by sequencing the jobs in accordance with the SPT rule.*

As a rule, 1-priorities are defined in such a way that an optimal permutation is found by sorting the jobs in *non-increasing* order of $\omega(j)$, i.e., the higher the 1-priority is, the earlier the corresponding job is scheduled. For problem $1| |\sum C_j$, its 1-priorities are defined either as $\omega(j) := -p_j$ or as $\omega(j) := 1/p_j$, $j \in N$.

Notice that the SPT sequence of jobs solves problem $1| |\sum C_j$, provided that the resulting permutation is formed from front to rear. It is also possible to fill an optimal permutation from rear to front, so that the job with the largest processing time is sequenced in the last position, the one with the second largest processing time takes the position one before last, etc. This can be interpreted as follows: Scan the jobs in accordance with the LPT sequence and assign the next job to the right most available position of the resulting permutation.

The theorem below demonstrates an important role of the SPT rule for minimizing more general functions than the total completion time $\sum C_j$, such as $\sum C_j^z$ where $z$ is a given positive number, and $\xi C_{\max} + \eta \sum C_j^z$, where $\xi$ and $\eta$ are positive coefficients.

**Theorem 2.3** *Let $\pi = (\pi(1), \ldots, \pi(n))$ be a permutation, in which two jobs $u$ and $v$ such that*

$$p_u > p_v \tag{2.12}$$

*occupy two consecutive positions $r$ and $r+1$, i.e., $u = \pi(r)$ and $v = \pi(r+1)$. Let permutation $\pi'$ be obtained from $\pi$ by swapping the jobs $u$ and $v$. For a single machine problem, let $C_{\pi(h)}$ and $C_{\pi'(h)}$ denote the completion time of the job sequenced in the hth position in permutation $\pi$ and $\pi'$, respectively, $1 \le h \le n$. Then, the equalities*

$$C_{\pi(h)} - C_{\pi'(h)} = \begin{cases} 0, & \text{for } 1 \le h \le n,\ h \ne r \\ p_u - p_v & \text{for } h = r; \end{cases} \tag{2.13}$$

*hold.*

*Proof* It is convenient to represent permutation $\pi$ as $\pi = (\pi_1, u, v, \pi_2)$, where $\pi_1$ and $\pi_2$ are subsequences of jobs that precede job $u$ and follow job $v$ in permutation $\pi$, respectively. Then, $\pi' = (\pi_1, v, u, \pi_2)$.

We present the proof assuming that both sequences $\pi_1$ and $\pi_2$ are non-empty; otherwise, the corresponding part of the proof can be skipped.

The completion times of all jobs in sequence $\pi_1$ are not affected by the swap of jobs $u$ and $v$, i.e., the equality (2.13) holds for each $h$, $1 \le h \le r-1$.

Define $X$ as the completion time of the job in the $(r-1)$th position in sequence $\pi$ (or, equivalently, in $\pi'$), i.e., $X = C_{\pi(r-1)} = C_{\pi'(r-1)}$. For $h = r$, we see that $C_{\pi(r)} = C_u(\pi) = X + p_u$ and $C_{\pi'(r)} = C_v(\pi') = X + p_v$, so that due to (2.12), the equality (2.13) holds for $h = r$.

For $h = r+1$, we derive that

$$C_{\pi(r+1)} = C_v(\pi) = X + p_u + p_v;$$
$$C_{\pi'(r+1)} = C_u(\pi') = X + p_v + p_u,$$

so that (2.13) holds for $h = r+1$.

The jobs that follow the position $r+1$ form the same sequence $\pi_2$ in both permutations $\pi$ and $\pi'$. Each of these jobs starts in permutation $\pi'$ exactly at the same time as in permutation $\pi$, and therefore, (2.13) holds for each $h$, $r+2 \le h \le n$.

This proves the theorem. □

For single machine problems to minimize a regular objective function $\Phi(\pi)$ that depends only on the completion times, Theorem 2.3 demonstrates that in a sequence that minimizes $\Phi(\pi)$, the jobs may be arranged in such a way that a job with a larger processing time is not followed by a job with a smaller processing time. Examples of such a function include, but not limited to $\sum C_j^z$ and $\xi C_{\max} + \eta \sum C_j^z$. This immediately implies the following statement.

**Theorem 2.4** *For problem* $1||\Phi$, *where* $\Phi \in \left\{\sum C_j^z, \xi C_{\max} + \eta \sum C_j^z\right\}$, *an optimal permutation can be found by the SPT rule.*

Reformulating Theorem 2.4, we conclude that problem $1||\Phi$, where $\Phi \in \left\{\sum C_j^z, \xi C_{\max} + \eta \sum C_j^z\right\}$, admits the 1-priority $\omega(j) = 1/p_j$.

In Part 2 of this book, we present statements similar to Theorems 2.3 and 2.4 for extended models with changing processing times.

We now formulate the following recipes for proving and disproving that a certain function $\Phi(\pi)$ admits or does not admit 1-priorities.

**Recipe 2.1**. How to prove that a function $\Phi(\pi)$ admits 1-priorities.

To do this, apply the pairwise interchange technique. Take an arbitrary permutation $\pi = (\pi(1), \pi(2), \dots, \pi(n))$, select a position $r$, and introduce a permutation $\pi'$ obtained from $\pi$ by swapping the elements $\pi(r)$ and $\pi(r+1)$. Derive a function $\omega : N \to \mathbb{R}$ such that for the inequality $\Phi(\pi) \le \Phi(\pi')$ to hold, it is sufficient that $\omega(\pi(r)) \ge \omega(\pi(r+1))$.

**Recipe 2.2**. How to disprove that a function $\Phi(\pi)$ does not admit 1-priorities.

If $\Phi(\pi)$ admitted a 1-priority $\omega(j)$, then for any pair of jobs $u$ and $v$ such that $\omega(u) \ge w(v)$, the value of $\Phi$ for any permutation $\pi$ in which $u$ precedes $v$ should be no larger than the value of $\Phi$ for permutation $\pi'$ obtained from $\pi$ by swapping the jobs $u$ and $v$. Thus, we should exhibit an instance of the problem of minimizing $\Phi(\pi)$ such that there exist two jobs $u$ and $v$ for which the following holds:

**(i)** There exists a permutation $(\pi_1, u, \pi_2, v, \pi_3)$ in which job $u$ is scheduled before job $v$ such that $\Phi(\pi_1, u, \pi_2, v, \pi_3) \le \Phi(\pi_1, v, \pi_2, u, \pi_3)$.

**(ii)** There exists another permutation $(\hat{\pi}_1, u, \hat{\pi}_2, v, \hat{\pi}_3)$ in which job $u$ is scheduled before job $v$ such that $\Phi(\hat{\pi}_1, u, \hat{\pi}_2, v, \hat{\pi}_3) > \Phi(\hat{\pi}_1, v, \hat{\pi}_2, u, \hat{\pi}_3)$.

### *2.1.2 Minimizing the Sum of Products*

Given two arrays $\mathbf{a} = (a_1, a_2, \dots, a_n)$ and $\mathbf{b} = (b_1, b_2, \dots, b_n)$, where $a_j \ge 0$ and either $b_j > 1$ or $b_j < 1$ for $1 \le j \le n$, and a permutation $\pi = (\pi(1), \pi(2), \dots, \pi(n))$, introduce the expression

$$K(\pi) = \sum_{j=1}^{n} a_{\pi(j)} \prod_{i=j+1}^{n} b_{\pi(i)} \tag{2.14}$$

and consider the problem of minimizing $K(\pi)$ over the set of all permutations, i.e., we are looking for permutation $\varphi = (\varphi(1), \varphi(2), \dots, \varphi(n))$ such that the inequality

$$K(\varphi) \le K(\pi) \tag{2.15}$$

holds for all permutations $\pi$.

Associate each $j$, $1 \le j \le n$, with the ratio

$$\kappa(j) = \frac{a_j}{b_j - 1}. \tag{2.16}$$

**Theorem 2.5** *Permutation $\varphi = (\varphi(1), \varphi(2), \dots, \varphi(n))$ such that*

$$\kappa_{\varphi(1)} \le \kappa_{\varphi(2)} \le \cdots \le \kappa_{\varphi(n)} \tag{2.17}$$

*satisfies (2.15), i.e., minimizes the expression $K(\pi)$, provided that all $a_j$'s are non negative and all differences $b_j - 1$ are of the same sign.*

*Proof* The pairwise interchange argument is applied. Suppose that $\varphi = (\varphi(1), \varphi(2), \dots, \varphi(n))$ is a permutation that satisfies (2.15), but does not follow the rule (2.17). Then, there exists an index $k < n$ such that

$$\kappa_{\varphi(1)} \le \cdots \le \kappa_{\varphi(k-1)} \le \kappa_{\varphi(k)}; \ \kappa_{\varphi(k)} > \kappa_{\varphi(k+1)}.$$

Consider the permutation $\varphi' = (\varphi'(1), \varphi'(2), \dots, \varphi'(n))$, obtained from $\varphi$ by swapping the elements $\varphi(k)$ and $\varphi(k+1)$, i.e., $\varphi$ and $\varphi'$ satisfy (2.8).

Define $\Delta := K(\varphi) - K(\varphi')$. Due to optimality of $\varphi$, we must have $\Delta \le 0$. However,

$$\begin{aligned}\Delta = &\left(\sum_{j=1}^{k-1} a_{\varphi(j)} \prod_{i=j+1}^{n} b_{\varphi(i)} + a_{\varphi(k)} \prod_{i=k+1}^{n} b_{\varphi(i)}\right.\\ &\left. + a_{\varphi(k+1)} \prod_{i=k+2}^{n} b_{\varphi(i)} + \sum_{j=k+1}^{n} a_{\varphi(j)} \prod_{i=j+1}^{n} b_{\varphi(i)}\right)\\ &-\left(\sum_{j=1}^{k-1} a_{\varphi(j)} \prod_{i=j+1}^{n} b_{\varphi(i)} + a_{\varphi(k+1)} \left(b_{\varphi(k)} \prod_{i=k+2}^{n} b_{\varphi(i)}\right)\right.\\ &\left. + a_{\varphi(k)} \prod_{i=k+2}^{n} b_{\varphi(i)} + \sum_{j=k+1}^{n} a_{\varphi(j)} \prod_{i=j+1}^{n} b_{\varphi(i)}\right)\end{aligned}$$

$$= a_{\varphi(k)}\big(b_{\varphi(k+1)} - 1\big) \prod_{i=k+2}^{n} b_{\varphi(i)} - a_{\varphi(k+1)}\big(b_{\varphi(k)} - 1\big) \prod_{i=k+2}^{n} b_{\varphi(i)}$$

$$= \prod_{i=k+2}^{n} b_{\varphi(i)}\Big(a_{\varphi(k)}\big(b_{\varphi(k+1)} - 1\big) - a_{\varphi(k+1)}\big(b_{\varphi(k)} - 1\big)\Big).$$

By the choice of $k$, we have that

$$\frac{a_{\varphi(k)}}{b_{\varphi(k)} - 1} > \frac{a_{\varphi(k+1)}}{b_{\varphi(k+1)} - 1},$$

Since $\big(b_{\varphi(k)} - 1\big)\big(b_{\varphi(k+1)} - 1\big) > 0$, it follows that $a_{\varphi(k)}\big(b_{\varphi(k+1)} - 1\big) > a_{\varphi(k+1)}\big(b_{\varphi(k)} - 1\big)$, and we deduce that $\Delta > 0$.

Thus, permutation $\varphi$ cannot be a solution that minimizes $K(\pi)$. Repeating this argument as many times as required, we conclude that for an optimal permutation $\varphi$, the condition (2.17) must hold. □

Theorem 2.5 implies that the problem of *minimizing* the sum of products $K(\pi)$ admits a 1-priority $\omega(j) = -\kappa(j) = -a_j/\big(b_j - 1\big)$. If we need to find a permutation that *maximizes* the value of $K(\pi)$, then the corresponding permutation can be found by sorting the values $\kappa(j)$ in non-increasing order.

## 2.2 Minimizing Total Weighted Completion Time

In this section, we consider problem $1||\sum w_j C_j$, which differs from problem $1||\sum C_j$ discussed in Sect. 2.1.1 by the fact that each job $j \in N$ is given a positive weight $w_j$. The weights reflect the relative importance of the jobs, so that the jobs with higher weights should be completed earlier.

Suppose that the jobs are sequenced in accordance with some permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$. The completion time of job $\pi(j)$ satisfies (2.1), so that the objective function for problem $1||\sum w_j C_j$ can be written as

$$Z(\pi) = \sum_{j=1}^{n} w_{\pi(j)} C_{\pi(j)} = \sum_{j=1}^{n} w_{\pi(j)} \sum_{i=1}^{j} p_{\pi(i)}. \qquad (2.18)$$

To solve problem $1||\sum w_j C_j$, we need to find a permutation $\varphi = (\varphi(1), \varphi(2), \ldots, \varphi(n))$ of jobs such that $Z(\varphi) \le Z(\pi)$ holds for all permutations $\pi$. We show that using the pairwise interchange argument, problem $1||\sum w_j C_j$ can be solved by applying the following extension of the SPT priority rule.

**Definition 2.3** The jobs of set $N = \{1, 2, \ldots, n\}$ are said to follow the *WSPT* rule (*weighted shortest processing time*) if the jobs are renumbered so that

$$\frac{p_1}{w_1} \le \frac{p_2}{w_2} \le \cdots \le \frac{p_n}{w_n}. \tag{2.19}$$

The following statement holds.

**Theorem 2.6** *For problem $1||\sum w_j C_j$, an optimal permutation can be found in $O(n \log n)$ time, by sequencing the jobs in accordance with the WSPT rule.*

*Proof* The proof is along the same lines as the proof of Theorem 2.1. Suppose that $\varphi = (\varphi(1), \varphi(2), \ldots, \varphi(n))$ is an optimal permutation of jobs for problem $1||\sum w_j C_j$, but it does not follow the rule (2.19). Then, there exist an index $k < n$ such that

$$\frac{p_{\varphi(1)}}{w_{\varphi(1)}} \le \cdots \le \frac{p_{\varphi(k-1)}}{w_{\varphi(k-1)}} \le \frac{p_{\varphi(k)}}{w_{\varphi(k)}}; \ \frac{p_{\varphi(k)}}{w_{\varphi(k)}} > \frac{p_{\varphi(k+1)}}{w_{\varphi(k+1)}}. \tag{2.20}$$

Consider the permutation $\varphi' = (\varphi'(1), \varphi'(2), \ldots, \varphi'(n))$, obtained from $\varphi$ by swapping the elements $\varphi(k)$ and $\varphi(k+1)$, i.e., defined by (2.8).

For function $Z(\pi)$ of the form (2.18), define $\Delta := Z(\varphi) - Z(\varphi')$. Due to optimality of $\varphi$, we must have $\Delta \le 0$. However,

$$\Delta = \left(\sum_{j=1}^{k} w_{\varphi(j)} \sum_{i=1}^{j} p_{\varphi(i)} + w_{\varphi(k)} \sum_{i=1}^{k} p_{\varphi(i)} + w_{\varphi(k+1)} \sum_{i=1}^{k+1} p_{\varphi(i)} + \sum_{j=k+2}^{n} w_{\varphi(j)} \sum_{i=1}^{j} p_{\varphi(i)}\right)$$
$$\left(-\sum_{j=1}^{k} w_{\varphi(j)} \sum_{i=1}^{j} p_{\varphi(i)} + w_{\varphi(k+1)}\left(\sum_{i=1}^{k-1} p_{\varphi(i)} + p_{\varphi(k+1)}\right)\right.$$
$$\left. + w_{\varphi(k)} \sum_{i=1}^{k+1} p_{\varphi(i)} + \sum_{j=k+2}^{n} w_{\varphi(j)} \sum_{i=1}^{j} p_{\varphi(i)}\right) = w_{\varphi(k+1)} p_{\varphi(k)} - w_{\varphi(k)} p_{\varphi(k+1)} > 0,$$

where the last strict inequality is due to (2.20).

Thus, permutation $\varphi$ cannot deliver an optimal solution to problem $1||\sum w_j C_j$. Repeating this argument as many times as required, we conclude that for an optimal permutation $\varphi$, the condition (2.19) must hold. □

## 2.3 Minimizing Total Completion Time on Parallel Machines

In this section, we consider the problem of minimizing the sum of the completion times on parallel machines. We show how the results of the previous sections of this chapter can be extended to solve the problems in the classical settings, as well as their capacitated version in which a machine cannot process more than a predefined number of jobs.

### 2.3.1 Uniform Machines

We start with the classical problem, traditionally denoted by $Qm||\sum C_j$. Here, each job $j$ of set $N = \{1, 2, \ldots, n\}$ has to be assigned to be processed on one of the $m \geq 2$ parallel uniform machines. Machine $M_i$ has speed $s_i$. Without loss of generality, we may assume that the machines are numbered in non-increasing order of their speeds, i.e.,

$$s_1 \geq s_2 \geq \ldots \geq s_m. \tag{2.21}$$

It is also convenient to assume that the speed of the slowest machine $M_m$ is equal to 1, and the processing time of job $j \in N$ on machine $M_m$ is equal to $p_j$. In general, if job $j$ is assigned to be processed on machine $M_i$, then such processing takes $p_j/s_i$ time units, $1 \leq i \leq m$. A feasible schedule $S$ is determined by

- a partition job $N$ into $m$ subsets $N_1, N_2, \ldots, N_m$, so that the jobs of set $N_i$ and only those are assigned to be processed on machine $M_i$, $1 \leq i \leq m$;
- the sequence of jobs $\pi^{[i]} = \left(\pi^{[i]}(1), \pi^{[i]}(2), \ldots, \pi^{[i]}(h_i)\right)$ on machine $M_i$, where $h_i = |N_i|$ and $1 \leq i \leq m$.

Take a machine $M_i$, $1 \leq i \leq m$, and suppose that in some schedule $S$, the jobs $\pi^{[i]}(1), \ldots, \pi^{[i]}(h_i)$ are processed on $M_i$ in this order. We have that

$$\begin{aligned} C_{\pi^{[i]}(1)} &= p_{\pi^{[i]}(1)}/s_i, \\ C_{\pi^{[i]}(2)} &= p_{\pi^{[i]}(1)}/s_i + p_{\pi^{[i]}(2)}/s_i, \\ &\cdots \\ C_{\pi^{[i]}(h_i)} &= p_{\pi^{[i]}(1)}/s_i + \ldots + p_{\pi^{[i]}(h_i)}/s_i. \end{aligned}$$

This implies that the contribution of the jobs of set $N_i$ toward the objective function is equal to

$$\sum_{j=1}^{h_i} C_{\pi^{[i]}(j)} = \sum_{j=1}^{h_i} \frac{h_i - j + 1}{s_i} p_{\pi^{[i]}(j)},$$

i.e., an individual contribution for each job assigned to machine $M_i$ is equal to its processing time multiplied by a positional factor of the form $k/s_i$, where $k$ is a position of the job from the rear of the processing sequence on machine $M_i$. In order to minimize the total completion time, we need to match the processing times to the $n$ smallest positional factors of the form $k/s_i$, where $1 \leq k \leq n$ and $1 \leq i \leq m$. As follows from Sect. 2.1, in an optimal schedule, the larger values of the processing times should be matched to smaller positional factors. This can be done by scanning the jobs in the LPT order and to match the next job to the smallest available positional factor.

Formally, an algorithm for solving problem $Qm||\sum C_j$ can be stated as follows.

**Algorithm QSum**

INPUT: The processing times $p_1, p_2, \ldots, p_n$ numbered in accordance with the LPT rule (2.11) and the machine speeds $s_1, s_2, \ldots, s_m$ numbered in accordance with (2.21)

OUTPUT: For each machine, $M_i$, $1 \le i \le m$, permutation $\pi^{[i]}$ that defines the processing sequence of job on the machine in an optimal schedule

**Step 1.** For each machine $M_i$, $i = 1, 2, \ldots, m$, define the positional factors $z_i = 1/s_i$ and $m$ empty sequences $\pi^{[i]}$.

**Step 2.** Scanning the jobs in the order of their numbering, for each job $j$ from 1 to $n$, do

**(a)** Find the machine $M_v$, $1 \le v \le m$, associated with the smallest positional factor, i.e.,

$$z_v := \min\{z_i | 1 \le i \le m\}; \tag{2.22}$$

If such a machine is not unique, break ties by setting $v$ to be the largest index $v$ for which (2.22) holds.

**(b)** Assign job $j$ to machine $M_v$ and place it in front of the current permutation $\pi^{[v]}$, i.e., define

$$\pi^{[v]} := (j, \pi^{[v]}),\ z_v := z_v + \frac{1}{s_v}.$$

The running time of the algorithm is $O(n \log n)$, including the renumbering of the jobs.

For illustration, consider the following example.

*Example 2.2* To test a new scheduling algorithm, a researcher wants to run six sets of test data on three computers. Compared to Computer 3, Computer 1 is three times faster, while Computer 2 is two times faster. The computation time (in minutes) needed to run the test sets on Computer 3 is given in Table 2.2. It is required to organize the computational experiment in such a way that the average completion time of a data set is minimized. Interpreting the three computers as three machines $M_1$, $M_2$, and $M_3$, and the data sets as six jobs, we can reduce the original problem to problem

**Table 2.2** Processing times of the tests sets on Computer 3 for Example 2.2

| Set $j$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Processing time $p_j$ | 12 | 36 | 24 | 42 | 18 | 30 |

$Q3||\sum C_j$ and solve it by Algorithm QSum. We may assume that the speed of machine $M_3$ (i.e., that of Computer 3) is taken as 1, so that $s_1 = 3, s_2 = 2$, and $s_3 = 1$. Table 2.3 shows how Algorithm QSum is run. The jobs are scanned in the LPT order. Each row of Table 2.3 shows the parameters after the corresponding job has been assigned to the machine. Notice that when assigning jobs 3 and 5, the current smallest positional factor is not unique, and we break ties giving preference to the machine with the slowest speed ($M_3$ for job 3 and $M_2$ for job 5). The last column shows the actual processing times of the jobs, i.e., the initial values $p_j$ from Table 2.2 divided by the speed of the machine the job is assigned to. Figure 2.2 presents a Gantt chart of an optimal schedule. The completion times of the jobs (test sets) are shown in Table 2.4. The average completion time is equal to $(4 + 27 + 24 + 28 + 9 + 14)/6 = 17.667$ min, i.e., 17 min 40 sec.

Notice that Algorithm QSum can be deduced from a solution procedure for a more general scheduling problem $Rm||\sum C_j$ on unrelated parallel machines (see Sect. 4.1.2).

**Table 2.3** Running Algorithm QSum for Example 2.2

| $j$ | $\pi^{[1]}$ | $\pi^{[2]}$ | $\pi^{[3]}$ | $z_1$ | $z_2$ | $z_3$ | Actual processing times |
|---|---|---|---|---|---|---|---|
| | () | () | () | $\frac{1}{3}$ | $\frac{1}{2}$ | 1 | |
| 4 | (4) | () | () | $\frac{2}{3}$ | $\frac{1}{2}$ | 1 | 14 |
| 2 | (4) | (2) | () | $\frac{2}{3}$ | 1 | 1 | 18 |
| 6 | (6, 4) | (2) | () | 1 | 1 | 1 | 10 |
| 3 | (6, 4) | (2) | (3) | 1 | 1 | 2 | 24 |
| 5 | (6, 4) | (5, 2) | (3) | 1 | $\frac{3}{2}$ | 2 | 9 |
| 1 | (1, 6, 4) | (5, 2) | (3) | $\frac{4}{3}$ | $\frac{3}{2}$ | 2 | 4 |

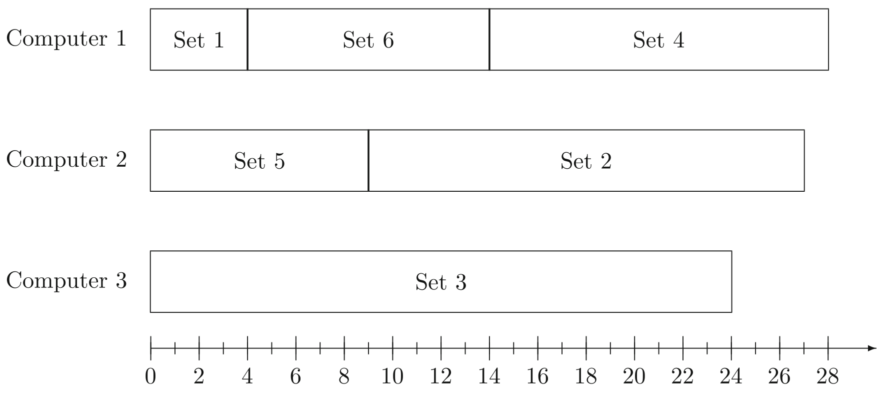


**Fig. 2.2** An optimal schedule for Example 2.2

**Table 2.4** Completion times of the test sets in an optimal schedule for Example 2.2

| Set $j$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Completion time $p_j$ | 4 | 27 | 24 | 28 | 9 | 14 |

Algorithm QSum above can be easily modified to handle a capacitated version of the problem, in which additional restrictions regarding the number of jobs to be assigned to a machine are imposed. Formally, suppose that in any feasible schedule, machine $M_i$ processes no more than $q^{[i]}$ jobs, where $\sum_{i=1}^{m} q^{[i]} \geq n$, so that a feasible schedule exists. We denote the problem under consideration by $Qm\left|\sum_{i=1}^{m} q^{[i]} \geq n\right| \sum C_j$. The algorithm below is a minor modification of Algorithm QSum: If no more jobs can be assigned to a machine, the machine is excluded from consideration by setting the corresponding positional factor to infinity.

**Algorithm QSumCap**

INPUT: The processing times $p_1, p_2, \ldots, p_n$ numbered in accordance with the LPT rule (2.11) and the machine speeds $s_1, s_2, \ldots, s_m$ numbered in accordance with (2.21) and machine capacities $q^{[1]}, q^{[2]}, \ldots, q^{[m]}$

OUTPUT: For each machine, $M_i$, $1 \leq i \leq m$, permutation $\pi^{[i]}$ that defines the processing sequence of job on the machine in an optimal schedule

**Step 1**. For each machine $M_i$, $i = 1, 2, \ldots, m$, define the positional factors $z_i := 1/s_i$ and $m$ empty sequences $\pi^{[i]}$.
**Step 2**. Scanning the jobs in the order of their numbering, for each job $j$ from 1 to $n$, do the following:

**(a)** Find the machine $M_v$, $1 \leq v \leq m$, associated with the smallest positional factor that satisfies (2.22). If such a machine is not unique, break ties by setting $v$ to be the largest index $v$ for which (2.22) holds.
**(b)** Assign job $j$ to machine $M_v$ and place it in front of the current permutation $\pi^{[v]}$, i.e., define $\pi^{[v]} := (j, \pi^{[v]})$. If $z_v = \frac{q^{[v]}}{s_v}$, then exclude this machine from consideration by defining $z_v := +\infty$; otherwise, define $z_v := z_v + \frac{1}{s_v}$.

The running time of Algorithm QSumCap is $O(n \log n)$.

### 2.3.2 Identical Machines

The results of Sect. 2.3.1 can be adapted to solving the problem of minimizing total completion time on identical parallel machines. The speeds of all machines are assumed equal to 1, so that the processing time of job $j \in N$ on any machine is equal to $p_j$. Problem $Pm|\beta|\sum C_j$ is a special case of problem $Qm|\beta|\sum C_j$, where the field $\beta$ either is empty or reads $\sum_{i=1}^{m} q^{[i]} \geq n$. Thus, Algorithms QSum and QSumCap solve the uncapacitated and capacitated versions of the problem on identical machines, respectively.

Given a schedule $S$, let $F(S) = \sum C_j(S)$ denote the total completion time. For problem $Pm| |\sum C_j$ on $m$ parallel machines, a schedule found by Algorithm QSum is denoted by $S^*(m)$. We exclude from consideration the case that $m \geq n$, since it is optimal to assign exactly one job to each of $n$ arbitrarily chosen machines.

Recall that for a real number $x$, the *ceiling* $\lceil x \rceil$ is equal to the smallest integer that is no less than $x$. Assuming that the jobs are numbered in accordance with the LPT rule (2.11), in schedule $S^*(m)$, each of the first $m$ jobs takes the last position on one of the machines, each of the next $m$ jobs takes the second from last position on one of the machines, etc. This implies that job $j$ contributes its processing time to the objective function exactly $\left\lceil \frac{j}{m} \right\rceil$ times, so that

$$F(S^*(m)) = \sum_{j=1}^{n} p_j \left\lceil \frac{j}{m} \right\rceil. \tag{2.23}$$

For problem $Pm| |\sum C_j$, a solution algorithm can be designed based on different principles, compared to Algorithm QSum. Indeed, an optimal solution can be obtained by scanning the jobs in the SPT order (2.10) and building a schedule from front to rear. Such an algorithm is presented below.

**Algorithm PSumSPT**

INPUT: The processing times $p_1, p_2, \ldots, p_n$ numbered in accordance with the SPT rule (2.10)

OUTPUT: A schedule $S_{SPT}$ that is defined by permutations $\pi^{[i]}$, $1 \leq i \leq m$

**Step 1** For each machine $M_i, i = 1, 2, \ldots, m$, define sequence $\pi^{[i]} := (i)$ that consists of one job $i$.

**Step 2** Scanning the jobs in the order of their numbering, for each job $j$ from $i + 1$ to $n$, do the following:

> For the current partial schedule, by checking the machines in the order of their numbering, determine the machine $M_v$, $1 \leq v \leq m$, that completes its jobs earlier than other machines. Assign job $j$ to machine $M_v$, update $\pi^{[v]} := \left(\pi^{[v]}, j\right)$.

Algorithm PSumSPT requires linear time, provided that the SPT sequence of jobs is available.

## 2.4 Bibliographic Notes

Theorem 2.1 is a classical result that traces back to Hardy et al. (1934), where it is formulated as Theorem 368 on p. 262. Several different proofs are provided for the theorem, starting from that based on the pairwise interchange argument. Another proof of the theorem is given in Sect. 4.1.3. More proofs of Theorem 2.1 and its extensions can be found in Chap. 6, Section A of the book by Marshall and Olkin (1979).

A link between the problem of minimizing a linear form and a single-flight low-risk helicopter transportation problem is established in Qian et al. (2012). The latter problem arises in the oil and gas offshore mining, when employees are to be delivered to and picked up from a number of offshore installations (rigs or platforms) by a helicopter, so as to minimize the number of people exposed to landings and takeoffs. If the number of people to be delivered to and picked up from an installation $j$ is equal to $P_j$ and $D_j$, respectively, then an optimal order of visits to the installations can be found by Algorithm Match, so that the installations are visited in non-decreasing order of $P_j - D_j$.

Theorem 2.5 is independently proved by Rau (1971) and Kelly (1982).

Historically, Theorem 2.6 on a priority rule for solving problem $1||\sum w_j C_j$ is one of the first scheduling results. It is proved by Smith (1956), and this is why the WSPT rule stated in Definition 2.3 is often referred to as *Smith's* rule. Queyranne (1993) presents a minimal linear description of the solution polyhedron defined as the convex hull of feasible completion time vectors and deduces Smith's rule using the greedy algorithm known in optimization over polymatroids.

A special case of Theorem 2.4 for problem $1||\sum C_j^2$ is proved in Townsend (1978).

A solution algorithm for problem $Qm||\sum C_j$ is first described in the book by Conway et al. (1967). Our exposition mainly follows the book by Brucker (2007). Algorithm QSumCap for problem $Qm\left|\sum_{i=1}^{m} q^{[i]} \geq n\right|\sum C_j$ is presented in Rustogi and Strusevich (2012).

Algorithm PSumSPT is also first presented in Conway et al. (1967). It should be seen as a version of the famous list scheduling algorithm by Graham (1966) with a list found by the SPT rule.

For an environment with $m$ parallel identical machines with the objective function $\Phi \in \left\{C_{\max}, \sum C_j\right\}$, Rustogi and Strusevich (2013) consider the problem of determining machine impact which shows what can be gained if extra machines are added.

The link between problem $Pm||\sum C_j$ and the low-risk multiflight helicopter transportation problem is established by Qian et al. (2015). That paper presents a number of approximation algorithms for the capacitated version of the problem with pickup only.

While problem $1||\sum w_j C_j$ is polynomially solvable, adding an additional machine changes the complexity dramatically. Indeed, problem $P2||\sum w_j C_j$ is NP-hard even if $p_j = w_j$ for all $j \in N$ (see Bruno et al. (1974)).

## References

Brucker P (2007) Scheduling algorithms, 5th edn. Guildford, Springer
Bruno JL, Coffman EG Jr, Sethi R (1974) Scheduling independent tasks to reduce mean finishing time. Comm ACM 17:382–387
Conway RW, Maxwell WL, Miller LW (1967) Theory of scheduling. Addison Wesley, Reading (MA)
Graham RL (1966) Bounds for certain multiprocessing anomalies. Bell Syst Technol J 45:1563–1581
Hardy G, Littlewood JE, Polya G (1934) Inequalities. Cambridge University Press, London
Kelly FP (1982) A remark on search and sequencing problems. Math Oper Res 7:154–157
Marshall AW, Olkin I (1979) Inequalities: Theory of majorization and its applications. Academic Press, New York
Qian F, Gribkovskaia IV, Halskau Ø (2012) Passenger and pilot risk minimization in offshore helicopter transportation. Omega 40:584–593
Qian F, Strusevich VA, Gribkovskaia IV, Halskau Ø (2015) Minimization of passenger takeoff and landing risk in offshore helicopter transportation: models, approaches and analysis. Omega 51:93–106
Queyranne M (1993) Structure of a simple scheduling polyhedron. Math Progr 58:263–285
Rau JG (1971) Minimizing a function of permutations of *n* integers. Oper Res 19:237–240
Rustogi K, Strusevich VA (2012) Simple matching vs linear assignment in scheduling models with positional effects: a critical review. Eur J Oper Res 222:393–407
Rustogi K, Strusevich VA (2013) Parallel machine scheduling: Impact of adding an extra machine. Oper Res 61:1243–1257
Smith WE (1956) Various optimizers for single state production. Naval Res Logist Quart 3:66–69
Townsend W (1978) The single machine problem with quadratic penalty function of completion times: a barnch-and-bound solution. Manag Sci 24:530–533

# Chapter 3
# Sequencing Under Precedence Constraints

In this chapter, we review scheduling problems on a single machine, provided that precedence constraints are imposed on the set of jobs, so that not all permutations of jobs are feasible.

The precedence constraints are normally given in the form of a *directed graph* (*digraph*). We pay a special attention to the case when the constraints are defined by a series-parallel graph, so that a scheduling problem of interest can be solved in polynomial time, provided that its objective function is priority-generating. For illustration, we often refer to the problem of minimizing the weighted sum of the completion times on a single machine, denoted either by $1|prec|\sum w_jC_j$ (if the precedence constraints are arbitrary) or by $1|SP-prec|\sum w_jC_j$ (if the constraints are series-parallel).

This chapter is structured as follows. Section 3.1 provides all required definitions regarding the reduction graphs. The concept of a priority-generating function is introduced and explored in Sect. 3.2. In particular, it is shown that for the problem of minimizing $\sum w_jC_j$ on a single machine, the objective function is priority-generating, while for the problem of minimizing a linear form, the objective function is not priority-generating. The issues of minimizing a priority-generating function under series-parallel precedence constraints are addressed in Sect. 3.3.

## 3.1 Graphs, Posets, and Other Definitions

Consider the following generic single machine scheduling problem. The jobs of set $N = \{1, 2, \ldots, n\}$ have to be processed with no preemption on a single machine, and the processing of job $j \in N$ takes $p_j$ time units. The jobs are simultaneously available at time zero, and for each job $j$, a weight $w_j$ is given that indicates its relative importance. The machine can handle only one job at a time and is permanently available from time zero.

For some schedule $S$, the completion time of job $j \in N$ is denoted by $C_j(S)$. If a schedule $S$ is defined by a permutation $\pi = (\pi(1), \pi(2), \dots, \pi(n))$ of jobs, we may denote the completion time of job $j$ by $C_j(\pi)$. If no confusion arises, we may write $C_j$, without making a reference to a particular schedule or sequence. The objective is to minimize a function $\Phi(\pi)$, a non-decreasing function of $C_j$, $j \in N$. Mainly, we will focus on three functions: the makespan $C_{\max}(\pi) = \max\{C_j | j \in N\}$, the total completion time $F(\pi) = \sum_{j\in N} C_j$, and the weighted total completion time $Z(\pi) = \sum_{j\in N} w_j C_j$.

It is often found in practice that some products are manufactured in a certain order implied, for example, by technological, marketing, or assembly requirements. Thus, in reality, not all permutations of jobs are permitted, and this can be modeled by imposing *precedence constraints* onto set $N$ of jobs to describe allowable (feasible) sequences of jobs.

### *3.1.1 Reduction Graphs*

Formally, precedence constraints among the jobs are defined by a *binary relation* $\to$. We write $i \to j$ and say that job $i$ *precedes* job $j$ if in any feasible schedule job $i$ must be completed before job $j$ starts; in this case, $i$ is a *predecessor* of $j$, and $j$ is a *successor* of $i$. Binary relation $\to$ is a *strong order* relation, that is both *asymmetric* ($i \to j$ implies that not $j \to i$) and *transitive* ($i \to j$ and $j \to k$ imply $i \to k$). We write $i \sim j$ if jobs are *independent*, i.e., neither $i \to j$ nor $j \to i$. A sequence (or a permutation) of jobs is *feasible* if no pair of jobs violate the precedence constraints. Precedence constraints are usually given by a directed circuit-free graph $G$ in which the set of vertices is identical with the set of jobs and there is a path from vertex $i$ to vertex $j$ if and only if job $i$ precedes job $j$. Recall that a graph is *circuit-free* (or *acyclic*) if it contains no cycles, i.e., contains no oriented closed paths with the same initial and terminal vertices. Moreover, any directed acyclic graph (*dag*) induces a partial order on its vertices: $i \to j$ if and only if there is a path from vertex $i$ to vertex $j$ in $G$.

Given a dag, the *outdegree* and the *indegree* of a vertex are equal to the number of arcs going from and coming to the vertex, respectively; a vertex of both zero outdegree and zero indegree is *isolated.*

In the presence of precedence constraints, the set of jobs is partially ordered. A *partially ordered set* (*poset*) $P = (N, R)$ is defined by a set $N$ of jobs and an order relation $R$. Relation $R$ is a binary relation, given by a set of (ordered) pairs $\langle i, j\rangle$, where $i$ and $j$ are distinct elements of $N$, and $i$ precedes $j$, i.e.,

$$\langle i, j\rangle \in R \text{ if and only if } i \to j.$$

Thus, in our case, the notions of “precedence constraints,” “dag,” and “poset” are interchangeable, and we will use that of these terms which we find the most appropriate in a particular context.

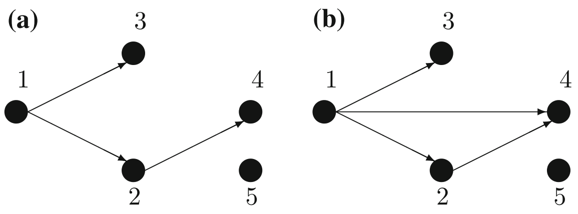


**Fig. 3.1** **a** Dag $G$ on five vertices; **b** the transitive closure of $G$

For a dag, let $(i, j)$ denote an arc that goes from vertex $i$ to vertex $j$. The *transitive closure* of a dag $G$ is a dag $G_T$ such that $G_T$ contains an arc $(i, j)$ if and only if $i \neq j$ and there is a path from $i$ to $j$ in $G$. An arc from vertex $i$ to vertex $j$ in $G$ is *transitive* (or *redundant*) if there is a path from $i$ to $j$ which avoids the arc $(i, j)$. The graph obtained from $G$ by removing all transitive arcs is called the *reduction graph* and is denoted by $G_R$. Notice that $G_R$ is the unique dag which has no transitive arcs and has the same transitive closure as $G$. The partial order induced either by $G_T$ or by $G_R$ is the same as that induced by $G$. Therefore, we may assume that given a dag with set $N$ of vertices and without transitive arcs, we are given a relation $\rightarrow$ of strong order which defines precedence constraints on $N$ and vice versa.

Figure 3.1a shows dag $G$ on five vertices, which is the reduction graph of the order relation

$$R = \{\langle 1, 2\rangle, \langle 1, 3\rangle, \langle 2, 4\rangle\}.$$

Its transitive closure is shown in Fig. 3.1b; in this graph, the path from vertex 1 to vertex 4 initiates the arc $(1, 4)$.

In terms of scheduling, the four jobs 1, 2, 3, and 4 can only be sequenced in accordance with one of the permutations

$$(1, 2, 3, 4), (1, 3, 2, 4), (1, 2, 4, 3).$$

In graph $G$, vertex 5 is isolated. Thus, to produce a feasible sequence, job 5 can be inserted into any position of any of the three sequences listed above.

### 3.1.2 Series-Parallel Graphs

Let $G = (X, U)$ be a dag, where $X$ is the set of vertices and $U$ is the set of arcs. A dag $G = (X, U)$ is said to be the *parallel composition* of two dags $G_1 = (X_1, U_1)$ and $G_2 = (X_2, U_2)$ such that $X_1 \cap X_2 = \varnothing$, if $X = X_1 \cup X_2$ and $U = U_1 \cup U_2$. A dag $G = (X, U)$ is said to be the *series composition* of two dags $G_1 = (X_1, U_1)$ and $G_2 = (X_2, U_2)$ such that $X_1 \cap X_2 = \varnothing$, if $X = X_1 \cup X_2$ and $U = U_1 \cup U_2 \cup \widetilde{U}$, where $\widetilde{U}$ is the set of arcs going from each vertex of graph $G_1$ with zero outdegree to each vertex of graph $G_2$ with zero indegree.

For illustration, look at two dags, $G_1$ and $G_2$ in Fig. 3.2a. If these two graphs are considered as a single graph that consists of two connected components, then this

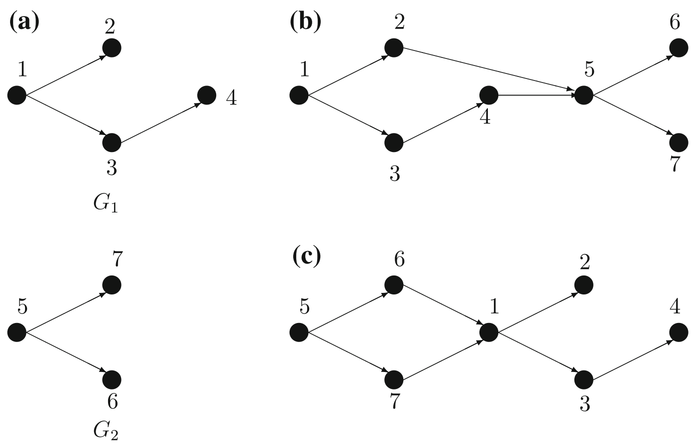
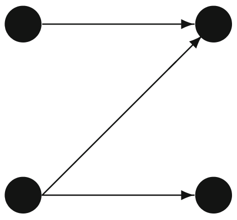


**Fig. 3.2** **a** Two dags $G_1$ and $G_2$, **b** the series composition of $G_1$ and $G_2$, and **c** the series composition of $G_2$ and $G_1$

**Fig. 3.3** $Z$-graph



graph is the parallel composition of $G_1$ and $G_2$. Figure 3.2b shows the graph obtained as the series composition of $G_1$ and $G_2$ (applied in this order). The graph in Fig. 3.2c is the series composition of $G_2$ and $G_1$ (applied in this order).

**Definition 3.1** A directed graph is called *series-parallel* (or an *SP-graph* ) if it either consists of only one vertex, or can be obtained from two series-parallel graphs by subsequent application of the operations of series and/or parallel composition.

There are several properties that identify an SP-graph. One of them is formulated in terms of subgraphs that are forbidden for an SP-graph. Recall that a graph $G' = (X', U')$ is a *subgraph* of another graph $G = (X, U)$ if $X' \subseteq X$ and $U' \subseteq U$, i.e., $G'$ is obtained form $G$ by a removal of some vertices and/or arcs. Further, for any subset $X'$ of vertices in graph $G$, an *induced* subgraph is the maximal subgraph of $G$ with the vertex set $X'$. A four-vertex graph shown in Fig. 3.3 is called a *Z-graph*.

**Theorem 3.1** *A dag $G = (X, U)$ with no transitive arcs is series-parallel if and only if its transitive closure does not contain a Z-graph shown in Fig. 3.3 as an induced subgraph.*

Let us denote the operations of parallel and series compositions by the letters **P** and **S**, respectively. We write $G = G_1\mathbf{P}G_2$ and $G = G_1\mathbf{S}G_2$ if graph $G$ is either

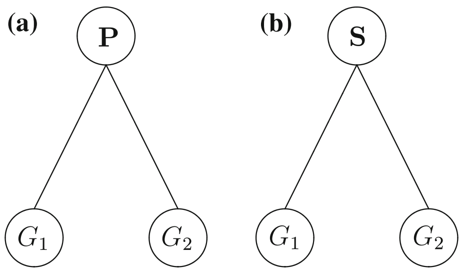


**Fig. 3.4** Tree representations **a** of parallel composition and **b** of series composition

the parallel composition or the series composition, respectively, of graphs $G_1 = (X_1, U_1)$ and $G_2 = (X_2, U_2)$. Parallel and series compositions can be represented by the so-called *decomposition trees* shown in Fig. 3.4 for graphs $G = G_1\mathbf{P}G_2$ and $G = G_1\mathbf{S}G_2$.

In order to represent an SP-graph by a binary decomposition tree, we first associate a one-node tree with each single-vertex graph which is used for constructing the SP-graph and then use the rules shown in Fig. 3.4 to build larger trees from smaller ones by applying operations **P** and **S**. Notice that several non-isomorphic binary decomposition trees may correspond to the same SP-graph since the operation of parallel composition is commutative, i.e., graphs $G_1$ and $G_2$ in Fig. 3.4a can be swapped, and both operations **S** and **P** are associative. See Fig. 3.5 that illustrates

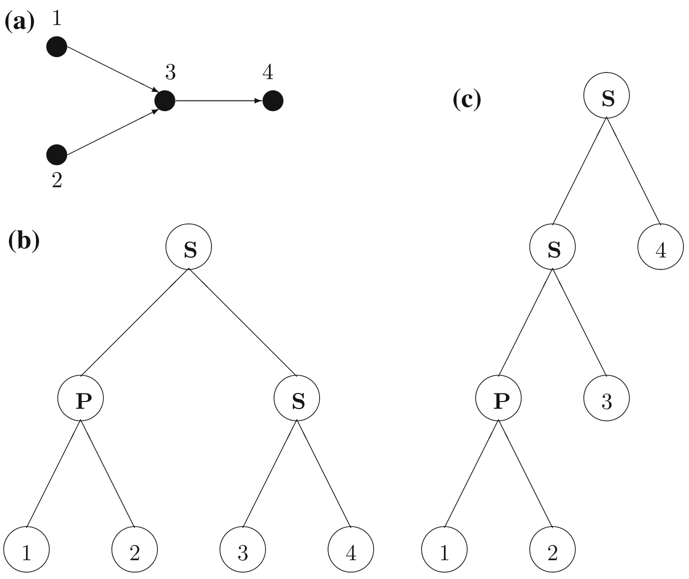


**Fig. 3.5** Representations of the SP-graph **a** by decomposition trees **b** and **c**

how the SP-graph shown in Fig. 3.5a can be represented by two non-isomorphic decomposition trees in Fig. 3.5b and c.

In what follows, we assume that SP-graphs have no transitive arcs and are given by a decomposition tree. Notice that given an SP-graph $G = (X, U)$, its decomposition tree can be found in $O\left(|X|^2\right)$ time.

## 3.2 Priority-Generating Functions

There is a wide class of objective functions depending on the elements of a poset, called *priority-generating* functions, which can be minimized in polynomial time, provided that the precedence constraints are given by a series-parallel graph.

Consider a scheduling problem of processing a set $N = \{1, 2, \ldots, n\}$ of jobs, in which a schedule is defined by a sequence in which the jobs are processed. Let $\Phi(\pi)$, a non-decreasing function of $C_j$, $j \in N$, be the objective function to be minimized.

Recall that for a scheduling problem that can be solved by some priority rule, a 1-priority of a job $j$ is a function $\omega(j)$ such that in an optimal schedule, the jobs are sequenced in non-increasing order of the $\omega(j)$ values (see Sect. 2.1.1). In particular, the LPT rule corresponds to the 1-priority function $\omega(j) = p_j$, while for the SPT rule either $\omega(j) = -p_j$ or $\omega(j) = 1/p_j$.

In order to be able to handle scheduling problems under precedence constraints, we need an extended notion of a priority function that is defined for subsequences of jobs rather than just for individual jobs.

**Definition 3.2** Let $\pi^{\alpha\beta} = (\pi'\alpha\beta\pi'')$ and $\pi^{\beta\alpha} = (\pi'\beta\alpha\pi'')$ be two permutations of $n$ jobs that differ only in the order of the subsequences $\alpha$ and $\beta$ (each of the sequences $\pi'$ and $\pi''$ may be empty). For a function $\Phi(\pi)$ that depends on a permutation, suppose that there exists a function $\omega(\pi)$ such that for any two permutations $\pi^{\alpha\beta}$ and $\pi^{\beta\alpha}$, the inequality $\omega(\alpha) > \omega(\beta)$ implies that $\Phi(\pi^{\alpha\beta}) \leq \Phi(\pi^{\beta\alpha})$, while the equality $\omega(\alpha) = \omega(\beta)$ implies that $\Phi(\pi^{\alpha\beta}) = \Phi(\pi^{\beta\alpha})$. In this case, function $\Phi$ is called a *priority-generating function*, while function $\omega$ is called its *priority function*. For a (partial) permutation $\pi$, the value of $\omega(\pi)$ is called the *priority* of $\pi$.

Intuitively, a priority function allows us to rank not only individual jobs but also partial permutations of jobs. A priority function applied to a single job becomes a 1-priority for that job. Thus, for function $\Phi(\pi)$ to be priority-generating, it is necessary that the problem of minimizing $\Phi(\pi)$ can be solved by an application of a priority rule. On the other hand, functions that admit a 1-priority do not necessarily admit a priority in the sense of Definition 3.2.

We now formulate the following recipes for proving and disproving that a certain function $\Phi(\pi)$ is priority-generating. Let $\pi^{\alpha\beta} = (\pi'\alpha\beta\pi'')$ and $\pi^{\beta\alpha} = (\pi'\beta\alpha\pi'')$ be two permutations of all jobs that only differ in the order of the subsequences $\alpha$ and $\beta$.

**Recipe 3.1** How to prove that a function $\Phi(\pi)$ is priority-generating

Derive a function $\omega(\pi)$ that depends only the elements of a partial permutation $\pi$ such that the sign of the difference $\Phi\left(\pi^{\alpha\beta}\right) - \Phi\left(\pi^{\beta\alpha}\right)$ depends on the sign of the difference $\omega(\alpha) - \omega(\beta)$.

**Recipe 3.2** How to disprove that a function $\Phi(\pi)$ is priority-generating

Exhibit an instance of the problem that $\Phi(\pi^{\alpha\beta}) < \Phi(\pi^{\beta\alpha})$ for some permutations $\pi^{\alpha\beta} = (\pi'\alpha\beta\pi'')$ and $\pi^{\beta\alpha} = (\pi'\beta\alpha\pi'')$, while $\Phi(\varphi^{\alpha\beta}) > \Phi(\varphi^{\beta\alpha})$ for some other permutations $\varphi^{\alpha\beta} = (\varphi'\alpha\beta\varphi'')$ and $\varphi^{\beta\alpha} = (\varphi'\beta\alpha\varphi'')$.

Given a (partial) permutation $\pi$, we denote the length of $\pi$, i.e., the number of jobs in $\pi$, by $|\pi|$. Let $\{\pi\}$ denote the set of jobs involved in permutation $\pi$. If a job $j \in N$ is associated with a parameter $\gamma_j$, then $\gamma(\pi)$ denotes the sum of all $\gamma$-values for the jobs in $\pi$, i.e.,

$$\gamma(\pi) = \sum_{j \in \{\pi\}} \gamma_j.$$

In particular, for the processing times $p_j$ and the weights $w_j$, we write

$$p(\pi) = \sum_{j \in \{\pi\}} p_j, \;\; w(\pi) = \sum_{j \in \{\pi\}} w_j.$$

### 3.2.1 *Minimizing Total Weighted Completion Time*

Consider problem $1||\sum w_j C_j$ with the objective of minimizing the weighted sum of the completion times, which we denote by $Z(\pi)$. Recall that this function can be minimized by sorting the jobs in non-increasing order of the 1-priorities $\omega(j) = w(j)/p(j)$ (see Sect. 2.2). We show that function $Z(\pi)$ is in fact priority-generating.

**Theorem 3.2** *For problem* $1||\sum w_j C_j$*, the objective function is priority-generating and*

$$\omega(\pi) = \frac{w(\pi)}{p(\pi)} \tag{3.1}$$

*is its priority function.*

*Proof* Take permutations $\pi^{\alpha\beta} = (\pi'\alpha\beta\pi'')$ and $\pi^{\beta\alpha} = (\pi'\beta\alpha\pi'')$ that only differ in the order of the subsequences $\alpha$ and $\beta$ and apply Recipe 3.1.

Notice that if the jobs are processed in accordance with permutation $\pi^{\alpha\beta}$, then the sequence of jobs $\alpha$ starts at time $p\left(\pi'\right)$, the sequence of jobs $\beta$ starts at time $p\left(\pi'\alpha\right) = p\left(\pi'\right) + p(\alpha)$, etc. The value $Z(\alpha)$ is the weighted sum of the completion times of the jobs in $\alpha$, provided that the sequence $\alpha$ starts at time zero. In sequence $\pi^{\alpha\beta}$, each job in $\alpha$ starts $p\left(\pi'\right)$ time units later, so that the weighted sum of the completion times of the jobs in sequence $\alpha$ is given by the sum of $Z(\alpha) + w(\alpha)p\big(\pi'\big)$. This implies that

$$\begin{aligned} Z\big(\pi'\alpha\beta\pi''\big) &= Z(\pi') + Z(\alpha) + w(\alpha)p\big(\pi'\big) + Z(\beta) + w(\beta)\big(p\big(\pi'\big) + p(\alpha)\big) \\ &\quad + Z(\pi'') + w\big(\pi''\big)\big(\big(p\big(\pi'\big) + p(\alpha) + p(\beta)\big)\big); \\ Z\big(\pi'\beta\alpha\pi''\big) &= Z(\pi') + Z(\beta) + w(\beta)p\big(\pi'\big) + Z(\alpha) + w(\alpha)\big(p\big(\pi'\big) + p(\beta)\big) \\ &\quad + Z(\pi'') + w\big(\pi''\big)\big(\big(p\big(\pi'\big) + p(\beta) + p(\alpha)\big)\big). \end{aligned}$$

Define $\Delta := Z(\pi^{\alpha\beta}) - Z(\pi^{\beta\alpha})$. In order to verify that in the case under consideration, the objective function is priority-generating, we need to determine a sufficient condition for the inequality $\Delta \le 0$.

It follows that

$$\Delta = w(\beta)p(\alpha) - w(\alpha)p(\beta),$$

which implies that $\Delta \le 0$ if

$$\frac{w(\beta)}{p(\beta)} \le \frac{w(\alpha)}{p(\alpha)}.$$

This proves that (3.1) is the required priority function. $\square$

### 3.2.2 *Minimizing a Linear Form*

Below, we demonstrate how Recipe 3.2 can be used to disprove that a linear form

$$L(\pi) = \sum_{j=1}^{n} a_{\pi(j)} b_j \tag{3.2}$$

is priority-generating. The problem of minimizing function (3.2) is studied in Sect. 2.1. Recall that if the condition

$$b_1 \ge b_2 \ge \cdots \ge b_n \tag{3.3}$$

holds, Theorem 2.1 asserts that an optimal permutation can be found by sorting the 1-priorities $\omega(j) = -a(j)$ in non-increasing order.

**Theorem 3.3** *Function $L(\pi)$ of the form (3.2) is not priority-generating for arbitrary a-values, even if the b-values are numbered in accordance with (3.3).*

*Proof* Let $\pi^{\alpha\beta} = (\pi'\alpha\beta\pi'')$ and $\pi^{\beta\alpha} = (\pi'\beta\alpha\pi'')$ be two permutations of all jobs that only differ in the order of the subsequences $\alpha$ and $\beta$.

Assume that

$$\left|\pi'\right| = q,\ |\alpha| = u,\ |\beta| = v,\ \left|\pi^{\alpha\beta}\right| = \left|\pi^{\beta\alpha}\right| = r.$$

For convenience, assume that

$$\pi^{\alpha\beta} = \Bigg(\underbrace{\pi(1), \ldots, \pi(q)}_{\pi'}, \underbrace{\pi(q+1), \ldots, \pi(q+u)}_{\alpha},$$
$$\underbrace{\pi(q+u+1), \ldots, \pi(q+u+v)}_{\beta}, \underbrace{\pi(q+u+v+1), \ldots, \pi(r)}_{\pi''}\Bigg),$$

so that

$$\pi^{\beta\alpha} = \Big(\underbrace{\pi(1), \ldots, \pi(q)}, \underbrace{\pi(q+u+1), \ldots, \pi(q+u+v)},$$
$$\underbrace{\pi(q+1), \ldots, \pi(q+u)}, \underbrace{\pi(q+u+v+1), \ldots, \pi(r)}\Big).$$

It follows that

$$L\left(\pi^{\alpha\beta}\right) = \sum_{j=1}^{q} a_{\pi(j)}b_j + \sum_{j=q+1}^{q+u} a_{\pi(j)}b_j + \sum_{j=q+u+1}^{q+u+v} a_{\pi(j)}b_j + \sum_{j=q+u+v+1}^{r} a_{\pi(j)}b_j;$$
$$L\left(\pi^{\beta\alpha}\right) = \sum_{j=1}^{q} a_{\pi(j)}b_j + \sum_{j=q+u+1}^{q+v+u} a_{\pi(j)}b_{j-u} + \sum_{j=q+1}^{q+u} a_{\pi(j)}b_{j+v} + \sum_{j=q+u+v+1}^{r} a_{\pi(j)}b_j.$$

Define $\Delta := L\left(\pi^{\alpha\beta}\right) - L\left(\pi^{\beta\alpha}\right)$. The inequality $\Delta \le 0$ holds if and only if

$$\sum_{j=q+1}^{q+u} a_{\pi(j)}\left(b_j - b_{j+v}\right) \le \sum_{j=q+u+1}^{q+v+u} a_{\pi(j)}\left(b_{j-u} - b_j\right).$$

The latter inequality cannot be rewritten in the required form $\omega(\alpha) \ge \omega(\beta)$, since it depends not only on the sequences $\alpha$ and $\beta$, but also on the number of jobs in sequence $\pi'$. $\square$

### 3.2.3 Minimizing Makespan Versus Total Completion Time

The statement below is proved for the single machine models with constant processing times. It demonstrates that the existence of a priority function $\omega$ for the sum of the completion times $\sum C_j$ implies that $\omega$ is also a priority function for the makespan criterion $C_{\max}$. This statement can easily be extended to scheduling models with various effects applied to the processing times (see Part II).

**Theorem 3.4** *Consider a single machine problem with constant processing times, in which a job may start at a time the previously scheduled job is completed. If $\omega$ is a priority function for the sum of the completion times $F(\pi) = \sum C_j$, then $\omega$ is also a priority function for the makespan $C_{\max}(\pi)$.*

*Proof* Take an arbitrary instance $I$ of a single machine scheduling problem. Consider any two permutations of the jobs of $I$ that are of the form $(\pi'\alpha\beta\pi'')$ and $(\pi'\beta\alpha\pi'')$. We assume without loss of generality that $\omega(\alpha) \geq \omega(\beta)$ (otherwise, we can interchange the labeling of $\alpha$ and $\beta$). Since $\omega$ is a priority function for the sum of the completion times, we have $F(\pi'\alpha\beta\pi'') \leq F(\pi'\beta\alpha\pi'')$.

First, assume that $\omega(\alpha) > \omega(\beta)$. Suppose that $\omega$ is not a priority function for the makespan objective, i.e., $C_{\max}(\pi'\alpha\beta\pi'') > C_{\max}(\pi'\beta\alpha\pi'')$. Define

$$n' := \left\lceil \frac{F(\pi'\beta\alpha\pi'') - F(\pi'\alpha\beta\pi'') + 1}{C_{\max}(\pi'\alpha\beta\pi'') - C_{\max}(\pi'\beta\alpha\pi'')} \right\rceil. \tag{3.4}$$

Extend instance $I$ by adding $n'$ additional jobs, with each additional job $j$ having the processing time $p_j = \varepsilon$, where $\varepsilon$ is a small positive number. Denote the obtained instance by $I'$ and an arbitrary permutation of the added new jobs by $\sigma'$. Assuming that $\varepsilon \to 0$, we deduce that in permutations $\pi'\alpha\beta\pi''\sigma'$ and $\pi'\beta\alpha\pi''\sigma'$, each of the jobs of $\sigma'$ completes at times $C_{\max}(\pi'\alpha\beta\pi''\sigma')$ and $C_{\max}(\pi'\beta\alpha\pi''\sigma')$, respectively. Therefore,

$$\begin{aligned} &\lim_{\varepsilon\to 0}\{F(\pi'\alpha\beta\pi''\sigma') - F(\pi'\beta\alpha\pi''\sigma')\} \\ &= (F(\pi'\alpha\beta\pi'') - F(\pi'\beta\alpha\pi'')) + n'(C_{\max}(\pi'\alpha\beta\pi'') - C_{\max}(\pi'\beta\alpha\pi'')). \end{aligned} \tag{3.5}$$

It follows from (3.4) that the right-hand side of (3.5) is strictly positive. However, this contradicts the fact that $\omega$ is a priority function for the sum of the completion times. Therefore, $C_{\max}(\pi'\alpha\beta\pi'') \leq C_{\max}(\pi'\beta\alpha\pi'')$, as required.

Now, consider the alternative case that $\omega(\alpha) = \omega(\beta)$. Again, suppose that $\omega$ is not a priority function for the makespan objective, i.e., $C_{\max}(\pi'\alpha\beta\pi'') \neq C_{\max}(\pi'\beta\alpha\pi'')$. Since $\omega$ is a priority function for the sum of the completion times, we have $F(\pi'\alpha\beta\pi'') = F(\pi'\beta\alpha\pi'')$. Consider an instance $I''$, which is obtained from instance $I$ similarly to instance $I'$; however, now only one job $n+1$ with $p_{n+1} = \varepsilon$ is added.

Let $\sigma''$ be the sequence that consists of one job $n+1$. Considering the limiting case when $\varepsilon \to 0$, we deduce

$$\begin{aligned} \lim_{\varepsilon\to 0}\{F(\pi'\alpha\beta\pi''\sigma'') - F(\pi'\beta\alpha\pi''\sigma'')\} &= (F(\pi'\alpha\beta\pi'') - F(\pi'\beta\alpha\pi'')) \\ &\quad + (C_{\max}(\pi'\alpha\beta\pi'') - C_{\max}(\pi'\beta\alpha\pi'')). \end{aligned} \tag{3.6}$$

Since the first term in (3.6) is equal to zero and the second term is nonzero, we deduce that $F(\pi'\alpha\beta\pi''\sigma'') \neq F(\pi'\beta\alpha\pi''\sigma'')$. Thus, again, we have a contradiction to the fact that $\omega$ is a priority function for the sum of the completion times. Therefore, $C_{\max}(\pi'\alpha\beta\pi'') = C_{\max}(\pi'\beta\alpha\pi'')$.

We have now verified that for the makespan objective function, the conditions of Definition 3.2 are satisfied, and therefore, $\omega$ is its priority function. □

As the contrapositive statement to Theorem 3.4, we obtain the following corollary.

**Corollary 3.1** *Consider a single machine problem with constant processing times, in which a job may start at a time the previously scheduled job is completed. If the makespan objective function is not priority-generating, then neither is the sum of the completion times priority-generating.*

In the next section, we present an algorithm for minimizing a priority-generating function subject to series-parallel constraints.

## 3.3 Minimizing Priority-Generating Functions Under Series-Parallel Constraints

Let $\Phi(\pi)$ be a priority-generating function defined over a poset $P = (N, R)$ and $\omega(\pi)$ be the corresponding priority function. Consider the problem of minimizing $\Phi(\pi)$ over $P = (N, R)$, provided that the poset $P$ is defined by series-parallel precedence constraints. Throughout this section, we assume that the constraints are given in the form of a decomposition tree.

The key concept that we need for an algorithm that minimizes a priority-generating function is that of a job module.

**Definition 3.3** Given a poset $P = (N, R)$, a subset $M \subseteq N$ is a *(job) module* of $P$ if for every job $k \in N \backslash M$ one of the following holds:

**(a)** $k \to i$ for all $i \in M$, or
**(b)** $i \to k$ for all $i \in M$, or
**(c)** $i \sim k$ for all $i \in M$.

Clearly, each job $j \in N$ is a module, and the whole set $N$ is a module. For example, for the precedence constraints given by the graph in Fig. 3.2b, the set $\{3, 4\}$ is a module. Indeed, job 1 precedes both jobs 3 and 4, while each of the jobs 5, 6, and 7 is a successor of both jobs 3 and 4; moreover, job 2 is independent of both jobs 3 and 4. On the other hand, the set $\{4, 5\}$ is not a module, since $2 \to 5$ but $2 \sim 4$.

Figure 3.6 shows a decomposition tree of the digraph in Fig. 3.2b. Use this graph for checking the following property.

**Lemma 3.1** *For an SP-graph, a subtree of its decomposition tree defines a module.*

Thus, we can identify the following non-trivial modules of the digraph in Fig. 3.2b: $\{3, 4\}$, $\{2, 3, 4\}$, $\{1, 2, 3, 4\}$, $\{1, 2, 3, 4, 5\}$, $\{6, 7\}$.

The algorithm that minimizes a priority-generating function under series-parallel precedence constraints is based on the following statement.

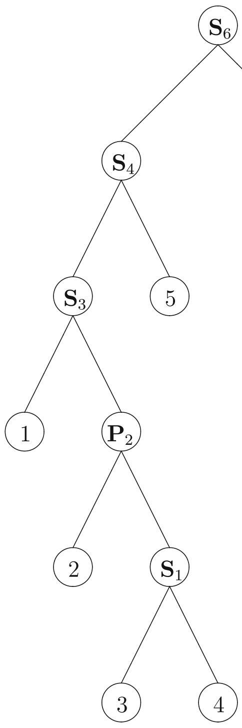


**Fig. 3.6** Decomposition tree for the graph in Fig. 3.2b

**Theorem 3.5** *Let $M$ be a module of a series-parallel graph $G = (N, U)$ and $\sigma$ be a permutation that minimizes function $\Phi$ over set $M$. Then, there exists a permutation that minimizes $\Phi$ over set $N$, in which the jobs of $M$ appear in the same order as in $\sigma$.*

Theorem 3.5 allows treating subsequences of $\sigma$ as modules. It is convenient to think of a module as consisting of subsequences (or strings) of jobs, rather than of individual jobs.

The algorithm scans the decomposition tree of the given digraph from leaves to the root. In each iteration, a composition operation (parallel or series, depending on the type of the node) is performed, forming a new module of two initial modules.

In the case of parallel composition of two modules $M'$ and $M''$, a new module $M$ is formed, which is the union of the elements of the two initial modules. For the obtained module $M$, an optimal sequence $\sigma$ can be obtained by sorting its elements in non-increasing order of their priorities. Such a sequence respects the given precedence graph.

If series composition of two modules $M'$ and $M''$ is to be performed, then the resulting module $M$ is obtained as the union of $M'$ and $M''$, provided that the smallest priority of an element of $M'$ is larger than the largest priority of an element in $M''$. Otherwise, the elements of $M'$ with smaller priorities are merged with the elements of $M''$ with larger priorities to form a string with the priority that is smaller than the smallest priority of the remaining elements of $M'$ and larger than the largest priority of the remaining elements of $M''$. The new module is formed to consist of the derived string and the remaining elements of the two initial modules.

Formally, the procedure for series composition is outlined below. In the description of the procedure, the elements of modules (i.e., jobs and strings) are denoted by the Greek letters $\kappa$, $\lambda$, $\mu$, etc. For a module, an element $\kappa$ is called $\omega$-minimal ($\omega$-maximal) if $\omega(\kappa)$ is smaller (respectively, larger) than the priority of all other elements of the module. For consistency, it is assumed that module $M'$ always contains a dummy element with priority $+\infty$, while module $M''$ always contains a dummy element with priority $-\infty$.

**Procedure SerComp**$(M', M'')$

Input: Two modules $M'$ and $M''$

Output: Module $M$, a result of series composition of $M'$ and $M''$

**Step 1**. Find a $\omega$-minimal element $\lambda$ in $M'$ and a $\omega$-maximal element $\mu$ in $M''$. If $\omega(\lambda) > \omega(\mu)$, then return $M = M' \cup M''$. Otherwise, remove the elements $\lambda$ and $\mu$ from their respective modules (i.e., update $M' := M' \backslash \{\lambda\}$ and $M'' := M'' \backslash \{\mu\}$), form the string $\kappa := (\lambda, \mu)$, and compute its priority $\omega(\kappa)$.
**Step 2**. Find a $\omega$-minimal element $\lambda$ in $M'$. While $\omega(\lambda) \leq \omega(\kappa)$, do the following: Remove $\lambda$ from $M'$ (i.e., update $M' := M' \backslash \{\lambda\}$), include it to string $\kappa$ (i.e., update $\kappa := (\lambda, \kappa)$), and compute the priority $\omega(\kappa)$.
**Step 3**. Find a $\omega$-maximal element $\mu$ in $M''$. While $\omega(\mu) \leq \omega(\kappa)$, do the following: Remove $\mu$ from $M''$ (i.e., update $M'' := M'' \backslash \{\mu\}$), include it to string $\kappa$ (i.e., update $\kappa := (\kappa, \mu)$), and compute the priority $\omega(\kappa)$.
**Step 4**. Return $M = M' \cup M'' \cup \{\kappa\}$.

The overall algorithm for minimizing a priority-generating function under series-parallel precedence constraints given by the decomposition tree can be described as follows.

**Algorithm SerPar**

Input: An instance of a scheduling problem with $n$ jobs to minimize the priority-generating objective function $\Phi$, a decomposition tree of the precedence digraph, and the priority function $\omega$ for $\Phi$

Output: Permutation $\pi^*$ of jobs that minimizes $\Phi$

**Table 3.1** Instance for Example 3.1

| Jobs | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| $p_j$ | 2 | 6 | 4 | 3 | 2 | 7 | 2 |
| $w_j$ | 5 | 2 | 3 | 4 | 8 | 2 | 4 |
| $\omega(j)$ | $\frac{5}{2}$ | $\frac{1}{3}$ | $\frac{3}{4}$ | $\frac{4}{3}$ | 4 | $\frac{2}{7}$ | 2 |

**Table 3.2** Running Algorithm SerPar for Example 3.1

| Iteration | Operation | Procedure SerComp | | | | | | Output |
|---|---|---|---|---|---|---|---|---|
| | | Step | $M'$ | $M''$ | $\lambda$ | $\mu$ | $\kappa,\ \omega(\kappa)$ | |
| 1 | $3\mathbf{S}_1 4$ | 1 | $\{3\}$ | $\{4\}$ | 3 | 4 | $(3, 4),\ \omega = 1$ | |
| | | 2 | $\varnothing$ | $\varnothing$ | – | – | $(3, 4)$ | |
| | | 3 | $\varnothing$ | $\varnothing$ | – | – | $(3, 4)$ | |
| | | 4 | | | | | | $M_1 = \{(3, 4)\}$ |
| 2 | $2\mathbf{P}_2 M_1$ | | $\{2\}$ | $M_1$ | | | | $M_2 = \{2, (3, 4)\}$ |
| 3 | $1\mathbf{S}_3 M_2$ | 1 | $\{1\}$ | $M_2$ | 1 | $(3, 4)$ | | $M_3 = \{1, 2, (3, 4)\}$ |
| 4 | $M_3\mathbf{S}_4 5$ | 1 | $M_3$ | $\{5\}$ | 2 | 5 | $(2, 5),\ \omega = \frac{5}{4}$ | |
| | | 2 | $\{1, (3, 4)\}$ | $\varnothing$ | $(3, 4)$ | – | $(3, 4, 2, 5), \omega = \frac{17}{15}$ | |
| | | 2 | $\{1\}$ | $\varnothing$ | 1 | – | $(3, 4, 2, 5)$ | |
| | | 3 | $\{1\}$ | $\varnothing$ | – | – | $(3, 4, 2, 5)$ | |
| | | 4 | | | | | | $M_4 = \{1, (3, 4, 2, 5)\}$ |
| 5 | $6\mathbf{P}_5 7$ | | $\{6\}$ | $\{7\}$ | | | | $M_5 = \{6, 7\}$ |
| 6 | $M_4\mathbf{S}_6 M_5$ | 1 | $M_4$ | $M_5$ | $(3, 4, 2, 5)$ | 7 | $(3, 4, 2, 5, 7),\ \omega = \frac{21}{17}$ | |
| | | 2 | $\{1\}$ | $\{6\}$ | 1 | – | $(3, 4, 2, 5, 7)$ | |
| | | 3 | $\{1\}$ | $\{6\}$ | | $(3, 4, 2, 5, 7)$ | $(3, 4, 2, 5, 7)$ | |
| | | 4 | | | | | | $M_6 = \{1, (3, 4, 2, 5, 7), 6\}$ |

**Step 1**. Scanning the decomposition tree from leaves to the root, for $k = 1$ to $n-1$, do the following:

- **(a)** Identify operation $M'\mathbf{O}_k M''$, where $M'$ and $M''$ are two modules and $\mathbf{O}_k$ corresponds to an operation node of the decomposition tree.
- **(b)** If $\mathbf{O}_k$ is an operation of parallel composition, then define $M_k := M' \cup M''$.
- **(c)** If $\mathbf{O}_k$ is an operation of series composition, then $M_k$ is the output of Procedure SerComp$\left(M', M''\right)$.

**Step 2**. Sort the elements of the module $M_{n-1}$ in non-increasing order of its priorities. Output $\pi^*$ as the obtained permutation of the original jobs.

Algorithm SerPar can be implemented in $O(n \log n)$ time, provided that the priority $\omega(\pi)$ for a given partial sequence $\pi$ can be computed in $O(n)$ time.

*Example 3.1* In order to illustrate Algorithm SerPar, consider the problem of minimizing of the weighted sum of the completion times on a single machine, i.e., problem $1|SP - prec|\sum w_j C_j$. The numerical parameters of the seven jobs, as well as their 1-priorities $\omega(j)$, are given in Table 3.1. The precedence relation is represented by the digraph in Fig. 3.2b. Its decomposition tree is shown in Fig. 3.6, and the operation nodes are numbered in the order of iterations of Algorithm SerPar. The run of Algorithm SerPar for this instance is presented in Table 3.2. For iterations 1, 4, and 6, the steps of Procedure SerComp are shown in details. The final permutation (1, 3, 4, 2, 5, 7, 6) is determined by module $M_6$.

## 3.4 Bibliographic Notes

Scheduling problems with precedence constraints have always been among the most studied types of problems. The first positive results have been derived for rather simple types of reduction graphs, such as chains and trees. The seminal paper by Lawler (1978) has given the first example of a polynomially solvable problem with series-parallel precedence constraints. That paper initiated extensive studies of scheduling problems under series-parallel precedence conducted in the late 1970s–early 1980s, which have resulted in an elegant theory briefly overviewed in this chapter.

Theorem 3.1 is independently proved by Gordon (1981) and Valdes et al. (1982); however, already Sidney (1975) and Lawler (1978) present the $Z$-graph as the simplest digraph that is not series-parallel. Alternative conditions that are necessary and sufficient for a dag $G = (X, U)$ to be series-parallel are proved in Gordon (1981) (see also Gordon et al. (2005)). Theorem 3.1 is the basis of an algorithm that recognizes an SP-graph in $O\left(|X|^2\right)$ time (see Shafransky and Yanova (1980) and Valdes et al. (1982)).

The concept of a priority-generating function has been independently introduced by Shafransky (1978a), Reva (1979), Monma and Sidney (1979), and Burdyuk and

Reva (1980). We mainly follow Chap. 3 of the book by Tanaev et al. (1984) for a systematic exposition of related issues.

A formal proof of Theorem 3.2 is given in Tanaev et al. (1984); however, already Horn (1972) and Lawler (1978) essentially use function (3.1) as a priority function for a subsequence of jobs. Theorem 3.3 is proved in Tanaev et al. (1984), while Theorem 3.4 and Corollary 3.1 are due to Gordon et al. (2008) (for scheduling models with a deterioration effect).

Definition 3.3 of a job module is given by Sidney (1975), who has established various properties that lead to a decomposition algorithm for minimizing functions under series-parallel constraints, including Lemma 3.1 and Theorem 3.5. There are multiple versions of algorithms that solve various scheduling problems under series-parallel precedence constraints; here, we only mention algorithms by Lawler (1978) for minimizing the weighted sum of the completion times on a single machine and by Gordon and Shafransky (1978), Monma (1979), and Sidney (1979) for the two-machine flow shop problem to minimize the makespan. The algorithms for minimizing an arbitrary priority-generating function in $O(n \log n)$ time are given by Shafransky (1978b), Monma and Sidney (1979), and Tanaev et al. (1984). In our description of Algorithm SerPar, we mainly follow Lawler (1978).

Queyranne and Wang (1991) present a linear programming formulation of problem $1|prec|\sum w_j C_j$ and prove that in the case of series-parallel precedence constraints, their formulation completely describes the scheduling polyhedron. This fact, as well as an alternative proof of the algorithm by Lawler (1978) for problem $1|SP - prec|\sum w_j C_j$, is given in Goemans and Williamson (2000), who use an elegant reasoning based on the so-called two-dimensional Gantt charts.

There is a more general type of precedence constraints that can be decomposed into job modules of special structure. Provided that the objective function possesses certain properties, it can be minimized over a poset in polynomial time. For completeness, below we give a brief discussion of this issue.

Let $G = (N, U)$ be a dag corresponding to poset $P = (N, R)$. Replacing all arcs of its transitive closure $G_T$ by undirected edges, we obtain the (undirected) graph $\widetilde{G} = (N, E)$. We may assume that $\widetilde{G} = (N, E)$ is given to us in the form of the adjacency matrix. A module $M$ is a set of vertices that is indistinguishable in graph $\widetilde{G}$ by the vertices outside $M$; that is, in graph $\widetilde{G}$, any vertex in $N \setminus M$ either is adjacent to all vertices of $M$, or is adjacent to no vertex in $M$.

The *complement* graph of $\widetilde{G} = (N, E)$ is the graph $(N, E')$, where $(u, v) \in E'$ if and only if $(u, v) \notin E$. A graph is *complement-connected* if its complement graph is connected. There are three distinct types of modules: parallel, series, and neighborhood. *Parallel* modules are characterized by the property that the subgraph induced by the vertices of the module is not connected. A module is a *series* module if the subgraph induced by the vertices of the module is not complement-connected. In a *neighborhood* module, the subgraph induced by the vertices of the module is both connected and complement-connected.

For a poset $P = (N, R)$, a subset $S \subseteq N$ is *initial* if for each $i \in S$, all predecessors of $i$ are also in $S$. For an initial set $S$, define the set

$$I(S) := \{j|j \in S,\ j \text{ has no successors in } S\}.$$

The maximum size of any set $I(S)$ is called the *width* of $P$. If the poset is decomposed into modules, the size of any module does not exceed the width of the original poset. The modular decomposition procedure of the original poset $P$ is implemented as an iterative process that is represented by a special data structure, a composition tree, For details on the decomposition process and on the construction of a composition tree, see Buer and Möhring (1983), Sidney and Steiner (1986), and Muller and Spinrad (1989). Notice that the decomposition algorithm in Muller and Spinrad (1989) requires $O(n^2)$ time.

One of the conditions under which an objective function $\Phi$ can be minimized over a poset is the so-called job module property: If $\sigma$ is a permutation that minimizes function $\Phi$ over set $M$, then there exists a permutation that minimizes $\Phi$ over set $N$, in which the jobs of $M$ appear in the same order as in $\sigma$. In fact, Theorem 3.5 states that this property holds for series-parallel precedence constraints. For more general constraints, it has to be assumed (see Sidney and Steiner (1986) and Monma and Sidney (1987)). The corresponding sequencing algorithms are based on efficient procedures by Möhring and Rademacher (1984) and Muller and Spinrad (1989).

Another condition is that function $\Phi$ can be computed recursively over initial sets by a dynamic programming algorithm. If this property is combined with the job module property, then function $\Phi$ can be minimized over a poset of width $w$ in $O\left(n^{w+1}\right)$ time, as demonstrated by Sidney and Steiner (1986) who combine the dynamic programming algorithm with the decomposition algorithm by Möhring and Rademacher (1984, 1985).

Scheduling problems under arbitrary precedence constraints are normally NP-hard, even if the objective is priority-generating. For example, Lawler (1978) proves the NP-hardness of minimizing the weighted sum of the completion times on a single machine, i.e., of problem $1|prec|\sum w_jC_j$. Other examples of NP-hard scheduling problems under precedence constraints are contained in Abdel-Wahab and Kameda (1978), Lenstra and Rinnooy Kan (1978), and Monma (1980, 1981) (see Tanaev et al. (1984) for a review). Methods of reducing the search for an optimal permutation under arbitrary precedence constraints can be found in Monma (1981) and Tanaev et al. (1984).

## References

Abdel-Wahab HM, Kameda T (1978) Scheduling to minimize maximum cumulative cost subject to series-parallel precedence constraints. Oper Res 26:141–158

Buer H, Möhring RH (1983) A fast algorithm for the decomposition of graphs and posets. Math Oper Res 8:170–184

Burdyuk VY, Reva VN (1980) A method for optimizing functions over permutations under constraints. Kibernetika (Kiev) No 1: 99–103 (in Russian)

Goemans MX, Williamson DP (2000) Two-dimensional Gantt charts and a scheduling algorithm of Lawler. SIAM J Discr Math 13:281–294

Gordon VS (1981) Some properties of series-parallel graphs. Vestsi Akademii Navuk BSSR, Seria Fizika-Matematychnykh Navuk (Proc Acad Sci BSSR, Phys Math Sci) No 1: 18–23 (in Russian)

Gordon VS, Proth J-M, Strusevich VA (2005) Single machine scheduling and due date assignment under series-parallel precedence constraints. Cent Eur J Oper Res 13:15–35

Gordon VS, Potts CN, Strusevich VA, Whitehead JD (2008) Single machine scheduling models with deterioration and learning: Handling precedence constraints via priority generation. J Sched 11:357–370

Gordon VS, Shafransky YM (1977) On a class of scheduling problems with partially ordered set of jobs. In: Proceedings of the 4th All-Union conference on the problems of theoretical cybernetics, Novosibirsk, p 101–103 (in Russian)

Gordon VS, Shafransky YM (1978) Optimal ordering with series-parallel precedence constraints. Doklady Akademii Nauk BSSR (Repts Acad Sci BSSR) 22:244–247 (in Russian)

Horn AW (1972) Single-machine job sequencing with treelike precedence ordering and linear delay penalties. SIAM J Appl Math 23:189–202

Lawler EL (1978) Sequencing jobs to minimize total weighted completion time subject to precedence constraints. Ann Discr Math 2:75–90

Lenstra JK, Rinnooy Kan AHG (1978) Complexity of scheduling under precedence constraints. Oper Res 26:22–35

Möhring RH, Rademacher FJ (1984) Substitution decomposition for discrete structures and connections with combinatorial optimization. Ann Discr Math 19:257–356

Möhring RH, Rademacher FJ (1985) Generalized results on the polynomiality of certain weighted sum scheduling problems. Meth Oper Res 49:405–417

Monma CL (1979) The two-machine maximum flow time problem with series-parallel precedence constraints: an algorithm and extensions. Oper Res 27:792–797

Monma CL (1980) Sequencing to minimize the maximum job cost. Oper Res 28:942–951

Monma CL (1981) Sequencing with general precedence constraints. Discr Appl Math 3:137–150

Monma CL, Sidney JB (1979) Sequencing with series-parallel precedence constraints. Math Oper Res 4:215–234

Monma CL, Sidney JB (1987) Optimal sequencing via modular decomposition: characterization of sequencing functions. Math Oper Res 12:22–31

Muller JH, Spinrad J (1989) Incremental modular decomposition: Polynomial algorithms. J ACM 36:1–19

Queyranne M, Wang Y (1991) Single-machine scheduling polyhedra with precedence constraints. Math Oper Res 16:1–20

Reva VN (1979) On an algorithm for optimizing functions over the permutations of a partially ordered set. In: Actual problems of computers and programming, Dnepropetrovsk, pp 92–95 (in Russian)

Shafransky YM (1978a) On optimal sequencing for deterministic systems with a tree-like partial order. Vestsi Akademii Navuk BSSR, Seria Fizika-Matematychnykh Navuk (Proc Acad Sci BSSR, Phys Math Sci) No 2: 120 (in Russian)

Shafransky YM (1978b) Optimization for deterministic scheduling systems with a tree-like partial order. Vestsi Akademii Navuk BSSR, Seria Fizika-Matematychnykh Navuk (Proc Acad Sci BSSR, Phys Math Sci) No 2: 119 (in Russian)

Shafransky YM (1980) On a problem of minimizing functions over a set of permutations of partially ordered elements: I. Vestsi Akademii Navuk BSSR, Seria Fizika-Matematychnykh Navuk (Proc Acad Sci BSSR, Phys Math Sci) No 5: 152 (in Russian)

Shafransky YM, Yanova OV (1980) The recognition and decomposition of series-parallel graphs. In: Library of programs for solving extremal problems, Minsk, pp 17–24 (in Russian)

Sidney JB (1975) Decompositions algorithms for single-machine sequencing with precedence relations and deferral costs. Oper Res 22:283–298

Sidney JB (1979) The two-machine maximum flow time problem with series-parallel precedence relations. Oper Res 27:782–791

Sidney JB (1981) A decomposition algorithm for sequencing with general precedence constraints. Math Oper Res 6:190–204

Sidney JB, Steiner G (1986) Optimal sequencing by modular decomposition: Polynomial algorithms. Oper Res 34:606–612

Smith WE (1956) Various optimizers for single state production. Naval Res Logist Quart 3:66–69

Tanaev VS, Gordon VS, Shafransky YM (1984) Scheduling theory. Single-stage systems. Nauka, Moscow (in Russian); Kluwer, Dordrecht, 1994 (in English)

Valdes JR, Tarjan RE, Lawler EL (1982) The recognition of series-parallel digraphs. SIAM J Comp 11:361–370

# Chapter 4
# Relevant Boolean Programming Problems

Quite often, an algorithm that finds either an exact or an approximate solution to a scheduling problem can be derived from a reformulation of the original problem in terms of another problem of combinatorial optimization, e.g., a Boolean programming problem. In this chapter, we review some of the most popular Boolean programming problems which are relevant to the content of this book.

In Sect. 4.1, we focus on different versions of the linear assignment problem, including its forms with a square and rectangular cost matrix. We present several polynomial-time algorithms, including Dinic's algorithm for the rectangular assignment problem. Besides, we link the linear assignment problem with a product cost matrix to the problem of minimizing a linear form, studied in Sect. 2.1. In Sect. 4.2, we address the linear knapsack problem and its versions, such as the subset-sum problem. Here, the main focus is on the design of fully polynomial-time approximation schemes (FPTASs) by converting pseudopolynomial-time dynamic programming algorithms. Section 4.3 is devoted to the problem of minimizing a specific quadratic function of Boolean variables, known as the half-product. This problem and its variants, without and with an additional linear knapsack constraint, are known to serve as mathematical models for many scheduling problems. Section 4.4 addresses problems with the objective that is a special form of the half-product. The main topic of Sects. 4.3 and 4.4 is also the design of FPTAS.

## 4.1 Linear Assignment Problems

The *linear assignment problem* (LAP) is one of the most popular and intensively studied problems in combinatorial optimization. Given a square or rectangular cost matrix, the problem is to match each row to a different column in such a way that the sum of the selected cost values is minimized.

A possible meaningful interpretation of the LAP is as follows. Suppose that $n$ jobs need to be assigned to $m \geq n$ candidates, exactly one job per candidate. It is

V.A. Strusevich and K. Rustogi, *Scheduling with Times-Changing Effects and Rate-Modifying Activities*, International Series in Operations Research & Management Science 243, DOI 10.1007/978-3-319-39574-6_4

known that $c_{ij}$ is the cost of assigning job $i$ to candidate $j$. The objective is to find an assignment so that the total cost is minimized.

It is convenient to arrange all costs as an $n \times m$ cost matrix $\mathbf{C} = \left(c_{ij}\right)$, where the $i$th row, $1 \leq i \leq n$, contains the elements $c_{i1}, c_{i2}, \ldots, c_{im}$, and the $j$th column, $1 \leq j \leq m$, contains the elements $c_{1j}, c_{2j}, \ldots, c_{mj}$. It is required to select $n$ elements, exactly one from each row and at most one from each column, so that their sum is minimized. The $n \times m$ LAP is known as a *rectangular assignment problem* and can be formulated as a Boolean programming problem in the following way:

$$\begin{aligned} \text{minimize} \ & \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} \\ \text{subject to} \ & \sum_{j=1}^{m} x_{ij} = 1, \quad i = 1, 2, \ldots, n; \\ & \sum_{i=1}^{n} x_{ij} \leq 1, \quad j = 1, 2, \ldots, m; \\ & x_{ij} \in \{0, 1\}, \quad i = 1, 2, \ldots, n; \ j = 1, 2, \ldots, m. \end{aligned} \tag{4.1}$$

A special case of the $n \times m$ LAP is the $n \times n$ LAP with a *square* cost matrix. Below, we give its formulation in terms of Boolean programming:

$$\begin{aligned} \text{minimize} \ & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \\ \text{subject to} \ & \sum_{j=1}^{m} x_{ij} = 1, \quad i = 1, 2, \ldots, n; \\ & \sum_{i=1}^{n} x_{ij} = 1, \quad j = 1, 2, \ldots, n; \\ & x_{ij} \in \{0, 1\}, \quad i = 1, 2, \ldots, n, \ j = 1, 2, \ldots, n. \end{aligned} \tag{4.2}$$

In the next subsection, we present two algorithms that can be used to solve a rectangular assignment problem of the form (4.1).

### 4.1.1 Methods for Solving the Rectangular Assignment Problem

In this subsection, we reproduce the main steps of two algorithms known for solving an $n \times m$ rectangular LAP of the form (4.1). When applied to a square $n \times n$ cost matrix, i.e. when the problem under consideration is of the form (4.2), these algorithms are able to provide a solution in $O\left(n^3\right)$ time.

We use the matrix terminology. Given a matrix $\mathbf{C} = \left(c_{ij}\right)_{n \times m}$ with $m \geq n$, either a row or a column is called a *line* of **C**; a collection of $n$ elements, no two of which belong to the same line is called a *diagonal*; and a collection of $k < n$ elements, no two of which belong to the same line is called a *subdiagonal*. In the linear assignment problem, it is required to find a diagonal with the smallest sum of its elements.

The first of the presented algorithms reduces the original (positive) elements of the cost matrix on a line-by-line basis, so that some of them become zero. Two zeros that do not belong to the same line are called *independent*. There are two types of labels applied to a zero: It can be *starred* to become $0^*$ or *primed* to become $0'$. During the run of the algorithm, some lines are said to be *covered*. In all iterations of the algorithm, the starred zeros are independent, and their number is equal to the number of the covered lines, with each covered line containing exactly one $0^*$. The algorithm stops having found $n$ starred zeros in the current matrix. The primed zeros in a current partial solution are seen as potential candidates to become starred zeros.

**Algorithm LAPBL**

**Step 0**. Considering the rows of matrix **C** in the order of their numbering, subtract the smallest element from each element in the row.

**Step 1**. Search for a zero, $Z$, in the matrix. If there is no starred zero in its row or column, star $Z$. Repeat for each zero in the matrix.

**Step 2**. Cover every column containing a $0^*$. If $n$ columns are covered, the starred zeros form the desired independent set; otherwise, go to Step 3.

**Step 3**. Choose a non-covered zero and prime it; then, consider the row containing the primed zero. If there is no starred zero in this row, go to Step 4. If there is a starred zero $Z$ in this row, cover this row and uncover the column of $Z$. Repeat until all zeros are covered. Go to Step 5.

**Step 4**. Construct a sequence of alternating starred and primed zeros as follows: Let $Z_0$ denote an uncovered $0'$. Let $Z_1$ denote a $0^*$ in $Z_0$'s column (if any). Let $Z_2$ denote the $0'$ in $Z_1$'s row. Continue in a similar way until the sequence stops at a $0'$, which has no $0^*$ in its column. In the obtained sequence, unstar each starred zero and star each primed zero. Erase all primes and uncover every line. Return to Step 2.

**Step 5**. Let $h$ denote the smallest non-covered element of the current matrix. Add $h$ to each covered row, and then subtract $h$ from each uncovered column. Return to Step 3 without altering any asterisks, primes, or covered lines.

Algorithm LAPBL can be implemented in $O\left(n^2 m\right)$ time. An iteration of Algorithm LAPBL is considered complete when all zeros are covered by the end of Step 3. After this, a transition is made to Step 5, where we search for the minimal elements in the uncovered part of the matrix and convert them to zero. At the end of an iteration, one of the two outcomes is possible: New $0^*$s are either added to the matrix, or not. If the total number of $0^*$s in the matrix is less than $n$, the existing $0^*$s represent a partial solution to the assignment problem. If the total number of $0^*$s in the matrix is equal to $n$, then the solution is considered complete and the optimal assignment is given by the positions occupied by the $0^*$s.

Another algorithm for solving a rectangular LAP is based on a concept of a *potential*. In any step of the algorithm, each column $j$, $1 \le j \le m$ associated with a potential $\Delta_j$. Let $\mathbf{\Delta} = (\Delta_1, \dots, \Delta_m)$ be an array of these potentials. We introduce a concept of $\Delta$-*difference*

$$c_{i,j_1} \overset{\Delta}{-} c_{i,j_2} := \left(c_{i,j_1} - \Delta_{j_1}\right) - \left(c_{i,j_2} - \Delta_{j_2}\right).$$

Along this definition, a $\Delta$-*minimal element* is the element which, when subtracted from any other element from the same row, returns a nonnegative value of $\Delta$-difference.

During an iteration of the algorithm, some columns and rows will be *highlighted* or *labeled*. The highlighted elements form a subdiagonal of the matrix. A line with no highlighted or labeled elements is called *free*. Let $R$ denote the set of free and/or labeled columns. Let $Q$ be the set of labeled rows. In the beginning of each iteration, all rows and columns are seen as non-labeled. For each row $i$, let $c_{i,\pi(i)}$ be the $\Delta$-minimal element, and $\delta_i$ is the $\Delta$-difference between the element that is $\Delta$-minimal among the elements of this row located in the columns in $R$ and $c_{i,\pi(i)}$. Define $\delta$ as the smallest $\delta_i$ over all non-labeled rows.

In the beginning of an iteration, we have a subdiagonal of $\ell$ elements and an array of potentials such that:

**(i)** Each highlighted element (i.e., an element of the current subdiagonal) of a non-labeled row is $\Delta$-minimal in its row;
**(ii)** The potentials of all free columns are equal.

**Algorithm LAPD**

**Step 0**. Find the smallest element in each row. For each column that contains more than one such element, highlight any one of them. The highlighted elements form the initial subdiagonal. Set all potentials to zero. Since there are no labeled columns to begin with, $R$ denotes the set of free columns. Compute $\delta$ and identify the *extremal* element $c_{i_1 j_1}$, i.e., the element for which the difference that defines $\delta_i$ is equal to $\delta$.

**Step 1**. If $\ell = n$, and conditions (i) and (ii) are satisfied, accept the found diagonal as a solution to the initial problem. If $\ell < n$, check that conditions (i) and (ii) are satisfied, then go to Step 2 with $k = 1$.

**Step 2**. For current value of $k$, $1 \le k \le n$, do:

**(a)** Increase the potentials of all columns in $R$ by $\delta$ (element $c_{i_k j_k}$ will become $\Delta$-minimal in its row, while each highlighted element remains $\Delta$-minimal in its row).

- If row $i_k$ is not free, then label row $i_k$ by "$j_k$" (the number of the column of the extremal element $c_{i_k j_k}$). In row $i_k$, find the highlighted (subdiagonal) element $c_{i_k, j^k}$ and label column $j^k$ by "$i_k$". Go to Step 2(b).

- If row $i_k$ is free, then update the subdiagonal as follows. Include $c_{i_k j_k}$ into the current subdiagonal (highlight it). If column $j_k$ is free, then complete the iteration, update $\ell := \ell + 1$, and go to Step 1. If it is not free, it has a label "$i$ " and contains the highlighted element $c_{i,j_k}$. Remove that element from the subdiagonal. Row $i$ is labeled "$j$". Include element $c_{i,j}$ into the subdiagonal. This process is repeated until a free column occurs. Once a free column is found, a new element is highlighted and the iteration completed. Remove all labels, update $\ell := \ell + 1$, and go to Step 1.

**(b)** Update the sets $R$ and $Q$. Compute $\delta$ and identify the extremal element $c_{i_{k+1} j_{k+1}}$. Update $k := k + 1$ and go to Step 2.

It is useful to employ a special data structure, a dictionary $q$ that consists of records $q_i$ defined for each non-labeled row $i$. Record $q_i$ stores (i) the value $\delta_i$ and (ii) column number $j(i)$ such that element $c_{i,j(i)}$ is $\Delta$-minimal among the elements of row $i$ located in the columns in $R$. Dictionary $q$ is created before the preliminary stage and before the start of each iteration. When moving from one step to the next within an iteration, the dictionary can be updated in $O(n)$ time. In particular, the value $\delta_i := \min\left\{\delta_i, c_{i,\pi(i)} \overset{\Delta}{-} c_{ij}\right\}$, where $j$ is the column labeled in the previous step.

If $m = n$, the algorithm solves the assignment problem in $O\left(n^3\right)$ time. If $m > n$, it can be implemented in $O\left(n^3 + nm\right)$ time. Below, we provide an example that walks through each step of Algorithm LAPD.

*Example 4.1* Consider a rectangular assignment problem with a cost matrix

$$\mathbf{C} = \begin{pmatrix} 1 & 2 & 3 & 4 & 8 \\ 1 & 1 & 3 & 4 & 8 \\ 1 & 5 & 5 & 6 & 8 \\ 2 & 5 & 6 & 8 & 2 \end{pmatrix}.$$

**Iteration 0**. The following iterations as outlined in Algorithm LAPD will enable us to find an optimal solution to the given assignment problem.

| $\Delta_j$ | 0 | 0 | 0 | 0 | 0 | *Step 0* | | |
|---|---|---|---|---|---|---|---|---|
| Label | – | – | – | – | – | $\Delta_{\min}$ | $\Delta_{\min R}$ | $\delta_i$ |
| – | $\boxed{1}$ | 2 | $3^*$ | 4 | 8 | 1 | 3 | $2^*$ |
| – | 1 | $\boxed{1}$ | 3 | 4 | 8 | 1 | 3 | 2 |
| – | 1 | 5 | 5 | 6 | 8 | 1 | 5 | 4 |
| – | 2 | 5 | 6 | 8 | $\boxed{2}$ | 2 | 6 | 4 |
| $R \in \{3, 4\}$, $Q \in \{\emptyset\}$, $\delta = 2$, $c_{i_1 j_1} = c_{13}$ | | | | | | | | |

In this tableau and in other tableaux related to the solution of Example 4.1, the elements in boxes represent the highlighted elements, i.e., the elements in the subdiagonal, and the starred element represents the extremal element $c_{i_1 j_1}$. For each non-labeled row, Column $\Delta_{\min}$ contains the $\Delta$-minimal element in that row, while Column $\Delta_{\min R}$ contains the $\Delta$-minimal element among those located in that row and in the columns of set $R$. Notice that $\ell = 3 < n = 4$, and conditions (i) and (ii) of a subdiagonal are satisfied.

**Iteration 1**. We proceed to a new iteration with $k = 1$.

| $\Delta_j$ | 0 | 0 | 2 | 2 | 0 | $k = 1$ | | |
|---|---|---|---|---|---|---|---|---|
| Label | 1 | – | – | – | – | $\Delta_{\min}$ | $\Delta_{\min R}$ | $\delta_i$ |
| 3 | $\boxed{1}$ | 2 | 3 | 4 | 8 | – | – | – |
| – | 1 | $\boxed{1}$ | 3 | 4 | 8 | 1 | 1 | 0 |
| – | 1* | 5 | 5 | 6 | 8 | 1 | 1 | 0* |
| – | 2 | 5 | 6 | 8 | $\boxed{2}$ | 2 | 2 | 0 |
| $R \in \{1, 3, 4\}$, $Q \in \{1\}$, $\delta = 0$, $c_{i_2 j_2} = c_{31}$ | | | | | | | | |

Now, the iteration requires us to go back to Step 2 with $k = 2$. The potentials do not change since $\delta = 0$. We include the current extremal element $c_{i_2 j_2} = c_{31}$ in the subdiagonal. Also, note that $c_{i_2 j_2}$ lies in Row 3 which is free, but the corresponding Column 1 is not free. Column 1 is associated with a label "1" and contains the highlighted element $c_{11}$. Also, note that Row 1 is associated with a label "3". We remove element $c_{11}$ from the subdiagonal and include element $c_{13}$ instead. The resulting tableau is written as follows.

| $\Delta_j$ | 0 | 0 | 2 | 2 | 0 | $k = 2$ | | |
|---|---|---|---|---|---|---|---|---|
| Label | – | – | – | – | – | $\Delta_{\min}$ | $\Delta_{\min R}$ | $\delta_i$ |
| – | 1 | 2 | $\boxed{3}$ | 4 | 8 | – | – | – |
| – | 1 | $\boxed{1}$ | 3 | 4 | 8 | – | – | – |
| – | $\boxed{1}$ | 5 | 5 | 6 | 8 | – | – | – |
| – | 2 | 5 | 6 | 8 | $\boxed{2}$ | – | – | – |
| $\ell = 4$. Iteration complete. | | | | | | | | |

Notice that at the end of the first iteration, $\ell = n = 4$, and conditions (i) and (ii) are satisfied. Thus, the current diagonal represents the optimal solution to the initial problem. The optimal value of the objective function is given as $3+1+1+2 = 7$.

We summarize the results discussed in this section as the following statement.

**Theorem 4.1** *An $n \times m$ rectangular assignment problem can be solved in $O(n^2 m)$ time by Algorithm LAPBL and in $O(n^3 + nm)$ time by Algorithm LAPD.*

### *4.1.2 Minimizing Total Completion Times of Unrelated Machines*

In this subsection, we illustrate how the rectangular linear assignment problem can be used for solving a classical scheduling problem of minimizing the sum of the completion times on $m$ unrelated parallel machines, i.e., problem $Rm| \,|\sum C_j$.

Each job $j$ of set $N = \{1, 2, \ldots, n\}$ has to be assigned to be processed on one of the $m \geq 2$ parallel unrelated machines. It is reasonable to assume that there are more jobs than machines, i.e., $n \geq m$. In general, if job $j$ is assigned to be processed on machine $M_i$, then such processing takes $p_{ij}$ time units, $1 \leq i \leq m$. A feasible schedule $S$ is determined by

- a partition of set $N$ into $m$ subsets $N_1, N_2, \ldots, N_m$, so that the jobs of set $N_i$ and only those are assigned to be processed on machine $M_i$, $1 \leq i \leq m$;
- the sequence of jobs $\pi^{[i]} = \left(\pi^{[i]}(1), \pi^{[i]}(2), \ldots, \pi^{[i]}(h_i)\right)$ on machine $M_i$, where $h_i = |N_i|$ and $1 \leq i \leq m$.

Take a machine $M_i$, $1 \leq i \leq m$, and suppose that in some schedule $S$, the jobs $\pi^{[i]}(1), \ldots, \pi^{[i]}(h_i)$ are processed on $M_i$ in this order. We have that

$$\begin{aligned}
C_{\pi^{[i]}(1)} &= p_{(i,\pi^{[i]}(1))}, \\
C_{\pi^{[i]}(2)} &= p_{(i,\pi^{[i]}(1))} + p_{(i,\pi^{[i]}(2))}, \\
&\cdots \\
C_{\pi^{[i]}(h_i)} &= p_{(i,\pi^{[i]}(1))} + \cdots + p_{(i,\pi^{[i]}(h_1))},
\end{aligned}$$

where $p_{(i,\pi^{[i]}(r))}$ is the processing time of a job $j = \pi^{[i]}(r)$ scheduled in position $r$ of permutation $\pi^{[i]}$ on machine $M_i$. The above relations imply that the contribution of the jobs of set $N_i$ toward the objective function is equal to

$$\sum_{r=1}^{h_i} C_{\pi^{[i]}(r)} = \sum_{r=1}^{h_i} (h_i - r + 1) p_{(i,\pi^{[i]}(r))};$$

compare this with the reasoning presented in Sect. 2.3.1.

For a schedule $S$ on unrelated machines defined by permutations $\pi^{[i]} = \left(\pi^{[i]}(1), \pi^{[i]}(2), \ldots, \pi^{[i]}(h_i)\right)$, the total completion time of all jobs can be written as follows:

$$\sum_{j=1}^{n} C_j(S) = \sum_{i=1}^{m} \sum_{r=1}^{h_i} (h_i - r + 1) p_{(i,\pi^{[i]}(r))}. \tag{4.3}$$

To minimize the objective $\sum C_j(S)$, let us define the cost function

$$c_{j,(i,r)} := (h_i - r + 1) p_{ij}, \tag{4.4}$$

which represents the contribution of a job $j = \pi^{[i]}(r)$ to the objective function (4.3). Notice that to compute the costs $c_{j,(i,r)}$, $j \in N$, $1 \leq r \leq h_i$, $1 \leq i \leq m$, we require knowledge of the number of jobs $h_i$ assigned to machine $M_i$. However, irrespective of $h_i$, if job $j$ is assigned to machine $M_i$, it will contribute exactly one of the values $p_{ij}$, $2p_{ij}$, $3p_{ij}, \dots, np_{ij}$. Thus, we may set the value $h_i = n$, $1 \leq i \leq m$ and compute all possible costs $c_{j,(i,r)}$, $1 \leq r \leq n$, $1 \leq i \leq m$, by (4.4) for every job $j \in N$.

Define a rectangular assignment problem with an $n \times k$ cost matrix $\mathbf{C} = \left(c_{j,(i,r)}\right)$ with $n$ rows, each corresponding to a job $j \in N$ and $k = nm$ columns. Number the columns by a string of the form $(i, r)$, where $i$, $1 \leq i \leq m$, refers to a machine index and $r$, $1 \leq r \leq n$, indicates a position in a permutation of jobs assigned to the machine. More precisely, the value of element $c_{j,(i,r)}$ at the intersection of the $j$th row and the $v$th column of matrix $\mathbf{C}$ for $v$, $1 \leq v \leq k$, such that $v = n(i-1) + r$, where $1 \leq i \leq m$ and $1 \leq r \leq n$, is defined by the relation (4.4).

As a result, the problem of minimizing the objective function (4.3) reduces to a rectangular assignment problem written as below:

$$\begin{array}{lll}
\text{minimize} & \displaystyle\sum_{j=1}^{n}\sum_{i=1}^{m}\sum_{r=1}^{n} c_{j,(i,r)} y_{j,(i,r)} & \\
\text{subject to} & \displaystyle\sum_{i=1}^{m}\sum_{r=1}^{n} y_{j,(i,r)} = 1, & j = 1, \dots, n; \\
& \displaystyle\sum_{j=1}^{n} y_{j,(i,r)} \leq 1, & i = 1, \dots, m,\ r = 1, \dots, n; \\
& y_{j,(i,r)} \in \{0, 1\}, & j = 1, \dots, n,\ i = 1, \dots, m, \\
& & r = 1, \dots, n.
\end{array} \tag{4.5}$$

This problem can be solved by Algorithm LAPD outlined in Sect. 4.1.1. In our case, the algorithm is applied to an $n \times k$ cost matrix and therefore requires $O(n^3 + kn)$ time, where $k = nm$. Thus, an optimal solution for problem (4.5) can be found in $O(n^3)$ time, due to the assumption that $n \geq m$. For the found solution, $y_{j,(i,r)} = 1$ implies that job $j$ is assigned to the $r$th position of machine $M_i$. The conditions of (4.5) mean that each job will be assigned to a position and no position will be used more than once. The following statement holds.

**Theorem 4.2** *Problem $Rm| |\sum C_j$ can be solved in $O\left(n^3\right)$ time, by reduction to a rectangular linear assignment problem.*

### *4.1.3 Linear Assignment Problems with a Product Matrix*

A special case of the LAP of the form (4.2) with a square cost matrix can be solved faster if $c_{ij} = a_i b_j$, $1 \leq i \leq n$, $1 \leq j \leq n$, so that the input of the problem is determined by two arrays $\mathbf{a} = (a_1, a_2, \ldots, a_n)$ and $\mathbf{b} = (b_1, b_2, \ldots, b_n)$. Such a problem is known as the linear assignment problem with a *product matrix.* This problem can be seen as another representation of the problem of minimizing a linear form

$$L(\pi) = \sum_{j=1}^{n} a_{\pi(j)} b_j \tag{4.6}$$

over a set of all permutations. Provided that

$$b_1 \ge b_2 \ge \cdots \ge b_n \tag{4.7}$$

holds, the problem reduces to finding a permutation $\varphi = (\varphi(1), \varphi(2), \ldots, \varphi(n))$ of the components of array $\mathbf{a}$, such that for any permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$, the inequality

$$\sum_{j=1}^{n} a_{\varphi(j)} b_j \le \sum_{j=1}^{n} a_{\pi(j)} b_j \tag{4.8}$$

holds. Section 2.1 of this book discusses this problem in detail and presents Theorem 2.1, which is reproduced below, with an alternative proof.

The proof is based on the so-called Monge property of a matrix. A square matrix $\mathbf{C} = \left(c_{ij}\right)_{n \times n}$ is said to be a *Monge* matrix if its elements satisfy the following property

$$c_{ij} + c_{rs} \le c_{is} + c_{rj}, \quad 1 \le i < r \le n, \; 1 \le j < s \le n. \tag{4.9}$$

It is known that the linear assignment problem with a Monge cost matrix $\mathbf{C} = \left(c_{ij}\right)_{n \times n}$ has an optimal solution given by

$$x_{ij} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{otherwise,} \end{cases}$$

so that the minimum total cost is equal to the sum of the elements of the main diagonal of $\mathbf{C}$.

**Theorem 4.3** *Provided that (4.7) holds, permutation* $\varphi = (\varphi(1), \varphi(2), \ldots, \varphi(n))$ *such that*

$$a_{\varphi(1)} \le a_{\varphi(2)} \le \cdots \le a_{\varphi(n)}$$

*satisfies (4.8), i.e., minimizes the linear form (4.6).*

*Proof* Renumber the elements of array $\mathbf{a} = (a_1, a_2, \ldots, a_n)$ in accordance with permutation $\varphi$, so that $a_1 \le a_2 \le \cdots \le a_n$. Then, matrix $\mathbf{C} = \left(c_{ij}\right)_{n \times n}$ with $c_{ij} = a_i b_j$ is a Monge matrix. Take arbitrary rows $i$ and $r$, $1 \le i < r \le n$, and columns $j$ and $s$, $1 \le j, s \le n$. Write the cost matrix $\mathbf{C} = \left(c_{ij}\right)$ as

$$\mathbf{C} = \begin{pmatrix} a_1b_1 & \cdots & a_1b_j & \cdots & a_1b_s & \cdots & a_1b_n \\ \vdots & \cdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ a_ib_1 & \cdots & \boxed{a_ib_j} & \cdots & \underline{a_ib_s} & \cdots & a_ib_n \\ \vdots & \cdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ a_rb_1 & \cdots & \underline{a_rb_j} & \cdots & \boxed{a_rb_s} & \cdots & a_rb_n \\ \vdots & \cdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ a_nb_1 & \cdots & a_nb_j & \cdots & a_nb_s & \cdots & a_nb_n \end{pmatrix},$$

where the boxed elements are the costs $c_{ij}$ and $c_{rs}$, while the underlined elements are the $c_{is}$ and $c_{rj}$. Notice that the boxed elements and the underlined elements form a rectangle, and the inequality (4.9) compares the sum of the diagonal elements of this rectangle.

Suppose that the property (4.9) does not hold for the four chosen elements, i.e.,

$$c_{ij} + c_{rs} > c_{is} + c_{rj}$$

or, equivalently,

$$a_ib_j + a_rb_s > a_ib_s + a_rb_j.$$

The latter inequality can be rewritten as

$$a_i\left(b_j - b_s\right) > a_r\left(b_j - b_s\right).$$

Since (4.7) holds, we have $a_i > a_r$, which contradicts the chosen numbering $a_1 \le a_2 \le \cdots \le a_n$. Thus, matrix $\mathbf{C}$ is a Monge matrix, and the smallest total cost is equal to $\sum_{j=1}^{n} a_jb_j$ and permutation $\varphi = (\varphi(1), \varphi(2), \ldots, \varphi(n))$ minimizes the corresponding linear form. $\square$

Finding a permutation $\varphi$ that minimizes the linear form (4.6) requires the sorting of two arrays of $n$ numbers, which can be done in $O(n \log n)$ time. This permutation matches the larger components of one of the two given arrays with smaller components of the other array. This process is formally described in Algorithm Match presented in Sect. 2.1.

See Sect. 4.5.1 for references and further discussion. In this book, we use different versions of the linear assignment problem on several occasions.

## 4.2 Knapsack and Subset-Sum Problems

In this section, we give a brief discussion of the most popular problem of Boolean programming, the linear knapsack problem, and its special case, the subset-sum problem. The main focus of the discussion is on derivation of the fully polynomial-time approximation schemes for these problems.

Let $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ be a vector with $n$ Boolean components, i.e., $x_j \in \{0, 1\}$, $1 \le j \le n$. Consider the following problem:

$$\begin{aligned} &\text{maximize} \sum_{j=1}^{n} \beta_j x_j \\ &\text{subject to} \sum_{j=1}^{n} \alpha_j x_j \le A \\ &\qquad\qquad x_j \in \{0, 1\},\ j = 1, 2, \ldots, n. \end{aligned} \tag{4.10}$$

This problem is known as the (linear) *knapsack problem*. Its popular interpretation is as follows. There are $n$ items, and an item $j$ is associated with a weight $\alpha_j$ and a profit value $\beta_j$, $1 \le j \le n$. All values of $\alpha_j$ and $\beta_j$, $1 \le j \le n$, are positive integers. The decision-maker wants to determine which of these items to place into a knapsack of the weight capacity $A$ in order to maximize the total profit of the taken items. A Boolean decision variable $x_j$ is equal to 1 if item $j$ is placed into the knapsack; otherwise, it is equal to 0, $1 \le j \le n$. Let a Boolean vector $\mathbf{x}^* = \left(x_1^*, x_2^*, \ldots, x_n^*\right)$ such that inequality $\sum_{j=1}^{n} \beta_j x_j^* \ge \sum_{j=1}^{n} \beta_j x_j$ holds for all feasible vectors $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ be called an optimal solution of the problem.

A popular version of the knapsack problem is the *subset-sum problem*, which we will also call *Problem SSP*. In this problem, it is assumed that $\beta_j = \alpha_j$, $1 \le j \le n$, so that the problem becomes

$$\begin{aligned} &\text{maximize} \sum_{j=1}^{n} \alpha_j x_j \\ &\text{subject to} \sum_{j=1}^{n} \alpha_j x_j \le A \\ &\qquad\qquad x_j \in \{0, 1\},\ j = 1, 2, \ldots, n. \end{aligned} \tag{4.11}$$

A possible interpretation of this problem can be presented in the following way. Given a set of $n$ items with an item $j$ is associated with a weight $\alpha_j$, $1 \le j \le n$, it is required to find a subset of items of the largest total weight that does not exceed the maximum weight capacity $A$. This problem is closely related to a scheduling problem of finding a non-preemptive schedule that minimizes the makespan on two parallel identical machines, i.e., problem $P2| |C_{\max}$. Indeed, if we interpret the value $\alpha_j$ as the processing time $p_j$ of job $j \in N = \{1, 2, \ldots, n\}$ and define $A := \frac{1}{2}p(N) = \frac{1}{2}\sum_{j \in N} p_j$, then a solution to the problem (4.11) will determine the corresponding optimal schedule: The jobs of set $N_1 = \left\{j \in N | x_j = 1\right\}$ are assigned to machine $M_1$, while the remaining jobs of set $N_2 = N \backslash N_1$ are to be assigned to machine $M_2$. The value of the makespan is equal to $\max\{p(N_1), p(N_2)\}$.

The decision version of problem (4.11) is related to the well-known problem PARTITION. Below, we reproduce its formulation from Sect. 1.3.2.

PARTITION: Given positive integers $e_1, \dots, e_r$ and the index set $R = \{1, \dots, r\}$ such that $e(R) = \sum_{i \in R} e_i = 2E$, is it possible to partition set $R$ into disjoint subsets $R_1$ and $R_2$ such that $e(R_1) = \sum_{i \in R_1} e_i = E$ and $e(R_2) = \sum_{i \in R_2} e_i = E$?

To see the link between PARTITION and problem (4.11), given an arbitrary instance of the former problem, define an instance of the latter problem such that $n = r$, $\alpha_j = e_j$, $1 \le j \le n$, and $A = E$. Due to this link, we deduce that Problem SSP is NP-hard in the ordinary sense. The same complexity status has the general knapsack problem (4.10).

The knapsack problem admits a solution by a dynamic programming (DP) pseudopolynomial-time algorithm. We first outline a general idea of the algorithm and then present its efficient implementation.

The DP algorithm scans the items in the order of their numbering and for next item $k \in N$ gives the variable $x_k$ a value of either 0 or 1, provided that the latter option is feasible. The algorithm generates partial solutions associated with the states of the form:

$$(k, Z_k, y_k),$$

where

$k$ is the number of the assigned variables;
$Z_k$ is the current value of the objective function; and
$y_k := \sum_{j=1}^{k} \alpha_j x_j$, the total weight of the items put into the knapsack.

**Algorithm KPDP1**

**Step 1**. Start with the initial state $(0, Z_0, y_0) = (0, 0, 0)$.
**Step 2**. For all $k$ from 1 to $n$, make transitions from each state of the form

$$(k-1, Z_{k-1}, y_{k-1}) \tag{4.12}$$

into the states of the form

$$(k, Z_k, y_k) \tag{4.13}$$

by assigning the next variable $x_k$.

**(a)** Define $x_k := 1$, provided that item $k$ fits into the knapsack, i.e., if the inequality $y_{k-1} + \alpha_k \le A$ holds. If feasible, the assignment $x_{k:} = 1$ changes a state (4.12) to a state of the form (4.13), where

$$Z_k := Z_{k-1} + \beta_k, \;\; y_k := y_{k-1} + \alpha_k. \tag{4.14}$$

**(b)** Define $x_k := 0$, which is always feasible. This assignment changes a state of the form (4.12) to a state of the form (4.13), where

$$Z_k := Z_{k-1};\ y_k := y_{k-1}. \tag{4.15}$$

**Step 3**. Find $Z_n^*$, the largest value of $Z_n$ among all found states of the form $(n, Z_n, y_n)$. Perform backtracking and find vector $\mathbf{x}^* = \left(x_1^*, x_2^*, \ldots, x_n^*\right)$ that leads to $Z_n^*$. Output $\mathbf{x}^*$ as the solution vector and $Z_n^*$ as the optimal value of the objective function.

Below, we present a more efficient implementation of the DP algorithm. It computes an array $Z = (Z(0), Z(1), \ldots, Z(A))$, where $Z(d)$ represents the largest current value of the objective function, provided that the size of the knapsack is equal to $d$, $0 \le d \le D$. The elements of the array can only be updated if the next item $k \in N$ fits the knapsack, for the values of $d$ between $\alpha_k$ and $A$, taken in the decreasing order. The value $Z(d)$ is only updated if placing an item into the knapsack increases the current value of the function.

**Algorithm KPDP2**

```
Step 1. For all integer $d$ from 0 to $A$, compute $Z(d) := 0$.
Step 2. For all $k$ from 1 to $n$, do
    For all integer $d$ from $A$ down to $\alpha_k$, do
    if $Z(d - \alpha_k) + \beta_k > Z(d)$, then compute $Z(d) := Z(d - \alpha_k) + \beta_k$.
Step 3. Perform backtracking and find vector $\mathbf{x}^* = (x_1^*, x_2^*, \ldots, x_n^*)$ that leads
    to $Z(A)$. Output $\mathbf{x}^*$ as the solution vector and $Z(A)$ as the optimal value of the
    objective function.
```

The running time of Algorithm KPDP2 is $O(nA)$, which is pseudopolynomial with respect to the length of input of the problem.

Algorithm KPDP2 computes "profits for weights," i.e., computes the values of the objective function (profits) for all possible values of the knapsack weight. It is possible to design an alternative form of the DP algorithm, which computes the knapsack weights for a range of possible values of profits. To determine such a range, we need an upper bound $U$ on the optimal value of the function.

For the knapsack problem, the "profit-weight" ratio $\alpha_j/\beta_j$ is called the *efficiency* of item $j \in N$. Notice that for Problem SSP, all items are equally efficient. For the knapsack problem of the form (4.10), assume that the items are renumbered in non-increasing order of their efficiencies and consider the *continuous relaxation* of the knapsack problem obtained from (4.10) by replacing the integrality condition $x_j \in \{0, 1\}$ by the inequality $0 \le x_j \le 1$ for all $j \in N$. The following statement holds.

**Theorem 4.4** *Let* $\mathbf{x}^* = \left(x_1^*, x_2^*, \ldots, x_n^*\right)$ *and* $\mathbf{x}^C = \left(x_1^C, x_2^C, \ldots, x_n^C\right)$ *be optimal solutions to the knapsack problem (4.10) and to its continuous relaxation, respectively. Then, there exists an index t such that*

$$x_j^C = \begin{cases} 1, & 1 \le j \le t \\ \frac{1}{\alpha_t}\left(A - \sum_{i=1}^{t-1} \alpha_i\right), & j = t \\ 0 & t+1 \le j \le n. \end{cases}$$

*Besides, the inequalities*

$$\sum_{j=1}^{n} \beta_j x_j^* \le U = \sum_{j=1}^{n} \beta_j x_j^C \le 2\sum_{j=1}^{n} \beta_j x_j^*$$

*hold.*

Thus, if the solution to the continuous relaxation is not fully integer, it contains exactly one fractional component, $0 < x_t^C < 1$.

A DP algorithm, which can be seen as an alternative to Algorithm KPDP2, computes "weights for profits". More formally, it computes an array $Y = (Y(0), Y(1), \ldots, Y(U))$, where $Y(q)$ represents the minimal smallest current weight of the knapsack, provided that the current profit value is equal to $q$, $0 \le q \le U$. The elements of array $Y$ can be updated if for an item $k \in N$, the value of $q$ is between $\beta_k$ and $U$, taken in the decreasing order. The value $Y(q)$ is only updated if placing an item into the knapsack decreases the minimal current weight of $q$.

**Algorithm KPDP3**

```
Step 1. Compute an upper bound $U$ and set $Y(0) := 0$.
Step 2. For all integer $q$ from 1 to $U$, compute $Y(q) := A + 1$.
Step 3. For all $k$ from 1 to $n$, do
    For all integer $q$ from $U$ down to $\beta_k$, do
    if $Y(q - \beta_k) + \alpha_k < Y(q)$, then compute $Y(q) := Y(q - \beta_k) + \alpha_k$.
Step 4. Find $Z^* = \max\{q | Y(q) \le A\}$. Perform backtracking and find vector $\mathbf{x}^* = (x_1^*, x_2^*, \ldots, x_n^*)$ that leads to $Z^*$. Output $\mathbf{x}^*$ as the solution vector and $Z^*$ as the optimal value of the objective function.
```

The running time of Algorithm KPDP3 is $O(nU)$, which is pseudopolynomial with respect to the length of input of the problem.

We now explain how a dynamic algorithm can be converted into a fully pseudopolynomial-time approximation scheme (FPTAS), which is the best possible approximation algorithm an NP-hard problem may admit.

We need to revise the definitions of approximation algorithms and schemes given in Sect. 1.3.4, which are presented for scheduling problems with a minimization objective. For a collection of decision variables $\mathbf{x}$, consider a problem of maximizing a function $\varphi(\mathbf{x})$, with a positive optimal value $\varphi(\mathbf{x}^*)$. A polynomial-time algorithm that finds a feasible solution $\mathbf{x}^H$ such that the inequality $\varphi(\mathbf{x}^H)/\varphi(\mathbf{x}^*) \ge \rho$ holds for all instances of the problem is called a *$\rho$-approximation* algorithm and $\rho \le 1$ *worst-case ratio bound.* A family of $\rho$-approximation algorithms is called a *fully polynomial-time approximation scheme (FPTAS)* if $\rho = 1 - \varepsilon$ for any $\varepsilon > 0$ and the running time is polynomial with respect to both the length of the problem input and $1/\varepsilon$. A special attention is paid to the design of FPTASs that require *strongly polynomial* running time, i.e., time bounded by a polynomial that depends on $n$ and $1/\varepsilon$ only.

The easiest way to demonstrate that the knapsack problem admits an FPTAS that requires strongly polynomial running time is to apply Algorithm KPDP3 to a modified instance of the original problem, in which the profit values are scaled to become $\beta'_j = \lfloor \beta_j/Q \rfloor$, $j \in N$, where $Q$ is a suitably chosen scaling factor.

**Algorithm KPFPTAS**

**Step 1**. For a given $\varepsilon > 0$, compute $Q := \frac{\varepsilon\beta_{\max}}{n}$, where $\beta_{\max} := \max\{\beta_j | j \in N\}$.

**Step 2**. Run Algorithm KPDP3 for the instance in which the profit values $\beta_j$ are replaced by $\beta'_j = \lfloor \beta_j/Q \rfloor$, $j \in N$.

**Step 3**. For the found solution vector $\mathbf{x}^\varepsilon = (x_1^\varepsilon, x_2^\varepsilon, \dots, x_n^\varepsilon)$, compute the value $Z^\varepsilon$ of the objective function with respect to the original profit values $\beta_j$, $j \in N$. Output $\mathbf{x}^\varepsilon$ as the vector of an approximate solution and $Z^\epsilon$ as an approximated value of the objective function.

It is clear that in the scaled instance used in Step 2, the inequalities $\beta'_j \le \frac{n}{\varepsilon}$, $j \in N$, hold. This implies that the value $n \max\left\{\beta'_j | j \in N\right\} \le \frac{n^2}{\varepsilon}$ can be used as an upper bound $U$ on the objective function of the scaled instance, i.e., Step 2 will require $O(nU) = O(n^3/\varepsilon)$ time, i.e., the running time of Algorithm KPFPTAS is strongly polynomial. It can be proved that it outputs a value $Z^\varepsilon \ge (1-\varepsilon)\sum_{j\in N} \beta_j x_j^*$.

There are various techniques that can be used to reduce both time and space required for implementing an FPTAS for the knapsack problem. To the best of our knowledge, the best FPTAS requires $O\left(n \min\left\{\log n, \log \frac{1}{\varepsilon}\right\} + \frac{1}{\varepsilon^2} \log \frac{1}{\varepsilon} \min\left\{n, \frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right\}\right)$ time and $O\left(n + \frac{1}{\varepsilon^2}\right)$ space, which makes it a practical algorithm for solving knapsack problem of a reasonable size.

A better FPTAS is available the subset-sum problem (see the statement below).

**Theorem 4.5** *Problem SSP of the form (4.11) admits an FPTAS that for a given positive* $\varepsilon$, *either finds an optimal solution* $x_j^* \in \{0, 1\}$, $j \in N$, *such that*

$$\sum_{j\in N} \alpha_j x_j^* < (1-\varepsilon)A$$

*or finds an approximate solution* $x_j^\varepsilon \in \{0, 1\}$, $j \in N$, *such that*

$$(1-\varepsilon)A \le \sum_{j\in N} p_j x_j^\varepsilon \le A.$$

*Such an FPTAS requires* $O\left(\min\left\{n/\varepsilon, n + \frac{1}{\varepsilon^2}\log\left(\frac{1}{\varepsilon}\right)\right\}\right)$ *time and* $O\left(n + \frac{1}{\varepsilon}\right)$ *space.*

Section 4.5.2 briefly discusses the related issues and provides necessary references.

## 4.3 Half-Product: Approximation and Relaxation

In this section, we give a brief discussion of the problem of quadratic Boolean programming, known as the half-product problem. The main focus of the discussion is on derivation of the fully polynomial-time approximation schemes for this and related problems. We also present an approach to solving its continuous relaxation in polynomial time.

### 4.3.1 *Formulation and Approximation*

Let $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ be a vector with $n$ Boolean components. Consider the function

$$H(\mathbf{x}) = \sum_{1 \le i < j \le n}^{n} \alpha_i \beta_j x_i x_j - \sum_{j=1}^{n} \gamma_j x_j, \tag{4.16}$$

where for each $j$, $1 \le j \le n$, the coefficients $\alpha_j$ and $\beta_j$ are nonnegative integers, while $\gamma_j$ is an integer that can be either negative or positive. The function $H(\mathbf{x})$ is called a *half-product* since its quadratic part consists of roughly half of the terms of the product $\left(\sum_{j=1}^{n} \alpha_j x_j\right)\left(\sum_{j=1}^{n} \beta_j x_j\right)$. Notice that we only are interested in the instances of the problem for which the optimal value of the function is strictly negative; otherwise, setting all decision variables to zero solves the problem.

We refer to the problem of minimizing function $H(\mathbf{x})$ of the form (4.16), as *Problem HP*. Let a Boolean vector $\mathbf{x}^* = \left(x_1^*, x_2^*, \ldots, x_n^*\right)$ such that inequality $H(\mathbf{x}^*) \le H(\mathbf{x})$ that holds for all Boolean vectors $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ be called an optimal solution of the problem. This problem is known to be NP-hard in the ordinary sense, even if $\alpha_j = \beta_j$. It has numerous applications, mainly to machine scheduling. Notice that in those applications, a scheduling objective function usually is written in the form

$$F(\mathbf{x}) = H(\mathbf{x}) + K, \tag{4.17}$$

where $K$ is a given additive constant. We refer to the problem of minimizing function $F(\mathbf{x})$ of the form (4.17), as *Problem HPAdd.*

For illustration, below we give an example of a scheduling problem that can be reformulated in terms of Problem HPAdd.

Consider the problem $P2| \,|\sum w_j C_j$. The jobs of set $N = \{1, 2, \ldots, n\}$ have to be assigned to be processed without preemption on one of the machines $M_1$ or $M_2$, and each machine processes at most one job at a time. The processing of job $j \in N$ on any machine takes $p_j$ time units. There is a positive weight $w_j$ associated with job $j$, which indicates its relative importance. All values $p_j$ and $w_j$ are positive integers. The goal is to minimize the weighted sum of the completion times.

Assume that the jobs are numbered in such a way that

$$\frac{p_1}{w_1} \le \frac{p_2}{w_2} \le \cdots \le \frac{p_n}{w_n}. \tag{4.18}$$

Recall from Sect. 2.2 that the sequence of jobs numbered in accordance with (4.18) is called a WSPT weighted shortest processing time sequence. In an optimal schedule for the classical single machine problem of minimizing the sum of the weighted completion times $\sum_{j \in N} w_j C_j$, the jobs are processed according to the WSPT sequence. Thus, for problem $P2| |\sum w_j C_j$, there exists an optimal schedule in which the jobs are processed on each machine in the order of their numbering.

Introduce Boolean decision variables and define

$$x_j := \begin{cases} 1, & \text{if job } j \text{ is scheduled on machine } M_1 \\ 0, & \text{otherwise} \end{cases}.$$

Then, for a schedule in which the jobs are considered in the order of a chosen numbering, a job $j$ assigned to machine $M_1$ completes at time

$$C_j = \sum_{i=1}^{j} p_i x_i;$$

otherwise, its completion time is

$$C_j = \sum_{i=1}^{j} p_i (1 - x_i).$$

For problem $P2| |\sum w_j C_j$, we derive

$$\sum_{j=1}^{n} w_j C_j = \sum_{j=1}^{n} w_j x_j \sum_{i=1}^{j} p_i x_i + \sum_{j=1}^{n} w_j \left(1 - x_j\right) \sum_{i=1}^{j} p_i (1 - x_i),$$

which due to $x_j^2 = x_j$, $j \in N$, can be rewritten as

$$\begin{aligned} \sum_{j=1}^{n} w_j C_j &= \sum_{j=1}^{n} w_j x_j \sum_{i=1}^{j-1} p_i x_i + \sum_{j=1}^{n} w_j \left(1 - x_j\right) \sum_{i=1}^{j-1} p_i (1 - x_i) \\ &\quad + \sum_{j=1}^{n} p_j w_j x_j + \sum_{j=1}^{n} p_j w_j \left(1 - x_j\right) \\ &= \sum_{j=1}^{n} w_j x_j \sum_{i=1}^{j-1} p_i x_i + \sum_{j=1}^{n} w_j \left(1 - x_j\right) \sum_{i=1}^{j-1} p_i (1 - x_i) + \sum_{j=1}^{n} p_j w_j. \end{aligned} \tag{4.19}$$

Since

$$\sum_{1\le i<j\le n} p_i w_j (1-x_i)(1-x_j) = \sum_{1\le i<j\le n} p_i w_j x_i x_j + \sum_{1\le i<j\le n} p_i w_j$$
$$-\sum_{j=1}^{n}\left(w_j\left(\sum_{i=1}^{j-1} p_i\right)+p_j\left(\sum_{i=j+1}^{n} w_i\right)\right)x_j,$$

we deduce that

$$\sum_{j=1}^{n} w_j C_j = 2\sum_{1\le i<j\le n} p_i w_j x_i x_j + \sum_{1\le i<j\le n} p_i w_j + \sum_{j=1}^{n} p_j w_j.$$
$$-\sum_{j=1}^{n}\left(w_j\left(\sum_{i=1}^{j-1} p_i\right)+p_j\left(\sum_{i=j+1}^{n} w_i\right)\right)x_j$$
$$= 2\sum_{1\le i<j\le n} p_i w_j x_i x_j + \sum_{1\le i\le j\le n} p_i w_j$$
$$-\sum_{j=1}^{n}\left(w_j\left(\sum_{i=1}^{j-1} p_i\right)+p_j\left(\sum_{i=j+1}^{n} w_i\right)\right)x_j.$$

Thus, problem $P2|\,|\sum w_j C_j$ reduces to Problem HPAdd with

$$\alpha_j = 2p_j,\ \beta_j = w_j,\ \gamma_j = w_j\left(\sum_{i=1}^{j-1} p_i\right)+p_j\left(\sum_{i=j+1}^{n} w_i\right),\ j\in N;\ K=\sum_{1\le i\le j\le n} p_i w_j.$$

Similarly to the problems related to the linear knapsack problem discussed in Sect. 4.2, a major direction of research on the half-product problem and its variants is aimed at designing fully polynomial-time approximation scheme (FPTAS), especially those of strongly polynomial time.

We need to refine the definition of an FPTAS for a problem of minimizing a function $\varphi(\mathbf{x})$ which takes both positive and negative values, which happens to Problem HP. For such a problem, an FPTAS delivers a feasible solution $\mathbf{x}^H$ such that $\varphi(\mathbf{x}^H)-\varphi(\mathbf{x}^*)\le\varepsilon|\varphi(\mathbf{x}^*)|$.

Below, we present an FPTAS for Problem HP of minimizing function $H(\mathbf{x})$, without an additive constant. We start with a pseudopolynomial DP algorithm and then explain how it can be converted into an approximation scheme.

The DP algorithm scans the items in the order of their numbering and for next item $k\in N$ gives the variable $x_k$ either a value of 0 or value of 1; here, both options are feasible. The algorithm generates partial solutions associated with states of the form $(k, Z_k, y_k)$, which have the same meaning as in the case of Algorithm KPDP1 from Sect. 4.2, except $y_k := \sum_{j=1}^{k} \alpha_j x_j$ is not interpreted the total weight of the items put into the knapsack. Since the optimal value of the function is negative, an assignment $x_k = 1$ is made only if it decreases the current value of the function.

**Algorithm HPDP**

**Step 1**. Define the initial state $(0, Z_0, y_0) := (0, 0, 0)$ and store that state.

**Step 2**. For all $k$ from 1 to $n$, do

**(a)** Define $x_k := 0$ and change a stored state of the form (4.12) to a state of the form (4.13) by setting

$$Z_k := Z_{k-1}, \ y_k := y_{k-1}. \tag{4.20}$$

**(b)** If $\beta_k y_{k-1} - \gamma_k < 0$, then define $x_k := 1$ and change a state of the form (4.12) to a state of the form (4.13) by setting

$$Z_k := Z_{k-1} + \beta_k y_{k-1} - \gamma_k; \ y_k := y_{k-1} + \alpha_k. \tag{4.21}$$

**(c)** For all generated states of the form (4.13) with the same value $Z_k$, store only one, with the smallest $y_k$.

**Step 3**. Find $Z_n^*$, the largest value of $Z_n$ among all found states of the form $(n, Z_n, y_n)$. Perform backtracking and find vector $\mathbf{x}^* = \left(x_1^*, x_2^*, \dots, x_n^*\right)$ that leads to $Z_n^*$. Output $\mathbf{x}^*$ as the solution vector and $Z_n^*$ as the optimal value of the objective function.

Algorithm HPDP can be implemented in $O\left(n \sum_{j \in N} \alpha_j\right)$ time. Its correctness follows from the fact that for two states $(k, Z_k, y_k)$ and $(k, Z_k', y_k')$ generated in iteration $k$, we can keep only the former state, provided that $Z_k \le Z_k'$ and $y_k \le y_k'$.

To convert Algorithm HPDP into an FPTAS for Problem HP, a popular technique of thinning the solution space can be used. For a given $\varepsilon > 0$, we want to make sure that the number of states kept after each iteration is $O(n/\varepsilon)$. For an iteration $k$, $1 \le k \le n$, compute $LB_k$, the smallest objective function value among all states $(k, Z_k, y_k)$ generated after Step 2(b) of Algorithm HPDP. Recall that $LB_k$, as well as all other function values computed by the algorithm, is negative. Thus, since $LB_k \ge H(\mathbf{x}^*)$, we deduce that $|LB_k| \le |H(\mathbf{x}^*)|$. For a given $\varepsilon > 0$, define $\Delta_k := (\varepsilon |LB_k|)/n = -\varepsilon LB_k/n$. It follows that for each $k$, $0 \le k \le n-1$, the inequality $\Delta_k \le \varepsilon |H(\mathbf{x}^*)|/n$ holds.

Now, we replace Step 2(c) by another storage mechanism:

**(i)** Divide the interval $[LB_k, 0]$ into subintervals of length $\Delta_k$.

**(ii)** From all states $(k, Z_k, y_k)$ of the form (4.13) generated after Step 2(b) with $Z_k$ in the same subinterval, retain the one with the smallest $y_k$.

Algorithm HPDP with Step 2(c) replaced by the actions (i) and (ii) described above will be referred to as *Algorithm HPFPTAS*. Let vector $\mathbf{x}^\varepsilon$ be found by this algorithm. It can be proved that $H(\mathbf{x}^\varepsilon) - H(\mathbf{x}^*) \le \varepsilon |H(\mathbf{x}^*)|$.

Notice that the number of subintervals created in each iteration is $O(n/\varepsilon)$. Since for each subinterval, at most one state is kept with the function value in that subinterval, the total number of states kept in each iteration is $O(n/\varepsilon)$.

Iteration $k$ starts with $O(n/\varepsilon)$ states of the form (4.12) kept, and each state will make transitions to at most two new states. For each of these states $(k, Z_k, y_k)$ of the form (4.13), it takes constant time to identify a subinterval $I$ of length $\Delta_k$ that contains the value $Z_k$. If there is no kept state associated with subinterval $I$, the state $(k, Z_k, y_k)$ becomes a kept state; otherwise, state $(k, Z_k, y_k)$ is compared with the kept state associated with $I$ and the one with the smaller value $y_k$ is kept. This means that for each iteration, the storage mechanism outlined above can be implemented in $O(n/\varepsilon)$ time.

Thus, Algorithm HPFPTAS behaves as an FPTAS, and the following statement holds.

**Theorem 4.6** *Problem HP admits an FPTAS that requires $O\left(n^2/\varepsilon\right)$ time.*

Notice that this running time is the best possible, at least regarding the number $n$ of items, since it takes $O\left(n^2\right)$ time to compute the value of function $H(\mathbf{x})$ for a given Boolean vector $\mathbf{x}$.

Algorithms that behave as an FPTAS for Problem HP do not necessarily deliver an $\varepsilon$-approximate solution for Problem HPAdd of minimizing function (4.17). This is due to the presence of an additive constant of the sign that is opposite to the sign of the variable part of the function.

To illustrate this, consider a function of the form (4.17). If vector $\mathbf{x}^*$ minimizes function $H(\mathbf{x})$ of the form (4.16), it will obviously minimize function $F(\mathbf{x})$ as well. Suppose that for minimizing function $H(\mathbf{x})$, an FPTAS is available that delivers a solution $\mathbf{x}^\varepsilon$, such that $H(\mathbf{x}^\varepsilon) - H(\mathbf{x}^*) \le \varepsilon|H(\mathbf{x}^*)|$.

For $\mathbf{x}^\varepsilon$ to be accepted as an $\varepsilon$-approximate solution for minimizing function $F(\mathbf{x})$, we must establish the inequality

$$F(\mathbf{x}^H) \le (1+\varepsilon)F(\mathbf{x}^*). \tag{4.22}$$

For a solution $\mathbf{x}^\varepsilon$ found by an FPTAS for minimizing $H(\mathbf{x})$, we will have

$$F(\mathbf{x}^\varepsilon) = H(\mathbf{x}^\varepsilon) + K \le H(\mathbf{x}^*) + \varepsilon\left|H(\mathbf{x}^*)\right| + K = F(\mathbf{x}^*) + \varepsilon\left|H(\mathbf{x}^*)\right|.$$

Since $H(\mathbf{x}^*) < 0$, we have $F(\mathbf{x}^\varepsilon) \le F(\mathbf{x}^*) - \varepsilon H(\mathbf{x}^*) \le (1-\varepsilon)F(\mathbf{x}^*) + \varepsilon K$. If $K > 0$, there is no evidence that (4.22) will hold, and further analysis must be performed.

There are several ways of achieving an FPTAS for Problem HPAdd. It is especially challenging to develop an FPTAS of the best possible running time $O\left(n^2/\varepsilon\right)$.

Several of these approaches are mentioned below.

**Theorem 4.7** *Let $\mathbf{x}^\varepsilon$ be a solution found by an FPTAS for Problem HP. If $|H(\mathbf{x}^*)/F(\mathbf{x}^*)| \le \rho$ for some positive $\rho > 0$, then $F(\mathbf{x}^\varepsilon) - F(\mathbf{x}^*) \le \varepsilon\rho F(\mathbf{x}^*)$.*

*Proof* It follows from $H(\mathbf{x}^{\varepsilon}) - H(\mathbf{x}^*) \leq \varepsilon|H(\mathbf{x}^*)|$ that $F(\mathbf{x}^{\varepsilon}) - F(\mathbf{x}^*) = (H(\mathbf{x}^{\varepsilon}) + K) - (H(\mathbf{x}^*) + K) \leq \varepsilon|H(\mathbf{x}^*)|$. Since by the theorem's condition $|H(\mathbf{x}^*)| \leq \rho F(\mathbf{x}^*)$, we deduce that the theorem holds. □

Another approach is applicable if the ratio of an upper bound and a lower bound on the optimal value of function (4.17) is bounded. For Problem HPAdd, suppose that we know an upper bound $UB$ on the optimal value $F(\mathbf{x}^*)$. The following DP algorithm is a modification of Algorithm HPDP. It also manipulates the states of the form $(k, Z_k, y_k)$, but now $Z_k$ is the current value of the objective function (4.17). The states with a value of $Z_k$ greater than $UB$ are not kept, since the corresponding partial solutions cannot be extended to an optimal solution.

**Algorithm HPAddDP**

**Step 1**. Start with the initial state $(0, Z_0, y_0) := (0, K, 0)$ and store that state.
**Step 2**. For all $k$ from 1 to $n$, do

- **(a)** Define $x_k := 0$ and change a state (4.12) to a state of the form (4.13) by setting (4.20).
- **(b)** If $\beta_k y_{k-1} - \gamma_k < 0$, then define $x_k := 1$ and change a state of the form (4.12) to a state of the form (4.13), by setting (4.21).
- **(c)** For a given upper bound $UB$, remove the states $(k, Z_k, y_k)$ with $Z_k > UB$.
- **(d)** For generated all states of the form (4.13) with the same value $Z_k$ keep only one, with the smallest $y_k$.

**Step 3**. Find $Z_n^*$, the largest value of $Z_n$ among all found states of the form $(n, Z_n, y_n)$. Perform backtracking and find vector $\mathbf{x}^* = \left(x_1^*, x_2^*, \ldots, x_n^*\right)$ that leads to $Z_n^*$. Output $\mathbf{x}^*$ as the solution vector and $Z_n^*$ as the optimal value of the objective function.

Algorithm HPAddDP can be implemented in $O(nUB)$ time. As for Algorithm HPDP, the correctness of Algorithm HPAddDP follows from the fact that for two states $(k, Z_k, y_k)$ and $(k, Z'_k, y'_k)$ generated in iteration $k$, we can keep only the former state, provided that $Z_k \leq Z'_k$ and $y_k \leq y'_k$.

To convert Algorithm HPAddDP into an FPTAS for Problem HPAdd, a technique similar to that used above for developing Algorithm HPFPTAS can be employed. Let $LB$ be a lower bound on the optimal value $F(\mathbf{x}^*)$ such that $UB/LB \leq \rho$ for some $\rho \geq 1$. For an iteration $k$, $1 \leq k \leq n$, compute $UB_k$, the largest objective function value among all states $(k, Z_k, y_k)$ generated after Step 2(c) of Algorithm HPAddDP. Define $\Delta := (\varepsilon LB)/n$. We need to replace Step 2(d) by another storage mechanism:

- **(i)** Divide the interval $[0, UB_k]$ into subintervals of length $\Delta$.
- **(ii)** From all states $(k, Z_k, y_k)$ of the form (4.13) generated after Step 2(c) with $Z_k$ in the same subinterval, retain the one with the smallest $y_k$.

Algorithm HPAddDP with Step 2(d) replaced by the actions (i) and (ii) described above will be referred to as *Algorithm HPRhoFPTAS*.

**Theorem 4.8** *For Problem HPAdd of minimizing function (4.17), denote the lower and upper bounds on the value of $F(\mathbf{x}^*)$ by $LB$ and $UB$, respectively, i.e., $LB \le F(\mathbf{x}^*) \le UB$. If the ratio $UB/LB$ is bounded from above by some $\rho$, then Algorithm HPRhoFPTAS delivers a solution $\mathbf{x}^0$ such that $F(\mathbf{x}^0) - LB \le \varepsilon LB$ in $O(\rho n^2/\varepsilon)$ time.*

If the value of $\rho$ is bounded from above by a polynomial of the length of the input of the problem, then Algorithm HPRhoFPTAS behaves as an FPTAS. Moreover, if $\rho$ is a constant, then such an FPTAS requires the best possible running time of $O\big(n^2/\varepsilon\big)$.

### 4.3.2 Convex Half-Product and Its Continuous Relaxation

Using the fact that for Boolean variables $x_j^2 = x_j$, $j \in N$, we can rewrite the half-product function as

$$H(\mathbf{x}) = \sum_{1\le i<j\le n}^{n} \alpha_i\beta_j x_i x_j - \sum_{j=1}^{n}\gamma_j x_j = \sum_{i=1}^{n}\beta_j x_j \sum_{i=1}^{j}\alpha_i x_i - \sum_{j=1}^{n}\big(\gamma_j + \alpha_j\beta_j\big)x_j. \tag{4.23}$$

The quadratic term of the above expression can be rewritten in the matrix form as

$$\sum_{j=1}^{n}\beta_j x_j \sum_{i=1}^{j} a_i x_i - \frac{1}{2}\sum_{j=1}^{n}\alpha_j\beta_j x_j = \frac{1}{2}\mathbf{x}^T\mathbf{G}\mathbf{x},$$

where

$$\mathbf{G} = \begin{bmatrix} \alpha_1\beta_1 & \alpha_1\beta_2 & \cdots & \alpha_1\beta_n \\ \alpha_1\beta_2 & \alpha_2\beta_2 & \cdots & \alpha_2\beta_n \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_1\beta_n & \alpha_2\beta_n & \cdots & \alpha_n\beta_n \end{bmatrix}. \tag{4.24}$$

For illustration of the matrix representation of the half-product, consider the following example.

*Example 4.2* Take the half-product function for $n = 3$ such that

$$\alpha_1 = 2,\ \alpha_2 = 3,\ \alpha_3 = 4;\ \beta_1 = 3,\ \beta_2 = 4,\ \beta_4 = 5.$$

Then, for a Boolean vector $\mathbf{x} = (x_1, x_2, x_3)$, we have that

$$\sum_{j=1}^{n}\beta_j x_j \sum_{i=1}^{j} a_i x_i = 3x_1 \times 2x_1 + 4x_2(2x_1 + 3x_2) + 5x_3(2x_1 + 3x_2 + 4x_3)$$
$$= 6x_1^2 + 8x_1x_2 + 10x_1x_3 + 12x_2^2 + 15x_2x_3 + 20x_3^2$$

and

$$\sum_{j=1}^{n}\alpha_j\beta_j x_j = \sum_{j=1}^{n}\alpha_j\beta_j x_j^2 = 6x_1^2 + 12x_2^2 + 20x_3^3,$$

so that

$$\sum_{j=1}^{n}\beta_j x_j \sum_{i=1}^{j} a_i x_i - \frac{1}{2}\sum_{j=1}^{n}\alpha_j\beta_j x_j = 3x_1^2 + 8x_1x_2 + 10x_1x_3 + 6x_2^2 + 15x_2x_3 + 10x_3^2.$$

On the other hand,

$$\frac{1}{2}\begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix}\begin{pmatrix} 2\times 3 & 2\times 4 & 2\times 5 \\ 2\times 4 & 3\times 4 & 3\times 5 \\ 2\times 5 & 3\times 5 & 4\times 5 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$
$$= 3x_1^2 + 8x_1x_2 + 10x_1x_3 + 6x_2^2 + 15x_2x_3 + 10x_3^2.$$

The following statement holds.

**Theorem 4.9** *Matrix* **G** *of the form (4.24) is positive semidefinite and function $H(\mathbf{x})$ is convex, provided that the items are numbered so that*

$$\frac{\alpha_1}{\beta_1} \le \frac{\alpha_2}{\beta_2} \le \cdots \le \frac{\alpha_n}{\beta_n}. \tag{4.25}$$

Many known scheduling applications reduce to minimizing function (4.17), either without any additional constraints, i.e., to Problem HPAdd, or with an additional knapsack constraint

$$\sum_{j=1}^{n}\alpha_j x_j \le A. \tag{4.26}$$

We can view the value $\alpha_j$ as the weight of item $j$, $1 \le j \le n$, i.e., $x_j = 1$ means that item $j$ is placed into a knapsack with capacity $A$, while $x_j = 0$ means that the corresponding item is not placed into the knapsack. An important feature is that the coefficients $\alpha_j$ in the knapsack constraint are the same as in the quadratic terms of the objective function.

Below, we present an approach to solving a *continuous relaxation* of Problem HPAdd which works even if the knapsack constraint is imposed, provided that the objective function is convex. The continuous relaxation is obtained from the original

Boolean formulation by relaxing the integrality constraints and replacing the condition $x_j \in \{0, 1\}$ by $0 \le x_j \le 1$, $j = 1, 2, \dots, n$.

Introduce new decision variables $\chi_j = \alpha_j x_j$, $j = 1, 2, \dots, n$, and using the representation (4.23), rewrite the continuous relaxation of Problem HPAdd with the knapsack constraint as

$$\begin{aligned} &\text{minimize } Z = \sum_{j=1}^{n} c_j \chi_j \sum_{i=1}^{j} \chi_j - \textstyle\sum_{j=1}^{n} \gamma'_j \chi_j + K' \\ &\text{subject to } \textstyle\sum_{j=1}^{n} \chi_j \le A \\ &\qquad\qquad 0 \le \chi_j \le \alpha_j, \ j = 1, 2, \dots, n; \end{aligned} \tag{4.27}$$

where $c_j = \beta_j/\alpha_j$, $\gamma'_j = \left(\gamma_j + \alpha_j \beta_j\right)/\alpha_j$, $j \in N$. We can reformulate the objective function in an almost separable form

$$\sum_{j=1}^{n} c_i \chi_i \sum_{i=1}^{j} \chi_j = \frac{1}{2} \sum_{j=1}^{n} c_j \chi_j^2 + \frac{1}{2} \sum_{j=1}^{n-1} (c_j - c_{j+1}) \left( \sum_{i=1}^{j} \chi_i \right)^2 + \frac{1}{2} c_n \left( \sum_{j=1}^{n} \chi_j \right)^2 . \tag{4.28}$$

Due to the representation (4.28), the problem (4.27) can be reduced to finding a flow of the minimum cost in a special network. Let $G = (V, A)$ be a digraph called *network*, where $V$ is a set of vertices that contains a single *source* and a single *sink* and $A$ is the set of arcs. No arc enters the source, and no arc leaves the sink. A flow $f$ is a function that associates each arc of a network with a real number. Each arc has a capacity, which is an upper bound on a feasible value of flow on that arc. The value of the flow in a network is equal to the sum of flows on the arcs that leave the source (or, equivalently, enter the sink). For all other vertices, the conservation law must hold: The total flow that enters a vertex is equal to the total flow that leaves that vertex. Additionally, each arc is associated with a cost of flow on that arc. In the case that is most studied in the literature, the cost of flow $f$ on an arc $e$ is linear and is equal to $c_e f$. The problem of minimizing the sum of these costs is a classical problem of network optimization known as the *min-cost flow problem*. Notice that if the capacities of the arcs are integer, the min-cost flow is also integer.

Introduce the network $G$ with the set $V$ of vertices and set $E$ of arcs. Set $V$ consists of a single source $v_s$, a single sink $v_t$, the vertices $w_n, w_{n-1}, \dots, w_2$, and the vertices $t_n, t_{n-1}, \dots, t_1$. Set $E$ consists of the following arcs: $(v_s, w_n)$ of capacity $A$, $(w_j, t_j)$ of capacity $\alpha_j$, and $(w_j, w_{j-1})$ of capacities $\sum_{i=1}^{j-1} \alpha_i$ for $j = n, n-1, \dots, 3$; $(w_2, t_2)$ and $(w_2, t_1)$ of capacities $\alpha_2$ and $\alpha_1$, respectively; besides, for each $j$, $1 \le j \le n$, vertex $t_j$ is connected to the sink by the arc $\left(t_j, v_t\right)$ of capacity $\alpha_j$. Let $f$ be a flow on an arc, then the cost of that flow is defined as $\frac{1}{2} c_n f^2$ for arc $(v_s, w_n)$; as $\frac{1}{2}\left(c_{j-1} - c_j\right) f^2$ for each arc $(w_j, w_{j-1})$ where $j = n, n-1, \dots, 3$; as $\frac{1}{2} c_j f^2 - \gamma'_j f$ for the arc that enters vertex $t_j$, $j = 2, 3, \dots, n$; and as $(c_1 - \frac{1}{2} c_2) f^2 - \gamma'_1 f$ for arc $(w_2, t_1)$, while the cost of the flow on each arc that enters the sink is zero. It can be verified that the minimum cost of the flow in the constructed network corresponds

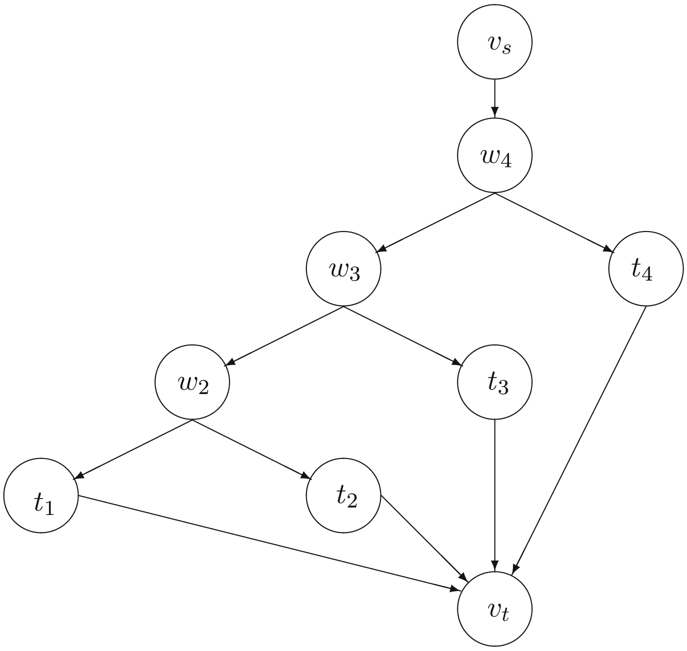


**Fig. 4.1** A rooted tree for $n = 4$ with the leaves connected to the sink $v_t$ for Example 4.3

to the minimum value of $Z - K$, while the flow on the arc that enters vertex $t_j$ is equal to the corresponding value of the decision variable $\chi_j$.

*Example 4.3* For illustration, consider the example below for $n = 4$. The network is shown in Fig. 4.1, while its parameters are given in Table 4.1.

Thus, the problem (4.27) reduces to finding the flow that minimizes a quadratic convex cost function. The latter problem admits a polynomial-time algorithm, as stated below.

**Theorem 4.10** *Let $G = (V, E)$ be a network that is a series-parallel digraph with an additional single source and a single sink. Then, the problem of finding a flow of total value $q$ that minimizes a convex quadratic cost function can be solved in $O(|V||E| + |E|\log|E|)$ time, the flow values being piecewise linear functions of $q$.*

Consult Sect. 3.1.2 for the definition of a series-parallel digraph. In our case, network $G$ is a tree with a single source and a single sink, so that $|V| = O(n)$ and $|E| = O(n)$. The value $q$ of the total flow should be set equal either to $\sum_{j\in N}\alpha_j$ if no knapsack constraint is imposed or to $A$ if the constraint (4.26) is added.

We conclude that the following statement holds.

**Theorem 4.11** *For a convex objective function, the continuous relaxation of Problem HPAdd, without and with the linear knapsack constraints, can be solved in $O(n^2)$ time.*

**Table 4.1** Parameters of the network for Example 4.3

| Arc | Capacity | Cost for flow $f$ |
|---|---|---|
| $(v_s, w_4)$ | $A$ | $\frac{1}{2}c_4 f^2$ |
| $(w_4, w_3)$ | $\alpha_1 + \alpha_2 + \alpha_3$ | $\frac{1}{2}(c_3 - c_4) f^2$ |
| $(w_4, t_4)$ | $\alpha_4$ | $\frac{1}{2}c_4 f^2 - \gamma_4' f$ |
| $(w_3, w_2)$ | $\alpha_1 + \alpha_2$ | $\frac{1}{2}(c_2 - c_3) f^2$ |
| $(w_3, t_3)$ | $\alpha_3$ | $\frac{1}{2}c_3 f^2 - \gamma_3' f$ |
| $(w_2, t_2)$ | $\alpha_2$ | $\frac{1}{2}c_2 f^2 - \gamma_2' f$ |
| $(w_2, t_1)$ | $\alpha_1$ | $(c_1 - \frac{1}{2}c_2) f^2 - \gamma_1' f$ |
| $(t_1, v_t)$ | $\alpha_1$ | 0 |
| $(t_2, v_t)$ | $\alpha_2$ | 0 |
| $(t_3, v_t)$ | $\alpha_3$ | 0 |
| $(t_4, v_t)$ | $\alpha_4$ | 0 |

Theorem 4.11 can be used for finding a lower bound on the optimal objective function values for problems related to minimizing the half-product.

## 4.4 Symmetric Quadratic Functions

In this section, we address the issues of design of the approximation schemes for minimizing special forms of the half-product, without additional constraints and with a linear knapsack constraint.

Consider the function

$$P(\mathbf{x}) = \sum_{1 \le i < j \le n}^{n} \alpha_i \beta_j x_i x_j + \sum_{j=1}^{n} \mu_j x_j + \sum_{j=1}^{n} \nu_j \left(1 - x_j\right) + K, \tag{4.29}$$

where all coefficients $\alpha_j$,$\beta_j$, $\mu_j$, $\nu_j$, and $K$ are nonnegative integers. The problem of minimizing the function $P(\mathbf{x})$ of the form (4.29) is called the *positive half-product problem* or *Problem PosHP*. The knapsack-constrained variant of Problem PosHP can be written as

$$\begin{aligned} \text{minimize } & P(\mathbf{x}) = \sum_{1 \le i < j \le n}^{n} \alpha_i \beta_j x_i x_j + \sum_{j=1}^{n} \mu_j x_j + \sum_{j=1}^{n} \nu_j \left(1 - x_j\right) + K \\ \text{subject to } & \sum_{j=1}^{n} \alpha_j x_j \le A \\ & x_j \in \{0, 1\}, \ \ j = 1, 2, \ldots, n, \end{aligned} \tag{4.30}$$

which we call the *positive half-product knapsack problem* and denote by *Problem PosHPK*.

Consider also another quadratic function

$$Z(\mathbf{x}) = \sum_{1 \le i < j \le n} \alpha_i \beta_j x_i x_j + \sum_{1 \le i < j \le n} \alpha_i \beta_j (1 - x_i)(1 - x_j) + \sum_{j=1}^{n} \mu_j x_j + \sum_{j=1}^{n} \nu_j (1 - x_j) + K, \tag{4.31}$$

where all coefficients are non-negative. We call function $Z(\mathbf{x})$ of the form (4.31) a *symmetric quadratic function*. The problem of minimizing function $Z(\mathbf{x})$ without additional constraints is called *Problem SQ*, and its version with a knapsack constraint (4.26) is called the *symmetric quadratic knapsack problem*, or *Problem SQK*. The term "*symmetric*" is used because both the quadratic and the linear parts of the objective function are separated into two terms: one depending on the variables $x_j$, and the other depending on the variables $(1 - x_j)$.

Function $Z(\mathbf{x})$ of the form (4.31) is a generalization of function $P(\mathbf{x})$ of the form (4.29), since the former function contains an additional quadratic term. The objective function $Z(\mathbf{x})$ of the form (4.31) can be rewritten as

$$\begin{aligned} Z(\mathbf{x}) = \sum_{i=1}^{n} \left( \mu_i - \alpha_i \sum_{j=i+1}^{n} \beta_j - \beta_i \left( \sum_{j=1}^{i-1} \alpha_j \right) - \nu_i \right) x_i \\ 2 \sum_{1 \le i < j \le n} \alpha_i \beta_j x_i x_j + K + \sum_{i=1}^{n} \left( \alpha_i \sum_{j=i+1}^{n} \beta_j + \nu_i \right), \end{aligned} \tag{4.32}$$

i.e., in a form that used in the formulation of Problem HPAdd, i.e., both functions (4.29) and (4.31) are special forms of the general half-product functions (4.17).

Table 4.2 summarizes the notation introduced above for all quadratic Boolean programming problems considered in this and the previous sections.

The interest in functions (4.29) and (4.31) is motivated by the fact that for all known scheduling problems that admit a reformulation as Problem HPAdd (with or without the knapsack constraints), the objective function can in fact be rewritten as

**Table 4.2** Notation for Boolean programming problems under consideration

| Notation | Objective/formulation | Additional constraints |
|---|---|---|
| HP | $H(\mathbf{x})$ (4.16) | None |
| HPAdd | $F(\mathbf{x})$ (4.17) | None |
| PosHP | $P(\mathbf{x})$ (4.29) | None |
| PosHPK | $P(\mathbf{x})$ (4.30) | (4.26) |
| SQ | $Z(\mathbf{x})$ (4.31) | None |
| SQK | $Z(\mathbf{x})$ (4.31) | (4.26) |

either a positive half-product or a symmetric quadratic function. Specific features of these functions allow us to design fast FPTASs for their minimization.

**Theorem 4.12** *Each Problem PosHP and Problem PosHPK admits an FPTAS that requires $O\left(n^2/\varepsilon\right)$ time, provided that the objective function is convex.*

As one of the ingredients of these schemes, finding a solution to the continuous relaxation is required, as described in Sect. 4.3.

Problem SQK also admits an FPTAS under special conditions, which hold for all its applications.

**Theorem 4.13** *Problem SQK of minimizing function (4.31) subject to a linear knapsack constraint (4.26) admits an FPTAS that requires $O\left(n^4/\varepsilon^2\right)$ time, provided that either the problem admits a constant ratio approximation algorithm that requires at most $O\left(n^4\right)$ time, or the objective function is convex and $\nu_j \geq \alpha_j\beta_j$, $j \in N$.*

See Sect. 4.5.3 for a discussion and references.

In the remainder of this section, we derive an FPTAS for minimizing function $Z(\mathbf{x})$ of the form (4.31), provided that the function is convex. The resulting FPTAS takes the best possible running time of $O\left(n^2/\varepsilon\right)$ and can be applied to various scheduling problems, including problem $P2|\,|\sum w_jC_j$ discussed in Sect. 4.3.1.

First, notice that due to (4.32) function $Z(\mathbf{x})$ is a form of the half-product, so that we can rely on the results known for Problem HPAdd. The main ingredient for developing the required FPTAS is Theorem 4.8. To be able to apply it, we need to demonstrate that function $Z(\mathbf{x})$ admits an upper bound $UB$ and a lower bound $LB$ on its optimal value such that $UB/LB \leq \rho$, where $\rho$ is a constant.

If function $Z(\mathbf{x})$ is convex, then Theorem 4.11 holds and the continuous relaxation can be solved in $O\left(n^2\right)$ time.

Let $\mathbf{x}^C = (x_1^C, \dots, x_n^C)$, $0 \leq x_j^C \leq 1$, be the corresponding solution vector of the continuous relaxation of the problem of minimizing a convex function $Z(\mathbf{x})$. Clearly, $Z(\mathbf{x}^C) \leq Z(\mathbf{x}^*)$, and we may set $LB = Z(\mathbf{x}^C)$.

To obtain an upper bound $UB$, we perform an appropriate rounding of the fractional components of vector $\mathbf{x}^C$. A very simple rounding algorithm is described below.

**Algorithm SQRound**

**Step 1**. Given a vector $\mathbf{x}^C = (x_1^C, \dots, x_n^C)$, $0 \leq x_j^C \leq 1$, a solution to the continuous relaxation of the problem of minimizing function (4.31), determine the sets $I_1 = \left\{j \in N,\ x_j^C \leq \frac{1}{2}\right\}$ and $I_2 = \left\{j \in N,\ x_j^C > \frac{1}{2}\right\}$ and find vector $\mathbf{x}^H = (x_1^H, \dots, x_n^H)$ with components

$$x_j^H = \begin{cases} 0 \text{ if } j \in I_1 \\ 1 \text{ if } j \in I_2 \end{cases}.$$

**Step 2**. Output vector $\mathbf{x}^H = (x_1^H, \dots, x_n^H)$ as heuristic solution for the problem of minimizing function (4.31).

The running time of Algorithm SQRound is $O(n)$. Clearly, the inequalities $Z(\mathbf{x}^C) \le Z(\mathbf{x}^*) \le Z(\mathbf{x}^H)$ hold, i.e., we may take $Z(\mathbf{x}^H)$ as an upper bound $UB$ on the optimal value $F_\lambda(\mathbf{x}^*)$. We now estimate the ratio $UB/LB = Z(\mathbf{x}^H)/Z(\mathbf{x}^C)$.

**Theorem 4.14** *Let $\mathbf{x}^C$ be an optimal solution to the continuous relaxation of the problem of minimizing function $Z(\mathbf{x})$ of the form (4.31) and $\mathbf{x}^H$ be a vector found by Algorithm SQRound. Then,*

$$\rho = \frac{Z(\mathbf{x}^H)}{Z(\mathbf{x}^C)} \le 4.$$

*Proof* For a vector $\mathbf{x}^C$, let $I_1$ and $I_2$ be the index sets found in Step 2 of Algorithm SQRound. For a vector $\mathbf{x} = (x_1, \dots, x_n)$, where $0 \le x_j \le 1$, define

$$Z_1(\mathbf{x}) := \sum_{\substack{1\le i<j\le n\\ i,j\in I_1}} \alpha_i\beta_j x_i x_j + \sum_{\substack{1\le i<j\le n\\ i,j\in I_1}} \alpha_i\beta_j (1-x_i)\big(1-x_j\big);$$

$$Z_2(\mathbf{x}) := \sum_{\substack{1\le i<j\le n\\ i\in I_1, j\in I_2}} \alpha_i\beta_j x_i x_j + \sum_{\substack{1\le i<j\le n\\ i\in I_1, j\in I_2}} \alpha_i\beta_j (1-x_i)\big(1-x_j\big);$$

$$Z_3(\mathbf{x}) := \sum_{\substack{1\le i<j\le n\\ i\in I_2, j\in I_1}} \alpha_i\beta_j x_i x_j + \sum_{\substack{1\le i<j\le n\\ i\in I_2, j\in I_1}} \alpha_i\beta_j (1-x_i)\big(1-x_j\big);$$

$$Z_4(\mathbf{x}) := \sum_{\substack{1\le i<j\le n\\ i,j\in I_2}} \alpha_i\beta_j x_i x_j + \sum_{\substack{1\le i<j\le n\\ i,j\in I_2}} \alpha_i\beta_j (1-x_i)\big(1-x_j\big);$$

$$Z_5(\mathbf{x}) := \sum_{j\in I_1} \mu_j x_j + \sum_{j\in I_1} \nu_j\big(1-x_j\big);$$

$$Z_6(\mathbf{x}) := \sum_{j\in I_2} \mu_j x_j + \sum_{j\in I_2} \nu_j\big(1-x_j\big).$$

By the rounding conditions in Step 2 of Algorithm SQRound, we derive

$$Z_2(\mathbf{x}^H) = Z_3(\mathbf{x}^H) = 0,$$

while

$$Z_1(\mathbf{x}^H) = \sum_{\substack{1\le i<j\le n\\ i,j\in I_1}} \alpha_i\beta_j;\ Z_1(\mathbf{x}^C) \ge \frac{1}{4}\sum_{\substack{1\le i<j\le n\\ i,j\in I_1}} \alpha_i\beta_j;$$
$$Z_4(\mathbf{x}^H) = \sum_{\substack{1\le i<j\le n\\ i,j\in I_2}} \alpha_i\beta_j;\ Z_4(x^C) \ge \frac{1}{4}\sum_{\substack{1\le i<j\le n\\ i,j\in I_2}} \alpha_i\beta_j;$$
$$Z_5(\mathbf{x}^H) = \sum_{j\in I_1}\nu_j; \qquad Z_5(\mathbf{x}^C) \ge \frac{1}{2}\sum_{j\in I_1}\nu_j;$$
$$Z_6(\mathbf{x}^H) = \sum_{j\in I_2}\mu_j; \qquad Z_6(\mathbf{x}^C) \ge \frac{1}{2}\sum_{j\in I_2}\mu_j;$$

Thus, we have that

$$\begin{aligned} Z(\mathbf{x}^H) &= \sum_{k=1}^{6} Z_k(\mathbf{x}^H) + K = Z_1(\mathbf{x}^H) + Z_4(\mathbf{x}^H) + Z_5(\mathbf{x}^H) + Z_6(\mathbf{x}^H) + K \\ &\le 4Z_1(\mathbf{x}^C) + 4Z_4(\mathbf{x}^C) + 2Z_5(\mathbf{x}^C) + 2Z_6(\mathbf{x}^C) + K \\ &\le 4\sum_{k=1}^{6} Z_k(\mathbf{x}^C) + 4K = 4Z(\mathbf{x}^C), \end{aligned}$$

as required. □

It follows immediately from Theorem 4.14 that for the problem of minimizing a convex function (4.31), Theorem 4.8 is applied with $\rho = 4$. Hence, we obtain the following statement.

**Theorem 4.15** *Problem SQ of minimizing a convex function (4.31) admits an FPTAS that requires $O\left(n^2/\varepsilon\right)$ time.*

For illustration, apply Theorem 4.15 to problem $P2||\sum w_j C_j$. First, notice that the objective function in problem $P2||\sum w_j C_j$ given by (4.19) is a special case of function (4.31) with

$$\alpha_j = p_j,\ \beta_j = w_j,\ \mu_j = 0,\ \nu_j = 0,\ j\in N;\ K = \sum_{j=1}^{n} p_j w_j.$$

Moreover, function (4.19) is convex, which follows from the WSPT numbering (4.18) (see Theorem 4.9). Thus, a direct application of Theorem 4.15 yields the following result.

**Theorem 4.16** *Problem $P2||\sum w_j C_j$ of minimizing the sum of the weighted completion times on two parallel identical machines admits an FPTAS that requires $O\left(n^2/\varepsilon\right)$ time.*

## 4.5 Bibliographic Notes

In this section, we give a brief review of the relevant literature sources that address the Boolean programming problems discussed in this chapter.

### *4.5.1 Assignment Problem*

A comprehensive exposition of various aspects of solving the linear assignment problem is given in the monograph Burkard et al. (2009).

One of the most famous results in the area is the Hungarian algorithm by Kuhn (1955) that solves the linear assignment problem of the form (4.2) with a square cost matrix. Munkres (1957) proves that the Hungarian algorithm requires $O\left(n^3\right)$ time.

Algorithm LAPBL in Sect. 4.1.1 for solving a rectangular linear assignment problem of the form (4.1) is due to Bourgeois and Lassale (1971); this explains the use of “BL” in the name of the algorithm.

Algorithm LAPD is given by Dinic (1976); this explains “D” in the name of the algorithm. Algorithm LAPD was first published in Russian (for the maximization problem) and to the best of our knowledge has not been earlier reproduced in English. The method is based on the traditional ideas of Kuhn (1955) and Dinic and Kronrod (1969), and its running time of $O\left(n^3 + nm\right)$ is the fastest known for the rectangular assignment problem.

Horn (1973) and Bruno et al. (1974) reduce problem $Rm| |\sum C_j$ to a Boolean programming problem similar to (4.5). The running time of $O\left(n^3\right)$ stated in Theorem 4.2 can be derived without the use of Algorithm LAPD.

See Burkard et al. (1996) for applications of Monge matrices in combinatorial optimization, including the property used in the proof of Theorem 4.3.

### *4.5.2 Linear Knapsack Problem*

Various aspects of solving the linear knapsack problem, its variants, and extensions are studied in detail in the monographs by Martello and Toth (1990) and by Kellerer et al. (2004).

Algorithm KPDP1 implements a classical Bellman’s recursion given by Bellman (1957). In our description of Algorithms KPDP2 and KPDP3, we follow Kellerer et al. (2004). The basic FPTAS similar to that presented in Algorithm KPFPTAS can be found in Kellerer et al. (2004) and Vazirani (2003).

Historically, the first FPTAS for the knapsack problem and for the subset-sum problem is due to Ibarra and Kim (1975). The best-known FPTAS for the knapsack problem is developed by Kellerer and Pferschy (1999, 2004) (see also Kellerer et al. (2004)). Theorem 4.5 on the best known FPTAS for Problem SSP is proved in Kellerer et al. (2003).

### *4.5.3 Half-Product Problem and Its Variants*

A detailed reviews on Boolean programming problems related to optimizing the half-product and related functions are given in Kellerer and Strusevich (2012, 2016).

Problems of minimizing quadratic functions similar to (4.16) were introduced in 1990s as mathematical models for various scheduling problems by Kubiak (1995) and Jurisch et al. (1997). Function (4.16) and the term "half-product" were introduced by Badics and Boros (1998), who considered the problem of minimizing that function $H(\mathbf{x})$ with respect to Boolean decision variables with no additional constraints. Badics and Boros (1998) also prove that minimizing $H(\mathbf{x})$ is NP-hard in the ordinary sense even if $\alpha_j = \beta_j$ for all $j = 1, 2, \dots, n$. By contrast, maximizing function $H(\mathbf{x})$ of the form (4.16) with respect to Boolean decision variables with no additional constraints requires $O(n^3)$ time, as demonstrated by Kellerer et al. (2015b).

The half-product function is a special case of the general quadratic function of Boolean variables. Let $(q_{ij})_{n\times n}$ be a symmetric quadratic matrix. For a Boolean vector $\mathbf{x} = (x_1, x_2, \dots, x_n)$, define the function

$$Q(\mathbf{x}) = \sum_{1\le i<j\le n} q_{ij}x_i x_j - \sum_{j=1}^{n} \gamma_j x_j.$$

The problem of optimizing function $Q(\mathbf{x})$ subject to a knapsack constraint is known as the *quadratic knapsack problem*. In general, this problem is NP-hard in the strong sense. See Chap. 12 of the book by Kellerer et al. (2004) and a survey by Pisinger (2007) for an overview of principal results on this problem. Badics and Boros (1998) give an $O(n^4)$-time algorithm that recognizes whether a quadratic function $Q(\mathbf{x})$ of $n$ Boolean variables is a half-product.

Algorithm HPFPTAS based on converting Algorithm HPDP is due to Erel and Ghosh (2008). This algorithm is the first FPTAS for the problem that requires strongly polynomial time. It is proved in Sarto Basso and Strusevich (2016) that Algorithm HPFPTAS can be modified to become an FPTAS for handling the problem of minimizing function $H(\mathbf{x})$ subject to a knapsack constraint (4.26).

Theorem 4.7 is proved by Kubiak (2005), while Theorem 4.8 is due to Erel and Ghosh (2008). These theorems are used for deriving FPTASs for several scheduling problems that can be reformulated as Problem HPAdd (see Kellerer and Strusevich (2012, 2016)).

An approach to solving the continuous relaxation of the problem of minimizing a convex half-product function that is finalized in Theorem 4.11 is developed by Kellerer and Strusevich (2010b). Theorem 4.9 is proved by Skutella (2001).

The representation (4.28) is provided by Shioura (2009). Theorem 4.10 and the corresponding algorithm for finding the min-cost flow with a convex quadratic function is due to Tamir (1993).

Problem PosHP of the form (4.29) and its knapsack-constrained version Problem PosHPK of the form (4.30) have been introduced by Janiak et al. (2005) and by Kellerer and Strusevich (2013), respectively. Theorem 4.12 that demonstrates that both problems admit the fastest possible FPTAS with the running time of $O\left(n^2/\varepsilon\right)$ is proved in Kellerer and Strusevich (2013).

The studies on Problem SQK of the form (4.31) have been initiated by Kellerer and Strusevich (2010a, b) who prove Theorem 4.13. Notice that the assumptions in Theorem 4.13 hold for all known scheduling applications of Problem SQK (see, e.g., Sect. 13.4.2). For Problem SQK without any additional assumptions regarding the structure of the objective, Xu (2012) gives an FPTAS that requires $O\left(n^4 \log\log n + n^4/\varepsilon^2\right)$ time.

Theorems 4.14 and 4.15 which lead to the best possible FPTAS for Problem SQ with a convex objective function are proved in Kellerer et al. (2015a).

For problem $P2| |\sum w_j C_j$, Theorem 4.15 presents an alternative approach to an FPTAS that requires $O\left(n^2/\varepsilon\right)$ time. There are at least two schemes of the same running time (see Sahni (1976) and Erel and Ghosh (2008)), where the latter paper uses a reformulation of the problem in terms of minimizing a half-product.

## References

Badics T, Boros E (1998) Minimization of half-products. Math Oper Res 33:649–660

Bellman R (1957) Dynamic programming. Princeton University Press, Princeton

Bruno JL, Coffman EG Jr, Sethi R (1974) Scheduling independent tasks to reduce mean finishing time. Comm ACM 17:382–387

Bourgeois F, Lassale JC (1971) An extension of the Munkres algorithm for the assignment problem to rectangular matrices. Comm ACM 14:802–804

Burkard RE, Klinz B, Rudolf R (1996) Perspectives of Monge properties in optimization. Discr Appl Math 70:95–161

Burkard R, Dell'Amico M, Martello S (2009) Assignment problems. SIAM, Philadelphia

Dinic EA (1976) On solving two assignment problems. In: Fridman AA (ed) Studies in discrete optimization (Issledovaniya po diskretnoj optimizatsii). Nauka, Moscow, pp 333–348 (in Russian)

Dinic EA, Kronrod MA (1969) An algorithm for the solution of the assignment problem. Soviet Math Doklady 10(6):1324–1326

Erel E, Ghosh JB (2008) FPTAS for half-products minimization with scheduling applications. Discr Appl Math 156:3046–3056

Horn WA (1973) Minimizing average flow time with parallel machines. Oper Res 21:846–847

Ibarra OH, Kim CE (1975) Fast approximation algorithms for the knpasack and sum of subset problem. J ACM 22:463–468

Janiak A, Kovalyov MY, Kubiak W, Werner F (2005) Positive half-products and scheduling with controllable processing times. Eur J Oper Res 165:416–422

Jurisch B, Kubiak W, J ózefowska J, (1997) Algorithms for minclique scheduling problems. Discr Appl Math 72:115–139

Kellerer H, Pferschy U (1999) A new fully polynomial time approxiamtion scheme for the knapsack problem. J Comb Opti 3:59–71

Kellerer H, Pferschy U (2004) Improved dynamic programming in connection with an FPTAS for the knapsack problem. J Comb Opti 8:5–11

Kellerer H, Strusevich VA (2010a) Fully polynomial approximation schemes for a symmetric quadratic knapsack problem and its scheduling applications. Algorithmica 57:769–795

Kellerer H, Strusevich VA (2010b) Minimizing total weighted earliness-tardiness on a single machine around a small common due date: An FPTAS using quadratic knapsack. Int J Found Comp Sci 21:357–383

Kellerer H, Strusevich VA (2012) The symmetric quadratic knapsack problem: Approximation and scheduling applications. 4OR - Quart J. Oper Res 10:111–161

Kellerer H, Strusevich VA (2013) Fast approximation schemes for Boolean programming and scheduling problems related to positive convex half-product. Eur J Oper Res 228:24–32

Kellerer H, Strusevich VA (2016) Optimizing the Half-Product and related quadratic Boolean functions: Approximation and scheduling applications. Ann Oper Res 240:39–94

Kellerer H, Mansini R, Pferschy U, Speranza MG (2003) An efficient fully polynomial approximation scheme for the Subset-Sum Problem. J Comp Syst Sci 66:349–370

Kellerer H, Pferschy U, Pisinger D (2004) Knapsack problems. Springer, Berlin

Kellerer H, Rustogi K, Strusevich VA (2015a) A fast FPTAS for single machine scheduling problem of minimizing total weighted earliness and tardiness about a large due date. University of Greenwich, London, Report SORG-10-2015

Kellerer H, Sarto Basso R, Strusevich VA (2015b) Approximability issues for unconstrained and constrained maximization of half-product related functions. University of Greenwich, London, Report SORG-01-2015

Kubiak W (1995) New results on the completion time variance minimization. Discr Appl Math 58:157–168

Kubiak W (2005) Minimization of ordered, symmetric half-products. Discr Appl Math 146:287–300

Kuhn HW (1955) The Hungarian method for the assignment problem. Naval Res Log Quart 2:83–97

Martello S, Toth P (1990) Knapsack problems. Algorithms and computer implementation. Chichester, Wiley

Munkres J (1957) Algorithms for the assignment and transportation problems. J SIAM 5:32–38

Pisinger D (2007) The quadratic knapsack problem - a survey. Discr Appl Math 155:623–648

Sahni SK (1976) Algorithms for scheduling independent tasks. J ACM 23:116–127

Sarto Basso R, Strusevich VA (2016) Differential approximation schemes for half-product related functions and their scheduling applications. Discr Appl Math (In Press), doi:10.1016/j.dam.2015.10.006

Shioura A (2009) Private communication

Skutella M (2001) Convex quadratic and semidefinite programming relaxations in scheduling. J ACM 48:206–242

Tamir A (1993) A strongly polynomial algorithm for minimum convex separable quadratic cost flow problems on two-terminal series-parallel networks. Math Progr 59:117–132

Vazirani V (2003) Approximation algorithms. Springer, Berlin

Xu Z (2012) A strongly polynomial FPTAS for the symmetric quadratic knapsack problem. Eur J Oper Res 218:377–381

# Chapter 5
# Convex Sequences and Combinatorial Counting

In this chapter, we establish certain properties that are used on several occasions in this book. The main results that we present here primarily revolve around the convex and $V$-shaped finite sequences and the inequalities that govern them. We prove an inequality that involves an arbitrary non-decreasing function that depends on ceiling functions, thereby establishing the convexity and $V$-shapeness of the corresponding sequence. This sequence often appears in scheduling problems, especially when the jobs of a given set are to be divided into a known number of groups. The $V$-shapeness of this sequence enables us to speed up the running times of several algorithms that we consider in this book (see, e.g., Chaps. 16 and 17).

Apart from the results related to convex and $V$-shaped sequences, we also present a discussion on combinatorial counting. Again, such concepts are required when the jobs of a given set are to be divided into a known number of groups.

## 5.1 Introduction to Convex and $V$-Shaped Sequences

A sequence $A(k)$, $1 \le k \le n$, is called *convex* if

$$A(k) \le \frac{1}{2}(A(k-1) + A(k+1)),\ 2 \le k \le n-1, \tag{5.1}$$

i.e., any element is no larger than the arithmetic mean of its neighbors. For a convex sequence, the rate $\nabla(k) = A(k+1) - A(k)$ does not decrease as $k$ grows. A concept of a convex sequence is closely related to the notion of a *log-convex* sequence, for which

$$A(k) \le \sqrt{A(k-1)A(k+1)},\ 2 \le k \le n-1,$$

i.e., any element is no larger than the geometric mean of its neighbors. Any log-convex sequence is also convex due to the inequality between the arithmetic and geometric means.

V.A. Strusevich and K. Rustogi, *Scheduling with Times-Changing Effects and Rate-Modifying Activities*, International Series in Operations Research & Management Science 243, DOI 10.1007/978-3-319-39574-6_5

A sequence $C(k)$, $1 \le k \le n$, is called *$V$-shaped* if there exists a $k_0$, $1 \le k_0 \le n$, such that

$$C(1) \ge \cdots \ge C(k_0 - 1) \ge C(k_0) \le C(k_0 + 1) \le \cdots \le C(n).$$

If a finite sequence that contains $n$ terms is $V$-shaped, then its minimum value and the position $k_0$ of that minimum in the sequence can be found by binary search in at most $\lceil \log_2 n \rceil$ comparisons.

There is also a somewhat dual concept of the *$\Lambda$-shaped* sequence, also known as unimodal sequence or pyramidal sequence, in which the elements are placed in non-decreasing order and then in non-increasing order.

Below, we give an elementary proof that a convex sequence is in fact $V$-shaped.

**Lemma 5.1** *A convex sequence $C(k)$, $1 \le k \le n$, is $V$-shaped.*

*Proof* Suppose that a convex sequence $C(k)$, $1 \le k \le n$, is not $V$-shaped, i.e., there exists a $k_1$, $1 < k_1 < n$, such that

$$C(k_1 - 1) \le C(k_1) > C(k_1 + 1).$$

Combining the inequalities

$$\begin{aligned} C(k_1 - 1) &\le C(k_1), \\ C(k_1) &> C(k_1 + 1), \end{aligned}$$

we obtain

$$C(k_1) > \frac{1}{2}(C(k_1 - 1) + C(k_1 + 1)).$$

The latter inequality contradicts (5.1). □

The statement opposite to Lemma 5.1 is not true: Indeed, any monotone sequence is $V$-shaped, but need not be convex.

It can be immediately verified that the sum of two convex sequences is convex and, therefore, is $V$-shaped. On the other hand, the sum of two $V$-shaped sequences is not necessarily $V$-shaped, which, e.g., is demonstrated by the following counterexample. For an integer $n$, $0 \le n \le 9$, both sequences $\sqrt{|n-1|}$ and $\sqrt{|n-9|}$ are $V$-shaped, but their sum is not, since the sum has two equal maxima at $n = 0$ and $n = 5$, as well as two equal minima at $n = 1$ and $n = 9$.

As an illustration, consider the following situation that we formulate as an inventory control problem. Suppose that the sequence $A(k)$, $1 \le k \le n$, represents the annual holding cost of some product, provided that $k$ orders are placed during a year. Let $T$ be the cost of placing one order, so that $B(k) = Tk$ is the total ordering cost for $k$ orders. The purpose is to determine the optimal number of orders that minimizes the total cost $C(k) = A(k) + B(k)$. Notice that the sequence $B(k)$, $1 \le k \le n$, is convex and the sequence $A(k)$, $1 \le k \le n$, is non-increasing (the more frequently we order, the smaller the holding cost will be). Now, if the sequence $A(k)$, $1 \le k \le n$, is convex, then $C(k)$, $1 \le k \le n$, is also convex and therefore $V$-shaped by Lemma 5.1. Thus, the optimal number of orders can be found by binary search by computing at most $\lceil \log_2 n \rceil$ elements of sequence $C(k)$, $1 \le k \le n$. On the other hand, if the sequence $A(k)$, $1 \le k \le n$, is not convex, then we cannot guarantee that sequence $C(k)$, $1 \le k \le n$, is $V-$shaped. To see this, consider the case that $T = 20$ and $n = 5$, while the sequence $A(k)$, $1 \le k \le n$, is as below.

$$A(1) = 100,\ A(2) = 70,\ A(3) = 60,\ A(4) = 25,\ A(5) = 20.$$

The sequence $A(k)$, $1 \le k \le n$, is not convex, and it is easy to verify that the resulting sequence $C(k)$, $1 \le k \le n$, is not $V$-shaped.

## 5.2 Convexity of a Sequence Involving Sums of Functions of Ceilings

Recall that for a real number $x$, the *ceiling* $\lceil x \rceil$ is equal to the smallest integer that is no less than $x$, while the *floor* $\lfloor x \rfloor$ is equal to the largest integer that does not exceed $x$.

The main focus of this chapter is on a specially structured sequence of the form

$$P(k) = \sum_{j=1}^{n} p_j g\left(\left\lceil \frac{j}{k} \right\rceil\right),\ 1 \le k \le n, \tag{5.2}$$

where $p_j$, $1 \le j \le n$, is a sequence of non-negative numbers and $g$ is an arbitrary non-negative non-decreasing function.

We start our consideration with a simpler sequence

$$G(k) = \sum_{j=1}^{n} g\left(\left\lceil \frac{j}{k} \right\rceil\right),\ 1 \le k \le n. \tag{5.3}$$

The sequence $G(k)$, $1 \le k \le n$, is a special case of the sequence $P(k)$, $1 \le k \le n$, defined by (5.2); however, it is of interest in its own right. Consider, e.g., the following interpretation of the sequence $G(k)$, $1 \le k \le n$. Suppose that the customers are to be assigned, in order, to one of $k$ service centers. Provided that the assignment is done to keep the centers uniformly loaded, a customer $j$ will get the position $\left\lceil \frac{j}{k} \right\rceil$ at one of the centers. Thus, the value of function $g\left(\left\lceil \frac{j}{k} \right\rceil\right)$ can be understood as dissatisfaction of customer $j$ with its position at the center. In this case, $G(k)$ represents the service cost measured as the total dissatisfaction of all customers, provided that $k$ centers are open. Let $T$ be the cost of running a center, so that $T(k) = kT$ is the total cost of running $k$ centers. The purpose is to determine the number of centers to be open so that the total cost $C(k) = G(k) + T(k)$, $1 \le k \le n$, is minimized. The sequence $T(k)$, $1 \le k \le n$, is convex, and for the sequence $C(k)$, $1 \le k \le n$, to be $V$-shaped, it is sufficient to show that the sequence $G(k)$, $1 \le k \le n$, is convex.

Below, we give an elementary proof of the convexity of the sequence (5.3).

**Theorem 5.1** *The sequence $G(k)$, $1 \le k \le n$, of the form (5.3) is convex.*

*Proof* In accordance with (5.1), we need to prove that

$$2G(k) - G(k-1) - G(k+1) \le 0,\ 2 \le k \le n-1, \tag{5.4}$$

Given a value of $k$, $2 \le k \le n-1$, for a $j$, $1 \le j \le n$, we can express $j$ as $ak + b$, where $a$ is an integer in $\left\{0, 1, \dots, \left\lfloor \frac{j}{k} \right\rfloor\right\}$ and $b \le k$. We can write

$$\begin{aligned} G(k) &= \sum_{a=0}^{\lfloor \frac{n}{k} \rfloor - 1} \sum_{b=1}^{k} g\left(\left\lceil \frac{ak+b}{k} \right\rceil\right) + \sum_{b=1}^{n-k\lfloor \frac{n}{k} \rfloor} g\left(\left\lceil \frac{\lfloor \frac{n}{k} \rfloor k + b}{k} \right\rceil\right) \\ &= \sum_{a=0}^{\lfloor \frac{n}{k} \rfloor - 1} \sum_{b=1}^{k} g\left(a + \left\lceil \frac{b}{k} \right\rceil\right) + \sum_{b=1}^{n-k\lfloor \frac{n}{k} \rfloor} g\left(\left\lfloor \frac{n}{k} \right\rfloor + \left\lceil \frac{b}{k} \right\rceil\right). \end{aligned}$$

Since $b \le k$, it follows that $\left\lceil \frac{b}{k} \right\rceil = 1$, so that

$$\begin{aligned} G(k) &= \sum_{a=0}^{\lfloor \frac{n}{k} \rfloor - 1} \sum_{b=1}^{k} g(a+1) + \sum_{b=1}^{n-k\lfloor \frac{n}{k} \rfloor} g\left(\left\lfloor \frac{n}{k} \right\rfloor + 1\right) \\ &= k \sum_{r=1}^{\lfloor \frac{n}{k} \rfloor} g(r) + \left(n - k\left\lfloor \frac{n}{k} \right\rfloor\right) g\left(\left\lfloor \frac{n}{k} \right\rfloor + 1\right), \end{aligned}$$

where in the second line, $r = a + 1$ is a new summation index. Similarly, we deduce

$$\begin{aligned} G(k+1) &= (k+1) \sum_{r=1}^{\lfloor \frac{n}{k+1} \rfloor} g(r) + \left(n - (k+1)\left\lfloor \frac{n}{k+1} \right\rfloor\right) g\left(\left\lfloor \frac{n}{k+1} \right\rfloor + 1\right); \\ G(k-1) &= (k-1) \sum_{r=1}^{\lfloor \frac{n}{k-1} \rfloor} g(r) + \left(n - (k-1)\left\lfloor \frac{n}{k-1} \right\rfloor\right) g\left(\left\lfloor \frac{n}{k-1} \right\rfloor + 1\right). \end{aligned}$$

To prove (5.4), we rewrite the difference $2G(k) - G(k+1) - G(k-1)$ as

$$\left(k\sum_{r=1}^{\lfloor \frac{n}{k}\rfloor} g(r) - (k+1)\sum_{r=1}^{\lfloor \frac{n}{k+1}\rfloor} g(r)\right) + \left(k\sum_{r=1}^{\lfloor \frac{n}{k}\rfloor} g(r) - (k-1)\sum_{r=1}^{\lfloor \frac{n}{k-1}\rfloor} g(r)\right)$$
$$+2\left(n - k\left\lfloor \frac{n}{k}\right\rfloor\right) g\left(\left\lfloor \frac{n}{k}\right\rfloor + 1\right) - \left(n - (k+1)\left\lfloor \frac{n}{k+1}\right\rfloor\right) g\left(\left\lfloor \frac{n}{k+1}\right\rfloor + 1\right)$$
$$-\left(n - (k-1)\left\lfloor \frac{n}{k-1}\right\rfloor\right) g\left(\left\lfloor \frac{n}{k-1}\right\rfloor + 1\right)$$

and show that it is non-positive.

Notice that the expression

$$k\sum_{r=1}^{\lfloor \frac{n}{k}\rfloor} g(r) - (k+1)\sum_{r=1}^{\lfloor \frac{n}{k+1}\rfloor} g(r)$$

can be simplified by canceling the values of the function $g$ computed for the same values of $r$. Since $\lfloor \frac{n}{k}\rfloor \geq \lfloor \frac{n}{k+1}\rfloor$, we obtain

$$k\sum_{r=1}^{\lfloor \frac{n}{k}\rfloor} g(r) - (k+1)\sum_{r=1}^{\lfloor \frac{n}{k+1}\rfloor} g(r) = k\sum_{r=\lfloor \frac{n}{k+1}\rfloor+1}^{\lfloor \frac{n}{k}\rfloor} g(r) - \sum_{r=1}^{\lfloor \frac{n}{k+1}\rfloor} g(r).$$

Similarly, since $\lfloor \frac{n}{k-1}\rfloor \geq \lfloor \frac{n}{k}\rfloor$, we obtain

$$k\sum_{r=1}^{\lfloor \frac{n}{k}\rfloor} g(r) - (k-1)\sum_{r=1}^{\lfloor \frac{n}{k-1}\rfloor} g(r) = \sum_{r=1}^{\lfloor \frac{n}{k-1}\rfloor} g(r) - k\sum_{r=\lfloor \frac{n}{k}\rfloor+1}^{\lfloor \frac{n}{k-1}\rfloor} g(r).$$

Combining the last two equalities displayed above, we deduce that

$$\left(k\sum_{r=1}^{\lfloor \frac{n}{k}\rfloor} g(r) - (k+1)\sum_{r=1}^{\lfloor \frac{n}{k+1}\rfloor} g(r)\right) + \left(k\sum_{r=1}^{\lfloor \frac{n}{k}\rfloor} g(r) - (k-1)\sum_{r=1}^{\lfloor \frac{n}{k-1}\rfloor} g(r)\right)$$
$$= k\left(\sum_{r=\lfloor \frac{n}{k+1}\rfloor+1}^{\lfloor \frac{n}{k}\rfloor} g(r) - \sum_{r=\lfloor \frac{n}{k}\rfloor+1}^{\lfloor \frac{n}{k-1}\rfloor} g(r)\right) + \sum_{r=\lfloor \frac{n}{k+1}\rfloor+1}^{\lfloor \frac{n}{k-1}\rfloor} g(r),$$

which reduces to

$$\left(k\sum_{r=1}^{\lfloor\frac{n}{k}\rfloor}g(r)-(k+1)\sum_{r=1}^{\lfloor\frac{n}{k+1}\rfloor}g(r)\right)+\left(k\sum_{r=1}^{\lfloor\frac{n}{k}\rfloor}g(r)-(k-1)\sum_{r=1}^{\lfloor\frac{n}{k-1}\rfloor}g(r)\right)$$
$$=(k+1)\sum_{r=\lfloor\frac{n}{k+1}\rfloor+1}^{\lfloor\frac{n}{k}\rfloor}g(r)-(k-1)\sum_{r=\lfloor\frac{n}{k}\rfloor+1}^{\lfloor\frac{n}{k-1}\rfloor}g(r).$$

Coming back to the initial difference in (5.4), we have that

$$\begin{aligned}&2G(k)-G(k+1)-G(k-1)\\&=(k+1)\sum_{r=\lfloor\frac{n}{k+1}\rfloor+1}^{\lfloor\frac{n}{k}\rfloor}g(r)-(k-1)\sum_{r=\lfloor\frac{n}{k}\rfloor+1}^{\lfloor\frac{n}{k-1}\rfloor}g(r)\\&+2\Big(n-k\Big\lfloor\frac{n}{k}\Big\rfloor\Big)g\Big(\Big\lfloor\frac{n}{k}\Big\rfloor+1\Big)\\&-\Big(n-(k+1)\Big\lfloor\frac{n}{k+1}\Big\rfloor\Big)g\Big(\Big\lfloor\frac{n}{k+1}\Big\rfloor+1\Big)\end{aligned}$$

Recombining, we deduce

$$\begin{aligned}&2G(k)-G(k+1)-G(k-1)\\&=(k+1)\sum_{r=\left\lfloor\frac{n}{k+1}\right\rfloor+2}^{\lfloor\frac{n}{k}\rfloor}g(r)+\Big((k+1)-\Big(n-(k+1)\Big\lfloor\frac{n}{k+1}\Big\rfloor\Big)\Big)g\Big(\Big\lfloor\frac{n}{k+1}\Big\rfloor+1\Big)\\&-(k-1)\sum_{r=\lfloor\frac{n}{k}\rfloor+2}^{\left\lfloor\frac{n}{k-1}\right\rfloor}g(r)+\Big(2n-2k\Big\lfloor\frac{n}{k}\Big\rfloor-(k-1)\Big)g\Big(\Big\lfloor\frac{n}{k}\Big\rfloor+1\Big)\\&-\Big(n-(k-1)\Big\lfloor\frac{n}{k-1}\Big\rfloor\Big)g\Big(\Big\lfloor\frac{n}{k-1}\Big\rfloor+1\Big).\end{aligned}$$

To prove (5.4), we collect together the terms that make a positive contribution $X$ and a negative contribution $Y$ so that $2G(k)-G(k+1)-G(k-1)=X-Y$, and show that $X\le Y$. It follows that

$$\begin{aligned}X=&\Big((k+1)\Big(\Big\lfloor\frac{n}{k+1}\Big\rfloor+1\Big)-n\Big)g\Big(\Big\lfloor\frac{n}{k+1}\Big\rfloor+1\Big)\\&+(k+1)\sum_{r=\lfloor\frac{n}{k+1}\rfloor+2}^{\lfloor\frac{n}{k}\rfloor}g(r)+(2n+1)g\Big(\Big\lfloor\frac{n}{k}\Big\rfloor+1\Big);\end{aligned}$$

$$Y = \left(2k\left\lfloor \frac{n}{k} \right\rfloor + k\right) g\left(\left\lfloor \frac{n}{k} \right\rfloor + 1\right) + (k-1) \sum_{r=\lfloor \frac{n}{k} \rfloor + 2}^{\lfloor \frac{n}{k-1} \rfloor} g(r) +$$
$$+\left(n - (k-1)\left\lfloor \frac{n}{k-1} \right\rfloor\right) g\left(\left\lfloor \frac{n}{k-1} \right\rfloor + 1\right).$$

In the expression for $X$, the range of arguments of the function $g$ is from $\lfloor \frac{n}{k+1} \rfloor + 1$ to $\lfloor \frac{n}{k} \rfloor + 1$, while in the expression for $Y$, the range of arguments of the function $g$ is from $\lfloor \frac{n}{k} \rfloor + 1$ to $\lfloor \frac{n}{k-1} \rfloor + 1$, so that we can write

$$X = \sum_{r=\lfloor \frac{n}{k+1} \rfloor + 1}^{\lfloor \frac{n}{k} \rfloor + 1} x_r g(r), \;\; Y = \sum_{r=\lfloor \frac{n}{k} \rfloor + 1}^{\lfloor \frac{n}{k-1} \rfloor + 1} y_r g(r),$$

where $x_r$ and $y_r$ are suitably chosen non-negative coefficients.

Computing

$$\sum_{r=\lfloor \frac{n}{k+1} \rfloor + 1}^{\lfloor \frac{n}{k} \rfloor + 1} x_r = (k+1)\left(\left\lfloor \frac{n}{k+1} \right\rfloor + 1\right) - n$$
$$+(k+1)\left(\left\lfloor \frac{n}{k} \right\rfloor - \left\lfloor \frac{n}{k+1} \right\rfloor - 1\right) + (2n+1),$$
$$\sum_{r=\lfloor \frac{n}{k} \rfloor + 1}^{\lfloor \frac{n}{k-1} \rfloor + 1} y_r = \left(2k\left\lfloor \frac{n}{k} \right\rfloor + k\right) + (k-1)\left(\left\lfloor \frac{n}{k-1} \right\rfloor - \left\lfloor \frac{n}{k} \right\rfloor - 1\right)$$
$$+\left(n - (k-1)\left\lfloor \frac{n}{k-1} \right\rfloor\right),$$

we deduce that both sums above are equal to $(k+1)\lfloor \frac{n}{k} \rfloor + n + 1$. Thus, each $X$ and $Y$ can be seen as the sum of $(k+1)\lfloor \frac{n}{k} \rfloor + n + 1$ values of the function $g$; however, for any $i$, $1 \le i \le (k+1)\lfloor \frac{n}{k} \rfloor + n + 1$, the $i$th smallest value of $g(r)$ involved in the expression for $X$ is no larger than the $i$th smallest value of $g(r)$ involved in the expression for $Y$. This proves that $X \le Y$, i.e., $2G(k) - G(k+1) - G(k-1) \le 0$, so that the sequence $G(k)$, $1 \le k \le n$, is convex. □

Now, we extend the convexity proof to the sequence (5.2), provided that the values $p_j$, $1 \le j \le n$, follow a non-increasing sequence

$$p_1 \ge p_2 \ge \cdots \ge p_n. \tag{5.5}$$

According to the terminology adopted in Sect. 2.1.1, the values $p_j$ form an LPT sequence.

**Theorem 5.2** *The sequence $P(k)$, $1 \le k \le n$, of the form (5.2) is convex, provided that (5.5) holds.*

*Proof* For a given $j$, $1 \le j \le n$, define

$$A_j(k) = \left[2g\left(\left\lceil \frac{j}{k} \right\rceil\right) - g\left(\left\lceil \frac{j}{k+1} \right\rceil\right) - g\left(\left\lceil \frac{j}{k-1} \right\rceil\right)\right],\ 2 \le k \le n-1.$$

By Theorem 5.1, due to the convexity of the sequence $G(k)$, $1 \le k \le n$, we deduce that

$$\sum_{i=1}^{q} A_i(k) \le 0 \tag{5.6}$$

for each $k$, $2 \le k \le n-1$, and all $q$, $1 \le q \le n$.

In order to prove the theorem, we need to demonstrate that the inequality

$$\sum_{j=1}^{n} p_j A_j(k) \le 0 \tag{5.7}$$

holds for each $k$, $2 \le k \le n-1$.

Fix a $k$, $2 \le k \le n-1$, and transform

$$\begin{aligned}\sum_{j=1}^{n} p_j A_j(k) &= p_n\left(\sum_{i=1}^{n} A_i(k) - \sum_{i=1}^{n-1} A_i(k)\right) + p_{n-1}\left(\sum_{i=1}^{n-1} A_i(k) - \sum_{i=1}^{n-2} A_i(k)\right) + \cdots \\ &\quad + p_1 A_1(k) \\ &= p_n \sum_{i=1}^{n} A_i(k) + (p_{n-1} - p_n)\sum_{i=1}^{n-1} A_i(k) + (p_{n-2} - p_{n-1})\sum_{i=1}^{n-2} A_i(k) + \cdots \\ &\quad + p_1 A_1(k) \\ &= \sum_{j=2}^{n}\left[(p_{j-1} - p_j)\sum_{i=1}^{j-1} A_i(k)\right] + p_n \sum_{i=1}^{n} A_i(k).\end{aligned}$$

The last right-hand expression is non-positive due to (5.6) and the LPT numbering of $p_j$, so that the desired inequality (5.7) holds and the sequence $P(k)$, $1 \le k \le n$, is convex. □

The convexity of the sequence $P(k)$, $1 \le k \le n$, as established in Theorem 5.2, allows us to use an efficient binary search algorithm to solve several problems considered in this book (see, e.g., Sects. 16.2.4 and 17.4).

## 5.3 Combinatorial Counting

Many scheduling problems reduce to enumeration of several feasible solutions, and we are required to choose the best among them as an optimal solution. For completeness, we provide a brief review of some basic principles of combinatorics, which are often used in this book.

- **Combinations**: A $v$-combination of a set $S$ is a subset of $v$ distinct elements of $S$. If $|S| = u$, then the number of $v$-combinations is equal to

$$\binom{u}{v} = \frac{u!}{v!(u-v)!}.$$

  Since $\frac{u!}{(u-v)!} = u(u-1) \times \cdots \times (u-v+1) \le u^v$, we estimate

$$\binom{u}{v} \le \frac{u^v}{v!}, \tag{5.8}$$

  which for a fixed $v$ yields

$$\binom{u}{v} = O(u^v)$$

- **Arrangements**: A $v$-arrangement of a set $S$ is an ordered subset of $v$ distinct elements of $S$. If $|S| = u$, then the number of $v$-arrangements is equal to

$$\binom{u}{v} v! = O(u^v), \tag{5.9}$$

  because (5.8) holds.
- **Partitions**: A partition of a positive integer $u$ into exactly $v$ positive summands is a sequence $(z_1, z_2, ..., z_v)$ of integers such that $u = z_1 + z_2 + \cdots + z_v$ and $z_1 \ge z_2 \ge \cdots \ge z_v \ge 1$. The total number of partitions of $u$ into at most $v$ positive summands (i.e., exactly $v$ non-negative summands) is denoted by $P_u^{(\le v)}$ and can be estimated as

$$\frac{u^{v-1}}{v!(v-1)!} = O\left(u^{v-1}\right).$$

  The total number of partitions of $u$ into exactly $v$ positive summands is

$$P_u^{(v)} = P_u^{(\le v)} - P_u^{(\le v-1)} = O\left(u^{v-1}\right). \tag{5.10}$$

- **Compositions**: A composition of an integer $u$ made of $v$ summands is a sequence $(z_1, z_2, ..., z_v)$ of positive integers such that $u = z_1 + z_2 + \cdots + z_v$. The total number of compositions $C_u^{(v)}$ in exactly $v$ summands is given by

$$C_u^{(v)} = \binom{u-1}{v-1}, \tag{5.11}$$

which can be estimated as $O\big((u-1)^{v-1}\big) = O\big(u^{v-1}\big)$ because (5.8) holds. The number of compositions into at most $v$ positive summands (i.e., exactly $v$ non-negative summands) is

$$C_u^{(\leq v)} = \binom{u+v-1}{v-1}, \tag{5.12}$$

which can be estimated as $O\big((u+v-1)^{v-1}\big) = O\big(u^{v-1}\big)$, because (5.8) and $u \geq v$ hold.

The number of compositions of all integers that do not exceed $u$ into at most $v$ positive summands can be expressed as

$$C_{(\leq u)}^{(\leq v)} = C_u^{(\leq v+1)} = \binom{u+v}{v}. \tag{5.13}$$

## 5.4 Bibliographic Notes

Convex sequences play an important role in the derivation of various inequalities. Wu and Debnath (2007) give a necessary and sufficient condition for a sequence to be convex in terms of majorization, and this gives access to a powerful toolkit that is systematically exposed by Marshall and Olkin (1979). Various applications of convex sequences to the problems of combinatorics, algebra, and calculus are studied by Mercer (2005), Toader (1996), and Wu and Debnath (2007). An additional important link between convex and log-convex sequences is pointed out by Došlić (2009). In Operations Research, an application of convex sequences to a special form of the assignment problem and their relations to the famous Monge property (see Sect. 4.1.3 for definitions) is discussed in Sect. 5.2 of the monograph by Burkard et al. (2009).

The $V$-shaped sequences often arise in optimization over a set of permutations, in particular in machine scheduling. For a number of scheduling problems, it is possible to establish that an optimal sequence of jobs possesses the $V$-shaped property; e.g., the elements of the input sequence of the processing times can be interchanged so that the resulting sequence is $V$-shaped. This property has been explored in numerous papers. Here, we refer to only five, mostly with the words "$V$-shaped" in the title (see Alidaee and Rosa (1995), Alturki et al. (1996), Federgruen and Mosheiov (1997), Mittenhal et al. (1995), and Mosheiov (1991)).

Theorems 5.1 and 5.2 are proved in Rustogi and Strusevich (2011) and Rustogi and Strusevich (2012).

The pyramidal or $\Lambda$-shaped sequences are the main objects of study in identifying efficiently solvable combinatorial optimization problems, in particular, the traveling salesman problem. Here, we only refer to the two most recent surveys by Burkard et al. (1998) and Kabadi (2007).

The sequence (5.2), or some version of it, is commonly found in the scheduling literature, especially in problems in which a given set of jobs needs to be divided into a known number of groups. In Chaps. 16 and 17, we encounter such sequences while studying the problem of scheduling jobs with rate-modifying activities.

It should be noticed that although the ceiling function and its counterpart, the floor function find applications in many areas, publications that study the relations that involve these functions are quite scarce; we mention only Chap. 3 of Graham et al. (1989) and Nyblom (2002). There are several papers devoted to obtaining closed form expressions of the sums $\sum_{j=1}^{n}\left\lceil \frac{j}{k}\right\rceil$ and $\sum_{j=1}^{n}\left\lfloor \frac{j}{k}\right\rfloor$, as well as for their generalizations; see Sivakumar et al. (1997) and Tuenter (1999, 2000).

In Sect. 5.3, we mainly follow the terminology and notation of Flajolet and Sedgewick (2009) and Mott et al. (1986).

## References

Alidaee B, Rosa D (1995) A note on the $V$-shaped property in one-machine scheduling. J Oper Res Soc 46:128–132

Alturki UM, Mittenthal J, Raghavachari M (1996) A dominant subset of $V$-shaped sequences for a class of single machine sequencing problems. Eur J Oper Res 88:345–347

Burkard RE, Deineko VG, van Dal R, van der Veen JAA, Woeginger GJ (1998) Well-solvable special cases of the traveling salesman problem: a survey. SIAM Rev 40:496–546

Burkard R, Dell'Amico M, Martello S (2009) Assignment problems. SIAM, Philadelphia

Došlić T (2009) Log-convexity of combinatorial sequences from their convexity. J Math Inequal 3:437–442

Federgruen A, Mosheiov G (1997) Single machine scheduling problems with general breakdowns, earliness and tardiness costs. Oper Res 45:66–71

Flajolet P, Sedgewick R (2009) Analytic combinatorics. Cambridge University Press, Cambridge

Graham RL, Knuth DE, Patashnik O (1989) Concrete mathematics. Addison-Wesley, New York

Kabadi SN (2007) Polynomially solvable cases of the TSP. In: Gutin G, Punnen AP (eds) The traveling salesman problem and its variations. Springer, Berlin, pp 489–583

Marshall AW, Olkin I (1979) Inequalities: Theory of majorization and its applications. Academic Press, New York

Mercer AM (2005) Polynomials and convex sequence inequalities. J Inequal Pure Appl Math 6:1–4

Mittenhal J, Raghavachari M, Rana AI (1995) $V$-shapes and Lambda-spared properties for optimal single-machine schedules for a class of nonseparable penalty-functions. Eur J Oper Res 86:262–269

Mosheiov G (1991) $V$-shaped policies for scheduling deteriorating jobs. Oper Res 39:979–991

Mott JL, Kandel A, Baker TP (1986) Discrete mathematics for computer scientists and mathematicians. Prentice-Hal, Englewood Cliffs

Nyblom MA (2002) Some curious sequences involving floor and ceiling functions. Am Math Mon 109:559–564

Rustogi K, Strusevich VA (2011) Convex and $V$-shaped sequences of sums of functions that depend on ceiling functions. J Integ 14:11.1.5. http://www.cs.uwaterloo.ca/journals/JIS/VOL14/Strusevich/strusevich2.html

Rustogi K, Strusevich VA (2012) Single machine scheduling with general positional deterioration and rate-modifying maintenance. Omega 40:791–804

Sivakumar T, Dimopoulos NJ, Lu W-S (1997) Analytical formulas for $\sum_{i=1}^{n}\left\lceil \frac{i}{p}\right\rceil$ and $\sum_{i=1}^{n}\left\lfloor \frac{i}{p}\right\rfloor$. Pi Mu Epsilon J 10: 458–461

Toader G (1996) On Chebychev's inequality for sequences. Discr Math 161:317–322

Tuenter HJH (1999) Another problem for Carl Friedrich: on the sums $\sum_{i=1}^{n}\left\lceil \frac{i}{p}\right\rceil$ and $\sum_{i=1}^{n}\left\lfloor \frac{i}{p}\right\rfloor$. The Math. Gazette 83(498):495–497

Tuenter HJH (2000) On the sums $\sum_{I=1}^{N}\left\lceil \frac{I}{P}\right\rceil^{M}$ and $\sum_{I=1}^{N}\left\lfloor \frac{I}{P}\right\rfloor^{M}$. Pi Mu Epsilon J 11:97–99

Wu S, Debnath L (2007) Inequalities for convex sequences and their application. Comput Math Appl 54:525–534

# Part II
# Scheduling with Time-Changing Effects

# Chapter 6
# Introduction to Time-Changing Effects

In the models of classical scheduling such as those reviewed in Part I of this book, the processing times of the jobs are given in advance and remain unchanged during the whole planning period. Since the early 1990s, there has been a considerable interest in enhanced models which allow the processing times of jobs to be affected by their locations in the schedule. The material of this part presents the approaches to handling scheduling problems under such time-changing effects. These models are further extended in Part III, where we additionally allow for introducing activities on the processing machines that are similar in nature to maintenance and change the processing capabilities of the machines.

The purpose of this chapter was to introduce the reader to various time-changing effects. For simplicity, below we focus on a single machine environment. We are given jobs of set $N = \{1, 2, \ldots, n\}$ to be processed on a single machine. Each job $j \in N$ is associated with its "*normal*" processing time $p_j$. It is convenient to think of normal processing times as the time requirements under normal processing conditions of the machine, which might change during the processing, thereby affecting the actual processing time of a job.

In the literature on scheduling with time-changing processing times, traditionally there is a distinction between the so-called deterioration effects and learning effects.

Informally, under a *deterioration* effect, the later a job is placed in a schedule, the longer it takes to process it. A common rationale, often found in practice, is that as the schedule evolves the processing conditions of the machine get worse. For example, in manufacturing, if the machine is a cutting tool it may lose its initial sharpness and that will increase the actual processing times of some jobs.

Under a *learning* effect, an opposite phenomenon is observed: The later a job is scheduled, the shorter its actual processing time is. To motivate a learning effect, think of the machine as a human operator who gains experience during the process, which leads to a certain processing time reduction.

There is no reason to limit consideration of time-changing effects to monotone effects only, such as deterioration and learning. This gives rise to an effect which has a non-trivial influence on the actual processing times. For example, if a human operator processes jobs on a certain equipment, then during the process the equipment might be subject to wear and tear, i.e., it might deteriorate with time; however, the operator simultaneously gains additional skills by learning from experience.

Mathematically, time-changing effects are represented by formulae that explicitly show how normal processing time of a job is affected. There are several main types of effects studied in the literature, which informally can be classified as follows:

- *Positional* effects: The actual processing time of a job is a function of its normal processing time and the position it takes in a schedule;
- *Start-time-dependent* effects: The actual processing time of a job is a function of its normal processing time and its start time in a schedule;
- *Cumulative* effects: The actual processing time of a job depends on its normal processing time and a function of the normal processing times of previously scheduled jobs;
- *Combined* effects: These are non-trivial effects which usually combine a positional effect with either a start-time-dependent effect or a cumulative effect.

In the reminder of this chapter, we present a brief discussion of each of the above effects, mainly for a single machine environment.

## 6.1 Positional Effects

In general, a *positional effect* on the actual processing time of job $j$ is defined by a function $g_j(r)$, where $r$, $1 \leq r \leq n$, is the position of job $j$ in the processing sequence. The actual processing time $p_j(r)$ of job $j$ sequenced in position $r$ is given by

$$p_j(r) = p_j g_j(r),\ \ j \in N, 1 \leq r \leq n, \tag{6.1}$$

where $g_j(r)$ is called a (job-dependent) *positional factor*. It is often assumed that $g_j(1) = 1,\ j \in N$, which guarantees that for the job that is sequenced first, i.e., in position $r = 1$, the actual processing time is equal to its normal time.

The formula (6.1) is useful when we want to trace the influence of the positional factors on the normal time $p_j$; otherwise, it suffices to say that actual processing times are taken from a matrix $\mathbf{P} = \left(p_{jr} = p_j(r)\right)_{n \times n}$, where the rows represent the jobs and the columns represent the positions. If each row of matrix $\mathbf{P}$ is monotone non-decreasing (or non-increasing), then we have a situation of *positional deterioration* (or of *positional learning*, respectively). In the most general setting, the rows of matrix $\mathbf{P}$ need not be monotone. Among the first papers that consider scheduling problems with a general positional effect (6.1) are Bachman and Janiak (2004) and Mosheiov and Sarig (2009).

A job-dependent positional effect implies that each job has a different (unique) effect on the state of the machine. Quite often, though, a change in the machine's processing conditions reflects similarly on all jobs, in which case we may talk about a *positional job-independent effect* given by

$$p_j(r) = p_j g(r), \ \ j \in N, 1 \le r \le n, \tag{6.2}$$

where $g(r), 1 \le r \le n$, forms an array of *positional factors* that is common for all jobs. Similar to the above, if array $g(r), 1 \le r \le n$, is monotone non-decreasing (or non-increasing), then we have a situation of positional deterioration (or of positional learning, respectively). For the most general job-independent positional effect, we make no assumption regarding the monotonicity of the elements of array $g(r), 1 \le r \le n$.

For an informal illustration of a positional deterioration effect, imagine that in a manufacturing shop, there are several parts that need a hole of the same diameter to be punched through by a pneumatic punching unit. Ideally, the time that is required for such an operation depends on the thickness of the metal to be punched through, and this will determine normal processing times for all parts. In reality, however, there occurs an unavoidable gas leakage after each punch, due to which the punching unit loses pressure, so that the later a part is subject to punching, the longer it takes to perform it, as compared to the duration under perfect conditions. Clearly, a positional deterioration effect is observed.

Being part of the academia, the authors have noticed that positional learning takes place when a teacher marks a number of coursework scripts based on the same question paper. It takes a reasonably long time to mark the first two or three scripts, then the teacher realizes the key factors to be checked, typical strong or weak points to be looked for, and the marking process goes faster and faster with each marked script.

Research on scheduling problems with positional effects conducted prior to the critical review Rustogi and Strusevich (2012) has had several limitations, from the point of view of the studied models. First, earlier authors focused on monotone effects only, such as learning and deterioration; moreover, these two types of effects have been often considered separately despite noticeable methodological similarities in their treatment. Second, assumptions on the exact shape of positional factors have been made (i.e., polynomial or exponential), despite the fact that many results would hold for an arbitrary array of positional factors, as defined, e.g., in (6.2).

Single machine problems under positional effects to minimize the makespan, the total completion time, and more general objective functions are considered in Chap. 7. Problems on parallel machines under positional effects are addressed in Chap. 11

## 6.2 Time-Dependent Effects

We distinguish between two types of a start-time-dependent effect: additive and multiplicative. Under the *additive* effect, the actual processing time of a job is obtained as the sum of its normal processing time and a function that depends on its start time. On the other hand, under the *multiplicative* effect, the actual processing time of a job is obtained as the product of its normal processing time and a function that depends on its start time. Let $p_j(\tau)$ denote the actual processing time of job $j \in N$ that starts at time $\tau \geq 0$. Then, under a general additive effect, we define

$$p_j(\tau) = p_j + f_j(\tau), \tag{6.3}$$

while under a general multiplicative effect, we define

$$p_j(\tau) = p_j f_j(\tau), \tag{6.4}$$

where $f_j$ is a job-dependent function of start time.

It is common to assume that in (6.3) for each $j \in N$, the equality $f_j(0) = 0$ holds. Similarly, in (6.4) for each $j \in N$, the equality $f_j(0) = 1$ holds. These assumptions make sure that for the job that is sequenced first, i.e., starts at time zero, the actual processing time is equal to its normal time.

If in (6.3), the inequalities $f_j(\tau) \geq 0$, $j \in N$, hold, we deal with a deterioration effect, while the inequalities $f_j(\tau) < 0$, $j \in N$, define a learning effect. In the latter case, additional assumptions are normally imposed that prevent actual processing times from becoming negative.

Similarly, in (6.4), a deterioration effect and a learning effect are represented by the inequalities $f_j(\tau) \geq 1$, $j \in N$, and $0 < f_j(\tau) \leq 1$, $j \in N$, respectively.

A systematic exposition of various aspects of scheduling with start-time-dependent effects has been undertaken in the monograph by Gawiejnowicz (2008). In particular, Sect. 5.4 of this book presents numerous examples of practical applications of scheduling problems. One of these examples, based on paper Gawiejnowicz et al. (2008), describe a single worker who performs maintenance of corroded items, and for an item treated later in a schedule, the maintenance time increases since the item gets more corroded than in the beginning of the process.

Additive start-time-dependent effects are probably the most studied among known time-changing effects that affect the actual processing time of jobs in a schedule. Papers Shafransky (1978) and Melnikov and Shafransky (1980) must be seen as historically the first that introduce not only scheduling models with (additive) start-time-dependent effects, but open up the whole area of scheduling with changing times. There are several influential surveys, such as Alidaee and Womer (1999) and Cheng et al. (2004), that review the developments in the area. Reviews of start-time-dependent models with learning effects and deterioration effects are provided in Biskup (2008) and Janiak and Kovalyov (2006), respectively. Most of the results for an additive start-time-dependent effect are derived in the case of linear functions $f_j(\tau)$.

Among the first papers which address a multiplicative start-time-dependent effect (with either polynomial or linear functions $f_j(\tau)$) are works by Kononov (1998) and Kuo and Yang (2007), which, however, are not totally free from errors (see Sect. 9.3 for details).

Apart from the start-time-dependent effects in the pure form, we also consider their enhanced versions, combined with positional effects. For example, we introduce a job-independent start-time-dependent additive effect combined with a general job-independent positional effect, so that the actual processing time of job $j$ that is scheduled in the $r$th position and starts at time $\tau$ is given by

$$p_j(\tau; r) = \big(p_j + f(\tau)\big)g(r),$$

where $f$ is a function, common to all jobs, and array $g(r)$, $1 \le r \le n$, is a monotone sequence of positional factors and defines a positional effect. We also consider its analogue given by

$$p_j(\tau; r) = p_j f(\tau)g(r), \tag{6.5}$$

i.e., a combination of a job-independent start-time-dependent multiplicative effect combined with a general job-independent positional effect. Studies on a similar effect have been initiated in Yin et al. (2009).

Scheduling problems with start-time-dependent effects, in the pure and combined formats, are considered in Chap. 8 (in the case of the additive effects) and in Chap. 9 (in the case of the multiplicative effects). Problems on parallel machines under pure and combined start-time-dependent effects are addressed in Chap. 11

## 6.3 Cumulative Effects

Suppose that the jobs are processed on a single machine in accordance with a permutation $\pi = (\pi(1), \ldots, \pi(n))$. Under a cumulative effect, the actual processing time $p_j(r)$ of a job $j$ scheduled in position $r$, $1 \le r \le n$, depends on the normal processing times of the previously sequenced jobs. One of the most general variants of a pure cumulative effect defines the actual processing time of job $j = \pi(r)$ sequenced in position $r$, $1 \le r \le n$, as

$$p_j(r) = p_j f(P_r), \tag{6.6}$$

where

$$P_r = \sum_{h=1}^{r-1} p_{\pi(h)}$$

is the sum of the normal processing times of the earlier sequenced jobs. In the case of learning, $f$ is a non-increasing function, while in the case of deterioration, $f$ is a non-decreasing function. One of the first versions of the cumulative effect given by

$$p_j(r) = p_j\left(1 + \sum_{h=1}^{r-1} p_{\pi(h)}\right)^A, \tag{6.7}$$

is introduced by Kuo and Yang (2006a, b), who studied the effect in the learning form, with $A < 0$.

A cumulative effect can be combined with a general job-independent positional effect, so that the actual processing time of job $j$ scheduled in the $r$th position of a permutation $\pi$ is given by

$$p_j(r) = p_j f(P_r) g(r), \tag{6.8}$$

which is similar to (6.5). Studies on a similar effect have been initiated in Yin et al. (2010).

It is assumed that in (6.6) and (6.8), the equalities $f(0) = 1$ and $g(1) = 1$ hold, which guarantees that for the job which is the first in the processing sequence, the actual processing time is equal to its normal time.

A common drawback of the scheduling models with a cumulative effect in the form (6.6) is related to lack of convincing practical motivations. In particular, there is no convincing justification why the actual processing time of the jobs must depend on normal times of the previously sequenced jobs. Rustogi and Strusevich (2015) introduce an alternative model which is more relevant to practice. Consider an effect that arises when a job $j \in N$ is associated not only with normal processing time $p_j$ but also with two additional parameters, $b_j$ and $q_j$. The actual processing time of job $j$ scheduled in the $r$th position of permutation $\pi$ is defined by

$$p_j(r) = p_j\left(1 + b_j \sum_{h=1}^{r-1} q_{\pi(h)}\right). \tag{6.9}$$

where $b_j > 0$ under a deterioration effect and $b_j < 0$ under a learning effect. No explicit dependence on the normal time of the previously scheduled jobs is assumed, and the values of $b_j$ can be understood as job-dependent rates that reflect how sensitive a particular job is to the previously scheduled jobs.

For illustration of this generalized model, suppose that a floor sanding machine is used to treat floors in several rooms. The normal time $p_j$ is the time requirement for sanding floors in room $j$, provided that the new sanding belt/disk is used. The value of $q_j$ can be seen as the amount of generated saw dust or an appropriately measured wear of the sanding belt/disk, which depends on the area of room $j$ and on the initial quality of the floor in the room but not explicitly on the time of treatment. For some rooms, the actual treatment time can be seriously affected by the quality of the equipment, whereas for some rooms, the effect is less noticeable, which is captured in the rate parameter $b_j$. It is not difficult to identify a similar cumulative deterioration effect in other activities/industries.

To illustrate the effect (6.9) in a learning environment, consider the following situation. Suppose a computer programmer is supposed to write $n$ software pieces for a particular project. These pieces can be developed in any order. Developing these pieces require particular transferable technical skills (such as manipulating a certain data structure), which the programmer initially does not possess. In the beginning of the process, a software piece $j \in N$ can be completed in $p_j$ time units.

Assume that after completing a particular software piece $j$, the technical skill of the programmer increases by $q_j$ appropriately measured units and that skill might help to speed up the creation of any piece to follow. Thus, the actual time needed to create a particular piece depends on the accumulated skills gained during the development of previously created pieces. Formally, the development time of a software piece decreases linearly with the technical skill of the programmer, so that the actual time taken to write a software piece $j = \pi(r)$ is given by $p_j(\pi; r) = p_j - a_j \sum_{h=1}^{r-1} q_{\pi(h)}$, where the quantity $a_j$ defines how sensitive the development time for software piece $j$ is to the gained technical skills. This formulation can be written in terms of the effect (6.9) with $b_j = -a_j/p_j$.

Scheduling problems under pure and combined cumulative effects are addressed in Chap. 10.

## References

Alidaee B, Womer NK (1999) Scheduling with time dependent processing times: review and extensions. J Oper Res Soc 50:711–720

Bachman A, Janiak A (2004) Scheduling jobs with position-dependent processing times. J Oper Res Soc 55:257–264

Biskup D (2008) A state-of-the-art review on scheduling with learning effects. Eur J Oper Res 188:315–329

Cheng TCE, Ding Q, Lin BMT (2004) A concise survey of scheduling with time-dependent processing times. Eur J Oper Res 152:1–13

Gawiejnowicz S (2008) Time-dependent scheduling. Springer, Berlin

Gawiejnowicz S, Kurc W, Pankowska L (2008) Pareto and scalar bicriterion scheduling of deteriorating jobs. Comput Oper Res 33:746–767

Janiak A, Kovalyov MY (2006) Scheduling deteriorating jobs. In: Janiak A (ed) Scheduling in computer and manufacturing systems. Wydawnictwa Komuniikacji i Łączności, Warsaw, pp 12–25

Kononov A (1998) Single machine scheduling problems with processing times proportional to an arbitrary function. Diskretnyj Analiz i Issledovanie Operatsij, Seria 1, 5(3):17–37 (in Russian)

Kuo W-H, Yang D-L (2006a) Minimizing the makespan in a single machine scheduling problem with a time-based learning effect. Inf Proc Lett 97:64–67

Kuo W-H, Yang D-L (2006b) Minimizing the total completion time in a single-machine scheduling problem with a time-dependent learning effect. Eur J Oper Res 174:1184–1190

Kuo W-H, Yang D-L (2007) Single-machine scheduling problems with start-time dependent processing time. Comput Math Appl 53:1658–1664

Melnikov OI, Shafransky YM (1980) Parametric problem of scheduling theory. Cybernetics 15:352–357

Mosheiov G, Sarig A (2009) Scheduling a maintenance activity and due-window assignment on a single machine. Comput Oper Res 36:2541–2545

Rustogi K, Strusevich VA (2012) Simple matching versus linear assignment in scheduling models with positional effects: a critical review. Eur J Oper Res 22:393–407

Rustogi K, Strusevich VA (2015) Single machine scheduling with a generalized job-dependent cumulative effect. University of Greenwich, London, Report-09-2015

Shafransky YM (1978) On optimal sequencing for deterministic systems with a tree-like partial order. Vestsi Akad Navuk BSSR, Ser Fiz-Mat Navuk 2:120 (in Russian)

Yin Y, Xu D, Sun K, Li H (2009) Some scheduling problems with general position-dependent and time-dependent learning effects. Inf Sci 179:2416–2425

Yin Y, Xu D, Wang J (2010) Single-machine scheduling problems with a general sum-of-actual-processing-times-based and job-position-based learning effect. Appl Math Model 34:3623–3630

# Chapter 7
# Scheduling with Positional Effects

In this chapter, we study the scheduling problems of minimizing makespan and total completion time on a single machine, provided that the actual processing times of the jobs are subject to a positional effect.

For a job $j \in N = \{1, 2, \dots, n\}$, its normal processing time $p_j$ is given. As stated in Sect. 6.1, a general *positional effect* on the actual processing time of job $j$ can be defined by a function $g_j(r)$, where $r$, $1 \le r \le n$, is a position of job $j$ in the processing sequence. The actual processing time $p_j(r)$ of job $j$ sequenced in position $r$ is given by

$$p_j(r) = p_j g_j(r), \;\; j \in N, 1 \le r \le n. \tag{7.1}$$

where $g_j(r)$ is a (job-dependent) positional factor.

If we do not need to trace how exactly a given normal time $p_j$ is affected by a positional factor, we say that the actual processing times are taken from a matrix $\mathbf{P} = \left(p_{jr} = p_j(r)\right)_{n \times n}$, where the rows represent the jobs and the columns represent the positions. In what follows, it is convenient to refer to an element $p_{jr}$ located in Row $j$, $1 \le j \le n$, and column $r$, $1 \le r \le n$, of matrix $\mathbf{P}$ simply as $p_j(r)$. If each row of matrix $\mathbf{P}$ is monotone non-decreasing (or non-increasing), then we have a situation of positional deterioration (or of positional learning, respectively). In the most general setting, the rows of matrix $\mathbf{P}$ need not be monotone.

As adopted throughout this book, if job $j$ is sequenced in position $\pi(r)$ of permutation $\pi = (\pi(1), \pi(2), \dots, \pi(n))$, its completion time is denoted either by $C_j(\pi)$ or by $C_{\pi(r)}$, whichever is more convenient.

We denote the problems of minimizing the makespan and the total completion time on a single machine subject to an effect (7.1) by $1\left|p_j(r) = p_j g_j(r)\right|C_{\max}$ and $1\left|p_j(r) = p_j g_j(r)\right|\sum C_j$, respectively. We also consider the problem of minimizing a linear combination of these two criteria, i.e., the function $\xi C_{\max} + \eta \sum C_j$, where $\xi$ and $\eta$ are non-negative coefficients. The latter problem is denoted by $1\left|p_j(r) = p_j g_j(r)\right|\xi C_{\max} + \eta \sum C_j$. In Sect. 7.1, we show that each of the problems $1\left|p_j(r) = p_j g_j(r)\right|\Phi$ with $\Phi \in \left\{C_{\max}, \sum C_j, \xi C_{\max} + \eta \sum C_j\right\}$ under a job-dependent positional effect (7.1) reduces to a square linear assignment problem (LAP) and is solvable in $O\left(n^3\right)$ time.

In Sect. 7.2, we pay attention to the problems with a positional job-independent effect given by

$$p_j(r) = p_j g(r), \ j \in N, 1 \le r \le n, \tag{7.2}$$

where $g(r)$, $1 \le r \le n$, forms an array of (job-independent) positional factors that is common for all jobs. Similar to the above, if array $g(r)$, $1 \le r \le n$, is monotone non-decreasing (or non-increasing), then we have a situation of positional deterioration (or of positional learning, respectively). For the most general job-independent positional effect, we make no assumption regarding the monotonicity of the elements of array $g(r)$, $1 \le r \le n$. The corresponding single machine problems with the objective $\Phi \in \{C_{\max}, \sum C_j, \xi C_{\max} + \eta \sum C_j\}$ are denoted by $1\left|p_j(r) = p_j g(r)\right|\Phi$.

As a rule, scheduling problems with a job-independent positional effect of the form (7.2) reduce to an $n \times n$ LAP with a product matrix, i.e., to minimizing a linear form (see Sect. 4.1.3). For an arbitrary permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ of jobs, such a linear form can be generically written as

$$\Phi(\pi) = \sum_{r=1}^{n} W(r) p_{\pi(r)} + \Gamma, \tag{7.3}$$

where the values $W(r)$ are *positional weights* that depend only on a position $r$, $1 \le r \le n$, of a job in sequence $\pi$. As stated in Sect. 2.1, a permutation that minimizes function $\Phi(\pi)$ of the form (7.3) over all permutations of jobs of set $N$ can be found by Algorithm Match which requires $O(n \log n)$ time. We also show that if the sequence $W(r)$, $1 \le r \le n$, of positional weights is monotone, then an optimal permutation can be found by ordering the jobs in accordance with a priority rule, typically either the SPT or the LPT rule applied to the normal processing times $p_j$. Recall that if the jobs are numbered in accordance with the LPT rule, then

$$p_1 \ge p_2 \ge \cdots \ge p_n, \tag{7.4}$$

while if they are numbered in accordance with the SPT rule, then

$$p_1 \le p_2 \le \cdots \le p_n. \tag{7.5}$$

According to the terminology introduced in Sect. 2.1, if a problem is solvable by the LPT rule, it admits a 1-priority function $\omega(j) = p_j$, $j \in N$. For a problem solvable by the SPT rule, a 1-priority function can be written as either $\omega(j) = -p_j$, $j \in N$, or $\omega(j) = 1/p_j$, $j \in N$.

The existence of 1-priority functions is necessary (but not sufficient) for the objective function to be priority-generating, which is important for solving relevant problems under series-parallel precedence constraints. The latter topic is the focus of Sect. 7.3. Please consult Chap. 3 for the relevant definitions and techniques, such as the proof of an objective function to be priority-generating.

Recall that throughout this book for a (partial) permutation of jobs $\pi$, its length, i.e., the number of elements in $\pi$, is denoted by $|\pi|$.

The remainder of this chapter is organized as follows. Problems of scheduling independent jobs under a job-dependent and a job-independent positional effect are considered in Sects. 7.1 and 7.2, respectively. Problems with series-parallel precedence constrains are studied in Sect. 7.3.

## 7.1 Scheduling Independent Jobs Under Job-Dependent Positional Effect

Given a matrix $\mathbf{P} = \left(p_{jr} = p_j(r)\right)_{n\times n}$ of possible actual processing times, consider problem $1\left|p_j(r) = p_j g_j(r)\right|C_{\max}$. In any feasible solution, the actual processing time of job $j \in N$ is determined by selecting exactly one element from Row $j$. Since exactly one job must be assigned to a position $r$, $1 \le r \le n$, it follows that exactly one element from each column of $\mathbf{P}$ will be selected. Thus, a feasible solution is associated with a choice of $n$ elements of matrix $\mathbf{P}$ such that no two of them belong to the same row or to the same column.

For problem $1\left|p_j(r) = p_j g_j(r)\right|C_{\max}$, the makespan is equal to the sum of actual processing times. The actual times that deliver the smallest makespan can be found by solving a linear assignment problem. Introduce $n^2$ Boolean variables $x_{jr}$, where

$$x_{jr} = \begin{cases} 1, & \text{job } j \text{ is assigned to position } r \\ 0, & \text{otherwise;} \end{cases} \qquad 1 \le j \le n, 1 \le r \le n.$$

Then, problem $1\left|p_j(r) = p_j g_j(r)\right|C_{\max}$ can be formulated as the following linear assignment problem (LAP)

$$\begin{aligned} \text{minimize} \quad & \sum_{j=1}^{n}\sum_{r=1}^{n} p_j(r)x_{jr} \\ \text{subject to} \quad & \sum_{r=1}^{n} x_{jr} = 1, \qquad 1 \le j \le n; \\ & \sum_{j=1}^{n} x_{jr} = 1, \qquad 1 \le r \le n; \\ & x_{jr} \in \{0,1\}, \qquad 1 \le j \le n, 1 \le r \le n. \end{aligned} \tag{7.6}$$

We now pass to problem $1\left|p_j(r) = p_j g_j(r)\right|\sum C_j$. It has been shown in Sect. 2.1.1 that for a given sequence of jobs, the contribution of a job scheduled in position $r$ is equal to its processing time multiplied by its position in the sequence counted by the rear of the sequence. Formally, for an arbitrary permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$, the total completion time $F(\pi)$ is given by

$$F(\pi) = \sum_{r=1}^{n} (n - r + 1) p_{\pi(r)}(r), \tag{7.7}$$

where $p_{\pi(r)}(r)$ is the actual processing time of a job $j = \pi(r)$ sequenced in position $r$ in permutation $\pi$ (see (2.9)). This reduces problem $1\left|p_j(r) = p_j g_j(r)\right|\sum C_j$ to the following LAP

$$\begin{array}{lll} \text{minimize} & \displaystyle\sum_{j=1}^{n}\sum_{r=1}^{n}(n-r+1)p_j(r)x_{jr} & \\ \text{subject to} & \displaystyle\sum_{r=1}^{n} x_{jr} = 1, & 1 \le j \le n; \\ & \displaystyle\sum_{j=1}^{n} x_{jr} = 1, & 1 \le r \le n; \\ & x_{jr} \in \{0, 1\}, & 1 \le j \le n, 1 \le r \le n. \end{array} \tag{7.8}$$

It is straightforward to see that problem $1\left|p_j(r) = p_j g_j(r)\right|\xi C_{\max} + \eta \sum C_j$, where $\xi$ and $\eta$ are given non-negative coefficients, reduces to the following LAP

$$\begin{array}{lll} \text{minimize} & \displaystyle\sum_{j=1}^{n}\sum_{r=1}^{n}(\xi + (n-r+1)\eta)p_j(r)x_{jr} & \\ \text{subject to} & \displaystyle\sum_{r=1}^{n} x_{jr} = 1, & 1 \le j \le n; \\ & \displaystyle\sum_{j=1}^{n} x_{jr} = 1, & 1 \le r \le n; \\ & x_{jr} \in \{0, 1\}, & 1 \le j \le n, 1 \le r \le n, \end{array} \tag{7.9}$$

Since solving an $n \times n$ linear assignment problem requires $O\left(n^3\right)$ time (see Sect. 4.1.1), we conclude that the following statement holds.

**Theorem 7.1** *Problem $1\left|p_j(r) = p_j g_j(r)\right|\sum C_j$, problem $1\left|p_j(r) = p_j g_j(r)\right|\sum C_j$ and problem $1\left|p_j(r) = p_j g_j(r)\right|\xi C_{\max} + \eta \sum C_j$ under a general positional job-dependent effect (7.1) reduce to the linear assignment problem (7.6), (7.8) and (7.9), respectively. Each of this problems is solvable in $O\left(n^3\right)$ time.*

*Example 7.1* For a single machine scheduling problem, consider the following matrix of possible actual processing times

**Table 7.1** Solutions to the problems in Example 7.1

| Objective function | LAP solution matrix | Optimal permutation | Function value |
|---|---|---|---|
| $C_{\max}$ | $\begin{pmatrix} 0&1&0&0&0\\ 0&0&0&0&1\\ 1&0&0&0&0\\ 0&0&1&0&0\\ 0&0&0&1&0 \end{pmatrix}$ | (3, 1, 4, 5, 2) | 21 |
| $\sum C_j$ | $\begin{pmatrix} 1&0&0&0&0\\ 0&0&0&1&0\\ 0&0&0&0&1\\ 0&1&0&0&0\\ 0&0&1&0&0 \end{pmatrix}$ | (1, 4, 5, 2, 3) | 60 |
| $2C_{\max} + \sum C_j$ | $\begin{pmatrix} 1&0&0&0&0\\ 0&0&0&0&1\\ 0&0&1&0&0\\ 0&1&0&0&0\\ 0&0&0&1&0 \end{pmatrix}$ | (1, 4, 3, 5, 2) | 102 |

$$\mathbf{P} = \begin{pmatrix} 3 & 5 & 6 & 5 & 11\\ 10 & 9 & 7 & 3 & 5\\ 6 & 9 & 8 & 8 & 9\\ 11 & 3 & 3 & 10 & 7\\ 4 & 9 & 6 & 2 & 9 \end{pmatrix}.$$

For scheduling problems with the objective functions $C_{\max}$, $\sum C_j$, and $\xi C_{\max} + \eta \sum C_j$ (with $\xi = 2$ and $\eta = 1$), Table 7.1 presents optimal permutations as well as solutions to the corresponding assignment problems.

## 7.2 Scheduling Independent Jobs Under Job-Independent Positional Effect

For a single machine scheduling problem, a job-independent positional effect (7.2) is defined by an array $g(r)$, $1 \le r \le n$. We refer to the value $g(r)$ as a *positional factor*. Notice that the elements of matrix $\mathbf{P} = \left(p_{jr} = p_j(r)\right)_{n \times n}$ of possible actual processing times can be written as $p_{jr} = p_j g(r)$, $1 \le j \le n$, $1 \le r \le n$.

## 7.2.1 Minimizing Makespan

Let the jobs be processed in accordance with some permutation $\pi = (\pi(1), \dots, \pi(n))$ and a job sequenced in position $\pi(r)$ be associated with a positional factor $g(r)$. Then for problem $1|p_j(r) = p_j g(r)|C_{\max}$, the makespan can be written as

$$C_{\max}(\pi) = \sum_{r=1}^{n} p_{\pi(r)} g(r),$$

which satisfies (7.3) with $W(r) = g(r),\ 1 \le r \le n$, and $\Gamma = 0$. Thus, an optimal schedule can be found by Algorithm Match, and this will take $O(n \log n)$ time.

Notice that here we do not require that the positional factors are monotone. However, as established in Sect. 2.1.1, a permutation that minimizes a linear form (7.3) can be obtained by applying the LPT priority rule (or the SPT priority rule, respectively), provided the sequence $W(r),\ 1 \le r \le n$, is non-decreasing (or non-increasing, respectively).

If the positional factors $g(r),\ 1 \le r \le n$, are non-decreasing, i.e.,

$$g(1) \le g(2) \le \cdots \le g(n), \tag{7.10}$$

then we have a deterioration effect, so that an optimal permutation is obtained by renumbering the jobs in the LPT order (7.4).

On the other hand, if the factors $g(r)$ are non-increasing, i.e.,

$$g(1) \ge g(2) \ge \cdots \ge g(n) \tag{7.11}$$

we have a learning effect, and an optimal permutation is obtained by renumbering the jobs in the SPT order.

The following statement summarizes the status of problem $1|p_j(r) = p_j g(r)|C_{\max}$.

**Theorem 7.2** *Problem $1|p_j(r) = p_j g(r)|C_{\max}$ under a general positional effect (7.2) reduces to minimizing a linear form (7.3) with $W(r) = g(r),\ 1 \le r \le n$, and $\Gamma = 0$, and is solvable in $O(n \log n)$ time by Algorithm Match. In the case of a deterioration effect (7.10), an optimal permutation is obtained in $O(n \log n)$ time by renumbering the jobs in the LPT order. In the case of a learning effect (7.11), an optimal permutation is obtained in $O(n \log n)$ time by renumbering the jobs in the SPT order.*

In particular, the LPT priority rule is optimal for the following two popular deterioration effects: for the *polynomial positional deterioration effect* that is defined by the positional factors

$$g(r) = r^A,\ 1 \le r \le n,\ A > 0, \tag{7.12}$$

and for *exponential positional deterioration effect* that is defined by the positional factors

$$g(r) = \gamma^{r-1},\ 1 \le r \le n,\ \gamma > 1. \tag{7.13}$$

Similarly, the SPT priority rule works for the following two popular learning effects: for the *polynomial positional learning effect* that is defined by the positional factors

$$g(r) = r^A,\ 1 \le r \le n,\ A < 0, \tag{7.14}$$

and for the *exponential positional learning effect* that is defined by the positional factors

$$g(r) = \gamma^{r-1},\ 1 \le r \le n,\ 0 < \gamma < 1. \tag{7.15}$$

### 7.2.2 *Minimizing Total Flow Time*

Let the jobs be processed in accordance with some permutation $\pi = (\pi(1), \dots, \pi(n))$. For the objective function in problem $1\left|p_j(r) = p_j g(r)\right| \sum C_j$ the equality (7.7) holds, where for job $j = \pi(r)$ sequenced in position $r$ of permutation $\pi$, its actual processing time is given by $p_{\pi(r)}(r) = g(r)p_j$. Then, the objective function can be written as

$$F(\pi) = \sum_{j=1}^{n} C_j(\pi) = \sum_{r=1}^{n} g(r)(n - r + 1)p_{\pi(r)}, \tag{7.16}$$

which satisfies (7.3) with $W(r) = (n - r + 1)g(r),\ \ 1 \le r \le n$. Thus, an optimal schedule can be found by Algorithm Match, and this will take $O(n \log n)$ time. As in the case of the makespan, the positional weights do not have to be monotone.

If the factors $g(r)$, $1 \le r \le n$, are non-increasing, i.e., satisfy (7.11), then we have a learning effect. In this case, for any $r$, $1 \le r \le n-1$, we have that $g(r) \ge g(r+1)$ and $n - r + 1 > n - (r+1) + 1$, so that

$$W(1) \ge W(2) \ge \cdots \ge W(n),$$

and an optimal solution is achieved by renumbering the jobs in the SPT order.

Factors $g(r)$, $1 \le r \le n$, that satisfy (7.10) define a deterioration effect. Since for any $r$, $1 \le r \le n-1$, we have that $g(r) \le g(r+1)$, but $n - r + 1 > n - (r+1) + 1$, we cannot guarantee that the positional weights $W(r)$, $1 \le r \le n$, form a monotone sequence. Thus, there is no evidence that a solution to problem $1\left|p_j(r) = p_j g(r)\right| \sum C_j$ with a deterioration effect can be obtained by a priority rule.

It is straightforward to verify that problem $1\left|p_j(r) = p_j g(r)\right|\xi C_{\max} + \eta \sum C_j$ reduces to minimizing the linear form (7.3) with the positional weights $W(r) =$

$(\xi + (n-r+1)\eta)g(r),\ 1 \le r \le n$. Similar to the problem of minimizing total completion time, here in the case of a positional learning effect (7.11), the sequence $W(r)$, $1 \le r \le n$, is non-increasing, so that the solution can be found by the SPT rule. Otherwise, unless $\eta = 0$, the sequence of positional weights need not be monotone, so that an optimal solution can be found by Algorithm Match, but not by a priority rule.

The following statement summarizes the status of problems $1\left|p_j(r) = p_j g(r)\right| \sum C_j$ and $1\left|p_j(r) = p_j g(r)\right|\xi C_{\max} + \eta \sum C_j$.

**Theorem 7.3** *Under a general positional effect (7.2), problems* $1\left|p_j(r) = p_j g(r)\right| \sum C_j$ *and* $1\left|p_j(r) = p_j g(r)\right|\xi C_{\max} + \eta \sum C_j$ *reduce to minimizing a linear form (7.3) with* $\Gamma = 0$ *and with* $W(r) = (n-r+1)g(r),\ 1 \le r \le n$, *and* $W(r) = (\xi + (n-r+1)\eta)g(r),\ 1 \le r \le n$, *respectively. Both problems are solvable in* $O(n \log n)$ *time by Algorithm Match. In the case of a learning effect (7.11), for each of these problems an optimal permutation is obtained in* $O(n \log n)$ *time by renumbering the jobs in the SPT order. In the case of a deterioration effect (7.10), both problems do not admit a* 1*-priority function for an arbitrary non-decreasing array* $g(r)$, $1 \le r \le n$, *of job-independent positional factors unless* $\eta = 0$.

It is worth investigating whether $1\left|p_j(r) = p_j g(r)\right| \sum C_j$ can be solved by a priority rule under additional assumptions regarding positional factors $g(r)$, $1 \le r \le n$, that define a deterioration effect.

Consider first an exponential deterioration effect given by (7.13). As proved below, for some values of $\gamma$, problem $1\left|p_j(r) = p_j g(r)\right| \sum C_j$ is solvable by a priority rule.

**Theorem 7.4** *For problem* $1\left|p_j(r) = p_j g(r)\right| \sum C_j$ *under an exponential positional deterioration effect (7.13), the following holds*

(**a**) *for* $\gamma \ge 2$, $\omega(j) = p_j$, $j \in N$, *is a* 1*-priority function, i.e., the problem is solvable by the LPT rule;*
(**b**) *for* $1 < \gamma < 2$, *no* 1*-priority function exists.*

*Proof* Let $C_{\max}(\pi)$ and $F(\pi)$, respectively, denote the makespan and the sum of the completion times of the jobs sequenced in accordance with a (partial) permutation $\pi$, provided that the sequence $\pi$ starts at time zero. It can be seen from (7.16) that

$$F(\pi) = \sum_{k=1}^{|\pi|} (|\pi| - k + 1) p_{\pi(k)} \gamma^{k-1}. \tag{7.17}$$

Let $\pi = (\pi_1, u, v, \pi_2)$ and $\pi' = (\pi_1, v, u, \pi_2)$ be two permutations of all jobs that only differ in the order of two consecutive jobs $u$ and $v$. Assume that there are $r-1$ jobs in permutation $\pi'$.

It follows that

$$\begin{aligned} F(\pi) = {} & F(\pi_1) + (C_{\max}(\pi_1) + p_u \gamma^{r-1}) + (C_{\max}(\pi_1) + p_u \gamma^{r-1} + p_v \gamma^r) \\ & + |\pi_2|(C_{\max}(\pi_1) + p_u \gamma^{r-1} + p_v \gamma^r) + \gamma^r F(\pi_2) \end{aligned}$$

and by symmetry

$$\begin{aligned}F(\pi') = {} & F(\pi_1) + (C_{\max}(\pi_1) + p_v\gamma^{r-1}) + (C_{\max}(\pi_1) + p_v\gamma^{r-1} + p_u\gamma^r) \\ & +|\pi_2|(C_{\max}(\pi_1) + p_v\gamma^{r-1} + p_u\gamma^r) + \gamma^r F(\pi_2).\end{aligned}$$

Define $\Delta := F(\pi) - F(\pi')$. Then,

$$\begin{aligned}\Delta &= \gamma^{r-1}((2+|\pi_2|)p_u + \gamma p_v(1+|\pi_2|) - (2+|\pi_2|)p_v - \gamma p_u(1+|\pi_2|)) \\ &= \gamma^{r-1}(p_v - p_u)(\gamma - |\pi_2| + |\pi_2|\gamma - 2).\end{aligned}$$

Suppose first that $\gamma \geq 2$, and apply Recipe 2.1 to prove that $\omega(j) = p_j$, $j \in N$, is a 1-priority function; i.e., the problem is solvable by the LPT rule. If $|\pi_2| = 0$, i.e., $u$ and $v$ are the last two jobs in the permutations $\pi$ and $\pi'$, then we conclude that $\Delta \leq 0$ if $p_u \geq p_v$, which corresponds to the LPT rule. If $\gamma = 2$, then $\Delta = 0$, and any order of the jobs $u$ and $v$ is acceptable (including the one implied by the LPT rule).

If $|\pi_2| \geq 1$, then it can be verified that the function $\gamma - |\pi_2| + |\pi_2|\gamma - 2$ under the constraint $\gamma \geq 2$ stays positive. In fact, it reaches its minimum value of 1 for $\gamma = 2$ and $|\pi_2| = 1$, which again proves optimality of the LPT rule.

Assume now that $1 < \gamma < 2$. Notice that $\gamma - |\pi_2| + |\pi_2|\gamma - 2 = 0$ if $|\pi_2| = (2 - \gamma)/(\gamma - 1)$. Therefore, for any $\gamma$, $1 < \gamma < 2$, there exist two consecutive integer values of $|\pi_2|$ such that $\gamma - |\pi_2| + |\pi_2|\gamma - 2$ changes its sign. Due to Recipe 2.2, this implies that the sign of $\Delta$ does not stay constant, and no 1-priority function exists. □

We now pass to problem $1\left|p_j(r) = p_j g(r)\right|\sum C_j$ under a polynomial deterioration effect given by (7.12). Similar to its counterpart with an exponential deterioration effect, our purpose is to find out for which range of $A$ the problem admits a solution in terms by a priority rule, either LPT or SPT.

For this model, the array of positional weights is given by

$$W(r) = (n - r + 1)r^A, \ 1 \leq r \leq n. \tag{7.18}$$

Considering $W(r)$ as a function of a continuous positive argument, observe that

$$\frac{d}{dr}(n - r + 1)r^A = r^{A-1}(A - r + An - Ar),$$

so that it has a single maximum $r_{\max} = \frac{A(n+1)}{A+1}$. This means that the function increases for $r \in (0, r_{\max})$ and decreases for $r > r_{\max}$.

The consideration above implies that sequence $W(r)$ of the form (7.18) in general consists of an increasing subsequence followed by a decreasing subsequence. A sequence of this structure is often called a $\Lambda$-shaped sequence (see Sect. 5.1).

Notice that an optimal permutation for problem $1\left|p_j(r) = p_j g(r)\right|\sum C_j$ under a polynomial positional deterioration effect (7.12) can be found by Algorithm Match, which orders the jobs in such a way that smaller processing times are associated with larger positional factors (7.18). This means that an optimal permutation of jobs consists of a subsequence of jobs taken in non-increasing order of $p_j$ 's, followed by a subsequence of jobs taken in non-decreasing order of $p_j$'s. A sequence of this structure is often called a $V$-shaped sequence (see Sect. 5.1).

For problem $1\left|p_j(r) = p_j g(r)\right|\sum C_j$ under a polynomial positional deterioration effect (7.12) to be solvable by a priority rule, either SPT or LPT, we need that the sequence (7.18) is either monotone increasing or monotone decreasing, respectively.

**Lemma 7.1** *Problem $1\left|p_j(r) = p_j g(r)\right|\sum C_j$ under a positional polynomial deterioration effect (7.12) is solvable by the SPT rule if $A < \log_2\left(\frac{n}{n-1}\right)$, and by the LPT rule if $A > \frac{1}{\log_2\left(\frac{n}{n-1}\right)}$. Otherwise, an optimal permutation is $V$-shaped.*

*Proof* As demonstrated above, sequence (7.18) of positional weights is in general $\Lambda$-shaped, and an optimal permutation found by Algorithm Match must be $V$-shaped.

If $W(r_0) > W(r_0 + 1)$ for some value of $r_0$, $1 \le r_0 \le n$, then the subsequence $W(r)$, $r_0 \le r \le n$, is decreasing. Thus, if $r_0 = 1$, i.e., $W(1) > W(2)$, then the whole sequence (7.18) is decreasing. From $W(1) = n$ and $W(2) = (n-1)2^A$, the inequality $n > (n-1)2^A$ implies that $A < \log_2\left(\frac{n}{n-1}\right)$. Thus, under the latter condition, Algorithm Match will sort the jobs in non-decreasing order of normal processing times, i.e., in accordance with the SPT rule.

Similarly, if $W(n-1) < W(n)$, then the whole sequence (7.18) is increasing. From $W(n-1) = 2(n-1)^A$ and $W(n) = n^A$, the inequality $2(n-1)^A < n^A$ implies that $A > \frac{1}{\log_2\left(\frac{n}{n-1}\right)}$. In an optimal permutation delivered by Algorithm Match, the jobs are placed in non-increasing order of normal processing times, i.e., in accordance with the LPT rule. □

*Example 7.2* Consider problem $1\left|p_j(r) = p_j g(r)\right|\sum C_j$ under a polynomial positional deterioration effect (7.12) with $n = 7$ jobs. By Lemma 7.1, the largest $A$ that guarantees that the positional weights (7.18) form a decreasing sequence is equal to $\log_2\left(\frac{7}{6}\right) = 0.222$, while the smallest $A$ for which the array of positional weights (7.18) forms an increasing sequence is equal to 4.496. Table 7.2 shows the positional weights computed for $A = 0.2$ and $A = 4.5$. Clearly, the positional weights $W(r)$ are suitably sorted. All reported computations are accurate up to three decimal places.

We conclude this section by presenting Table 7.3 that summarizes the results for problems $1\left|p_j(r) = p_j g(r)\right|\Phi$ with $\Phi \in \left\{C_{\max}, \sum C_j, \xi C_{\max} + \eta \sum C_j\right\}$.

**Table 7.2** Positional factors for Example 7.2

| $A$ | $W(1)$ | $W(2)$ | $W(3)$ | $W(4)$ | $W(5)$ | $W(6)$ | $W(7)$ |
|---|---|---|---|---|---|---|---|
| 0.2 | 7 | 6.892 | 6.229 | 5.278 | 4.139 | 2.278 | 1.476 |
| 4.5 | 7 | 135.765 | 701.481 | 2048.000 | 4192.667 | 6349.077 | 6352.449 |

**Table 7.3** Solution algorithms for single machine problems with a job-independent positional effect

| $\Phi$ | Factors $g(r)$ | Algorithm |
|---|---|---|
| $C_{\max}$ | arbitrary | Match |
| $C_{\max}$ | (7.10) | LPT |
| $C_{\max}$ | (7.11) | SPT |
| $\sum C_j$ | arbitrary | Match |
| $\xi C_{\max} + \eta \sum C_j$ | arbitrary | Match |
| $\sum C_j$ | (7.11) | SPT |
| $\xi C_{\max} + \eta \sum C_j$ | (7.11) | SPT |
| $\sum C_j$ | (7.12), $A < \log_2\left(\frac{n}{n-1}\right)$ | LPT |
| $\sum C_j$ | (7.12), $A > \log_2^{-1}\left(\frac{n}{n-1}\right)$ | SPT |
| $\sum C_j$ | (7.13), $\gamma \geq 2$ | LPT |

## 7.3 Scheduling with Series-Parallel Precedence Constraints Under Positional Effects

In this section, we consider single machine problems to minimize an objective function $\Phi \in \left\{C_{\max}, \sum C_j\right\}$ with specific job-independent positional effects. Unlike in Sect. 7.2, here we assume that the jobs of set $N$ are not independent and a precedence relation given by a series-parallel reduction graph $G = (N, U)$ is imposed over the set $N$ of jobs. The purpose of this section is to investigate whether for some of the scheduling problems generically denoted by $1\left|p_j(r) = p_j g(r), SP - prec\right|\Phi$, the objective function is priority-generating, so that the corresponding problem is solvable in $O(n \log n)$ time. See Chap. 3 for definitions and main results on scheduling under precedence constraints, including the concept of a priority-generating objective function and a priority function for partial permutations, as well as Recipes 3.1 and 3.2 that contain recommendations how to prove or disprove that an objective function is priority-generating.

Recall that it follows from Chap. 3 that for an objective function to be priority-generating, the existence of a 1-priority function $\omega(j)$, $j \in N$, is necessary, so that the best permutation among all permutations of jobs of set $N$ can be found by sorting the jobs in non-increasing order of their 1-priorities. In particular, if a problem is solvable by the LPT rule, then it admits a 1-priority function $\omega(j) = p_j$, $j \in N$, while if the SPT rule finds an optimal permutation, then the problem admits a 1-priority function $\omega(j) = 1/p_j$, $j \in N$. As seen from Sect. 7.2, among the candidate problems that might have a priority-generating objective function are problems $1\left|p_j(r) = p_j g(r)\right|C_{\max}$ under either a positional deterioration effect (7.10) or a positional learning effect (7.11), as well as problem $1\left|p_j(r) = p_j g(r)\right|\sum C_j$ under a positional learning effect (7.11).

It appears that the fact that the array $g(r)$, $1 \le r \le n$, is ordered is not enough to guarantee that an objective function is priority-generating. Instead, we provide analysis for two popular types of monotone positional effects, namely polynomial and exponential, establishing both positive and negative results regarding priority-generating objective functions.

Given a scheduling problem with a positional job-independent effect, let $\pi$ be a (partial) permutation of jobs contained as a subsequence in some schedule. Assume that (i) the first job in this permutation is sequenced in the $r$th position in an overall schedule and (ii) this first job starts at time $t \ge 0$. Under these assumptions, let $C_{\max}(\pi; t; r)$ denote the maximum completion time of the jobs in $\pi$.

## 7.3.1 *Exponential Positional Effect*

In this subsection, we consider the single machine model in which the processing time of a job depends exponentially on the position in which it is scheduled, i.e.,

$$g(r) = \gamma^{r-1},\ 1 \le r \le n, \tag{7.19}$$

where $\gamma$ is a positive constant, not equal to 1.

We start with the problem of minimizing the makespan under a positional effect (7.19), without making any assumption on value of $\gamma$; i.e., we do not distinguish between a deterioration effect ($\gamma > 1$) or a learning effect ($0 < \gamma < 1$).

It immediately follows from (7.19) that for an arbitrary permutation $\pi$, we have that

$$C_{\max}(\pi; 0; 1) = \sum_{j=1}^{|\pi|} p_{\pi(j)}\gamma^{j-1},$$

and, if the first job in $\pi$ is sequenced in the $r$th position and starts at time $t$ in an overall schedule, we deduce that

$$C_{\max}(\pi; t; r) = t + \gamma^{r-1} C_{\max}(\pi; 0; 1).$$

Let $\pi^{\alpha\beta} = (\pi_1 \alpha\beta\pi_2)$ and $\pi^{\beta\alpha} = (\pi_1\beta\alpha\pi_2)$ be two permutations of all jobs that only differ in the order of the subsequences $\alpha = (\alpha(1), \dots, \alpha(u))$ of $u$ jobs and $\beta = (\beta(1), \dots, \beta(v))$ of $v$ jobs.

Define

$$\Delta := C_{\max}(\pi^{\alpha\beta}) - C_{\max}(\pi^{\beta\alpha}) = C_{\max}(\pi^{\alpha\beta}; 0; 1) - C_{\max}(\pi^{\beta\alpha}; 0; 1). \tag{7.20}$$

In accordance with Recipe 3.1, we need to determine a sufficient condition for the inequality $\Delta \le 0$ to hold. The form of such a condition should be suitable for defining a priority function.

Assume that there are $r-1$ jobs in permutation $\pi_1$. It follows that $C_{\max}(\pi_1\alpha\beta\pi_2; 0; 1) = C_{\max}(\pi_1; 0; 1) + C_{\max}(\alpha\beta\pi_2; 0; r)$ and $C_{\max}(\pi_1\beta\alpha\pi_2; 0; 1) = C_{\max}(\pi_1; 0; 1) + C_{\max}(\beta\alpha\pi_2; 0; r)$, so that

$$\Delta = C_{\max}(\alpha\beta\pi_2; 0; r) - C_{\max}(\beta\alpha\pi_2; 0; r).$$

Further, we deduce that

$$\begin{aligned}
&C_{\max}(\alpha\beta\pi_2; 0; r) - C_{\max}(\beta\alpha\pi_2; 0; r)\\
&= C_{\max}(\pi_2; C_{\max}(\alpha\beta; 0; r); r+u+v) - C_{\max}(\pi_2; C_{\max}(\beta\alpha; 0; r); r+u+v)\\
&= \left(C_{\max}(\alpha\beta; 0; r) + \gamma^{r+u+v-1}\sum_{j=1}^{|\pi_2|} p_{\pi(j)}\gamma^{j-1}\right)\\
&\quad -\left(C_{\max}(\beta\alpha; 0; r) + \gamma^{r+u+v-1}\sum_{j=1}^{|\pi_2|} p_{\pi(j)}\gamma^{j-1}\right)\\
&= C_{\max}(\alpha\beta; 0; r) - C_{\max}(\beta\alpha; 0; r) = \gamma^{r-1}C_{\max}(\alpha\beta; 0; 1) - \gamma^{r-1}C_{\max}(\beta\alpha; 0; 1).
\end{aligned}$$

Thus, the sign of $\Delta$ depends on the sign of the difference $C_{\max}(\alpha\beta; 0; 1) - C_{\max}(\beta\alpha; 0; 1)$. We deduce

$$\begin{aligned}
&C_{\max}(\alpha\beta; 0; 1) - C_{\max}(\beta\alpha; 0; 1)\\
&= C_{\max}(\beta; C_{\max}(\alpha; 0; 1); u+1) - C_{\max}(\alpha; C_{\max}(\beta; 0; 1); v+1)\\
&= \left(C_{\max}(\alpha; 0; 1) + \gamma^{u}C_{\max}(\beta; 0; 1)\right) - \left(C_{\max}(\beta; 0; 1) + \gamma^{v}C_{\max}(\alpha; 0; 1)\right)\\
&= C_{\max}(\beta; 0; 1)\left(\gamma^{u}-1\right) - C_{\max}(\alpha; 0; 1)(\gamma^{v}-1).
\end{aligned}$$

Therefore, $\Delta \le 0$ if

$$C_{\max}(\beta; 0; 1)\left(\gamma^{u}-1\right) \le C_{\max}(\alpha; 0; 1)(\gamma^{v}-1), \tag{7.21}$$

and the following statement holds.

**Theorem 7.5** *For problem $1\left|p_j(r) = p_j g(r)\right|C_{\max}$ under an exponential positional effect (7.19), the objective function is priority-generating and*

$$\omega(\pi) = \frac{C_{\max}(\pi; 0; 1)}{\gamma^{|\pi|}-1} = \frac{\sum_{j=1}^{|\pi|} p_{\pi(j)}\gamma^{j-1}}{\gamma^{|\pi|}-1} \tag{7.22}$$

*is its priority function. Problem $1\left|p_j(r) = p_j g(r), SP-prec\right|C_{\max}$ is solvable in $O(n\log n)$ time.*

*Proof* Dividing both sides of (7.21) by $(\gamma^{u}-1)(\gamma^{v}-1)$, which is positive for all positive $\gamma$ not equal to 1, we derive that $\Delta \le 0$, provided that

$$\frac{C_{\max}(\alpha; 0; 1)}{\gamma^u - 1} \geq \frac{C_{\max}(\beta; 0; 1)}{\gamma^v - 1}.$$

For an arbitrary (partial) permutation $\pi$, define the function $\omega(\pi)$ by (7.22), since $C_{\max}(\pi; 0; 1) = \sum_{j=1}^{|\pi|} p_{\pi(j)}\gamma^{j-1}$. It is easily verified that $\omega(\alpha) > \omega(\beta)$ implies $C_{\max}(\pi^{\alpha\beta}) \leq C_{\max}(\pi^{\beta\alpha})$, while $\omega(\alpha) = \omega(\beta)$ implies $C_{\max}(\pi^{\alpha\beta}) = C_{\max}(\pi^{\beta\alpha})$, as required by Definition 3.2. □

Notice that if (7.22) is applied to a single job $j$, i.e., to a permutation of length one, then the right-hand side of (7.22) boils down to $p_j/(\gamma - 1)$. This correlates well with Theorem 7.2, which implies that for problem $1\left|p_j(r) = p_j g(r)\right|C_{\max}$ under an exponential positional effect (7.19):

- For $0 < \gamma < 1$ (learning), $\omega(j) = -p_j$ is a 1-priority function, and SPT is an optimal priority rule.
- For $\gamma > 1$ (deterioration), $\omega(j) = p_j$ is a 1-priority function, and LPT is an optimal priority rule.

Notice that for a slightly more general effect

$$p_j(r) = p_j\gamma^{r-1} + q_j,$$

the makespan remains priority-generating and

$$\omega(\pi) = \frac{\sum_{j=1}^{|\pi|} p_{\pi(j)}\gamma^{j-1}}{\gamma^{|\pi|} - 1}$$

remains a priority function.

We now pass to consideration of the single machine problem to minimize the sum of the completion times $\sum C_j$ under an exponential positional effect (7.19).

Problem $1\left|p_j(r) = p_j g(r)\right|\sum C_j$ under an exponential positional effect (7.19) due to Theorem 7.3 is solvable by the SPT rule in the case of learning $(0 < \gamma < 1)$, while by Theorem 7.4, it is solvable by the LPT rule in the case of fast deterioration $(\gamma \geq 2)$. These facts, however, do not imply that the objective function is priority-generating for the relevant values of $\gamma$. We demonstrate this below for two particular values of $\gamma$.

Recall from Sect. 3.2 that in accordance with Recipe 3.2, in order to disprove that an objective function $\Phi$ is priority-generating, the following approach can be employed. An instance of the problem should be exhibited such that $\Phi(\pi^{\alpha\beta}) < \Phi(\pi^{\beta\alpha})$ for some permutations $\pi^{\alpha\beta} = (\pi_1\alpha\beta\pi_2)$ and $\pi^{\beta\alpha} = (\pi_1\beta\alpha\pi_2)$, while $\Phi(\varphi^{\alpha\beta}) > \Phi(\varphi^{\beta\alpha})$ for some other permutations $\varphi^{\alpha\beta} = (\varphi'\alpha\beta\varphi'')$ and $\varphi^{\beta\alpha} = (\varphi'\beta\alpha\varphi'')$.

All counterexamples in the lemma below and the lemmas in the following subsection have a similar structure: There are four jobs, and permutations $\alpha = (1, 2)$,

$\beta = (3)$ and $\pi_2 = (4)$ are selected. In the proofs, we compare two pairs of permutations: $\pi^{\alpha\beta} = (\alpha\beta, 4) = (1, 2, 3, 4)$ and $\pi^{\beta\alpha} = (\beta\alpha, 4) = (3, 1, 2, 4)$, as well as $\varphi^{\alpha\beta} = (4, \alpha\beta) = (4, 1, 2, 3)$ and $\varphi^{\beta\alpha} = (4, \beta\alpha) = (4, 3, 1, 2)$.

**Lemma 7.2** *For problem $1\left|p_j(r) = p_j g(r)\right|\sum C_j$ under an exponential positional effect (7.19) with $\gamma = 2$ and $\gamma = \frac{1}{2}$ the objective function is not priority-generating.*

*Proof* For $\gamma = 2$, consider the following instance of the problem in question. Take four jobs with the normal processing times

$$p_1 = 4,\ p_2 = 1,\ p_3 = 3,\ p_4 = 1$$

and select permutations $\alpha = (1, 2)$, $\beta = (3)$. Let $F(\pi)$ denote the sum of the completion times of the jobs sequenced in accordance with a permutation $\pi$, defined by (7.17). Comparing permutations $\pi^{\alpha\beta} = (\alpha\beta, 4)$ and $\pi^{\beta\alpha} = (\beta\alpha, 4)$, we see that

$$\begin{aligned} F(\alpha\beta, 4) &= 4 \times (4 \times 1) + 3 \times \left(1 \times 2^1\right) + 2 \times \left(3 \times 2^2\right) + 1 \times 2^3 = 54 \\ &> F(\beta\alpha, 4) = 4 \times (3 \times 1) + 3 \times \left(4 \times 2^1\right) + 2 \times \left(1 \times 2^2\right) + 1 \times 2^3 \end{aligned}$$

On the other hand, comparing permutations $\varphi^{\alpha\beta} = (4, \alpha\beta)$ and $\varphi^{\beta\alpha} = (4, \beta\alpha)$, we compute

$$F(4, \alpha\beta) = 60 < F(4, \beta\alpha) = 62.$$

For $\gamma = \frac{1}{2}$, consider the four-job instance with the normal processing times

$$p_1 = 19,\ p_2 = 1,\ p_3 = 14,\ p_4 = 1$$

and select permutations $\alpha = (1, 2)$, $\beta = (3)$. We see that $F(\alpha\beta, 4) = 84\frac{5}{8} < F(\beta\alpha, 4) = 85\frac{1}{8}$, but $F(4, \alpha\beta) = 34\frac{3}{4} > C_{\max}(4, \beta\alpha) = 34\frac{5}{8}$.

This proves the lemma. □

## 7.3.2 *Polynomial Positional Effect*

We start this subsection with analyzing problem $1\left|p_j(r) = p_j g(r)\right|C_{\max}$, where

$$g(r) = r^A,\ 1 \le r \le n. \tag{7.23}$$

In the beginning, we do not make any assumption on the sign of $A$, i.e., do not distinguish between learning and deterioration. On the other hand, we limit our consideration to integer values of $A$.

Similar to Sect. 7.3.1, below we also manipulate values $C_{\max}(\pi; t; r)$. It immediately follows from (7.23) that for an arbitrary permutation $\pi$, we have that

$$C_{\max}(\pi; 0; 1) = \sum_{j=1}^{|\pi|} p_{\pi(j)} j^A;$$

$$C_{\max}(\pi; 0; r) = \sum_{j=1}^{|\pi|} p_{\pi(j)} (r + j - 1)^A;$$

$$C_{\max}(\pi; t; r) = t + \sum_{j=1}^{|\pi|} p_{\pi(j)} (r + j - 1)^A = t + C_{\max}(\pi; 0; r). \quad (7.24)$$

Let $\pi^{\alpha\beta} = (\pi_1 \alpha\beta\pi_2)$ and $\pi^{\beta\alpha} = (\pi_1 \beta\alpha\pi_2)$ be two permutations of all jobs that only differ in the order of the subsequences $\alpha$ (of $u$ jobs) and $\beta$ (of $v$ jobs). Define $\Delta$ by (7.20).

Following Recipe 3.1, in order to verify that in the case under consideration, the objective function is priority-generating, we need to determine a sufficient condition for the inequality $\Delta \le 0$ to hold.

Assume that thereare $r - 1$ jobs in permutation $\pi_1$ and denote $C_{\max}(\pi_1; 0; 1)$ by $t'$. Since $C_{\max}(\pi_1\alpha\beta\pi_2; 0; 1) = C_{\max}(\alpha\beta\pi_2; t'; r)$, it follows from (7.24) that $C_{\max}(\pi_1\alpha\beta\pi_2; 0; 1) = t' + C_{\max}(\alpha\beta\pi_2; 0; r)$; similarly, $C_{\max}(\pi_1\beta\alpha\pi_2; 0; 1) = t' + C_{\max}(\beta\alpha\pi_2; 0; r)$, so that

$$\Delta = C_{\max}(\alpha\beta\pi_2; 0; r) - C_{\max}(\beta\alpha\pi_2; 0; r).$$

Besides, $C_{\max}(\alpha\beta\pi_2; 0; r) = C_{\max}(\alpha\beta; 0; r) + C_{\max}(\pi_2; 0; r + u + v)$ and $C_{\max}(\beta\alpha\pi_2; 0; r) = C_{\max}(\beta\alpha; 0; r) + C_{\max}(\pi_2; 0; r + v + u)$, so that

$$\Delta = C_{\max}(\alpha\beta; 0; r) - C_{\max}(\beta\alpha; 0; r).$$

We obtain

$$C_{\max}(\alpha\beta; 0; r) = \sum_{i=1}^{u} p_{\alpha(i)} (r + i - 1)^A + \sum_{j=1}^{v} p_{\beta(j)} (r + u + j - 1)^A;$$

$$C_{\max}(\beta\alpha; 0; r) = \sum_{j=1}^{v} p_{\beta(j)} (r + j - 1)^A + \sum_{i=1}^{u} p_{\alpha(i)} (r + v + i - 1)^A,$$

so that

$$\Delta = \sum_{i=1}^{u} p_{\alpha(i)} \big( (r + i - 1)^A - (r + v + i - 1)^A \big) \quad (7.25)$$
$$+ \sum_{j=1}^{v} p_{\beta(j)} \big( (r + u + j - 1)^A - (r + j - 1)^A \big).$$

For $A = 1$, (7.25) simplifies to

$$\Delta = -v \sum_{i=1}^{u} p_{\alpha(i)} + u \sum_{j=1}^{v} p_{\beta(j)}. \tag{7.26}$$

and the following statement holds.

**Theorem 7.6** *For problem $1\left|p_j(r) = p_j g(r)\right|C_{\max}$ under the deterioration effect (7.12) with $A = 1$, the objective function is priority-generating and*

$$\omega(\pi) = \frac{\sum_{j=1}^{|\pi|} p_{\pi(j)}}{|\pi|} \tag{7.27}$$

*is its priority function. Under these conditions problem $1\left|p_j(r) = p_j g(r),\right.$ $SP - prec|C_{\max}$ is solvable in $O(n \log n)$ time.*

*Proof* Taking (7.26) and dividing the left-hand by $uv$, we deduce that $\Delta \le 0$, provided that

$$\frac{\sum_{i=1}^{u} p_{\alpha(i)}}{u} \ge \frac{\sum_{j=1}^{v} p_{\beta(j)}}{v}.$$

For an arbitrary (partial) permutation $\pi$, define the function $\omega(\pi)$ by (7.27). It is easily verified that $\omega(\alpha) > \omega(\beta)$ implies $C_{\max}(\pi^{\alpha\beta}) \le C_{\max}(\pi^{\beta\alpha})$, while $\omega(\alpha) = \omega(\beta)$ implies $C_{\max}(\pi^{\alpha\beta}) = C_{\max}(\pi^{\beta\alpha})$, as required by Definition 3.2. $\square$

The meaning of $\omega(\pi)$ of the form (7.27) is the average normal processing time of the jobs in sequence $\pi$, which correlates well with the fact that $w(j) = p_j$ is a 1-priority function for the problem.

The existence of the priority function for the case above is due to the fact that for determining the sign of the difference $\Delta$, we are able to (i) remove all parameters that are not related to permutations $\alpha$ and $\beta$, and (ii) to separate the parameters associated with permutation $\alpha$ from those related to $\beta$.

It is unlikely that a priority function exists for other (integer) values of $A$. The exhibited counterexamples in the statements below follow the same pattern as in Sect. 7.3.1.

The lemma below demonstrates that there is no priority function for $A = 2$.

**Lemma 7.3** *For problem $1\left|p_j(r) = p_j g(r)\right|C_{\max}$ under the deterioration effect (7.12) with $A = 2$ the objective function is not priority-generating.*

*Proof* Consider the following instance of the problem in question. Take four jobs with the normal processing times

$$p_1 = 1, \ p_2 = 6, \ p_3 = 4, \text{and } p_4 = 1.$$

and select permutations $\alpha = (1, 2), \beta = (3)$. Comparing permutations $\pi^{\alpha\beta} = (\alpha\beta, 4)$ and $\pi^{\beta\alpha} = (\beta\alpha, 4)$, we see that

$$C_{\max}(\alpha\beta, 4) = 1 + 6 \times 2^2 + 4 \times 3^2 + 1 \times 4^2 = 77$$
$$< C_{\max}(\beta\alpha, 4) = 4 + 1 \times 2^2 + 6 \times 3^2 + 1 \times 4^2 = 78.$$

On the other hand, comparing permutations $\varphi^{\alpha\beta} = (4, \alpha\beta)$ and $\varphi^{\beta\alpha} = (4, \beta\alpha)$, we compute

$$C_{\max}(4, \alpha\beta) = 123 > C_{\max}(4, \beta\alpha) = 122,$$

which proves the lemma. □

Lemma 7.3 can be extended to the problem under the deterioration effect (7.12) with integer $A > 2$.

Now we consider problem $1\big|, p_j(r) = p_j g(r)\big| \sum C_j$ under the deterioration effect (7.12) for integer $A$. Recall that Corollary 3.1 states that for a scheduling model, the fact that the makespan is not a priority-generating function implies that neither the total completion time is a priority-generating function. Due to Lemma 7.3, we only need to look at the case of $A = 1$. However, Lemma 7.1 states that an optimal permutation for $A = 1$ is $V$-shaped; i.e., the problem does not admit a 1-priority function. This means that for $A = 1$, the objective function is not priority-generating.

We now turn to the models with a learning effect (7.14). Unlike for its deterioration counterpart, where at least one particular effect is associated with a priority-generating function, a learning effect does not lead to a priority-generating objective function, for both objectives, the makespan and the total completion time.

**Lemma 7.4** *For problem $1\big|p_j(r) = p_j g(r)\big|C_{\max}$ under the learning effect (7.14) with $A = -1$ and $A - 2$ the objective function is not priority-generating.*

*Proof* For $A = -1$, consider the following instance of the problem in question. Take four jobs with the normal processing times

$$p_1 = 12,\ \ p_2 = 96,\ \ p_3 = 36, \text{and}\ p_4 = 4$$

and select permutations $\alpha = (1, 2)$ and $\beta = (3)$. We see that $C_{\max}(\alpha\beta, 4) = 73 < C_{\max}(\beta\alpha, 4) = 75$, but $C_{\max}(4, \alpha\beta) = 51 > C_{\max}(4, \beta\alpha) = 50$.

For $A = -2$, consider the four-job instance with the normal processing times

$$p_1 = 36,\ \ p_2 = 252,\ \ p_3 = 72,\ \ p_4 = 4$$

and select permutations $\alpha = (1, 2)$ and $\beta = (3)$. We see that $C_{\max}(\alpha\beta, 4) = 107\frac{1}{4} < C_{\max}(\beta\alpha, 4) = 109\frac{1}{4}$, but $C_{\max}(4, \alpha\beta) = 45\frac{1}{2} > C_{\max}(4, \beta\alpha) = 41\frac{3}{4}$.

This proves the lemma. □

Due to Corollary 3.1 and Lemma 7.4, we deduce that for problem $1\big|p_j(r) = p_j g(r)\big| \sum C_j$ under the learning effect (7.14) with $A = -1$ and $A - 2$, the objective function is not priority-generating.

## 7.4 Bibliographic Notes

In this section, we only review the publications relevant to the content of this chapter. The problems with parallel machines under positional effects are discussed in Chap. 11. For other relevant models and problems with other objective functions, the reader is referred to focused surveys Janiak and Rudek (2006) and Rustogi and Strusevich (2012b).

Among the first papers that consider scheduling problems with a general position effect (7.1) are Bachman and Janiak (2004) and Mosheiov and Sarig (2009). However, reductions to a full form linear assignment problems similar to those presented in Sect. 7.1 appear in Biskup (1999) and Mosheiov and Sidney (2003). Although the authors of the two latter papers consider problems with a learning effect, the assumption on learning in not used in the reduction.

Research on scheduling problems with positional effects conducted before the critical review Rustogi and Strusevich (2012b) has had several limitations. First, earlier authors focused on monotone effects only, such as learning and deterioration; moreover, these two types of effects have been considered separately despite their similarities. Second, assumptions on the exact shape of positional factors have been made (i.e., polynomial or exponential), despite the fact that many results would hold for an arbitrary array of positional factors. Third, the choice of solution approaches has included only simple priority rules, such as either the LPT or the SPT, and a reduction to a full form of the linear assignment problem, while a possible use of Algorithm Match has been neglected.

Surprisingly, the fact that problem $1\left|p_j(r) = p_j g(r)\right|C_{\max}$ for an arbitrary array $g(r)$, $1 \le r \le n$, of positional factors is solvable in $O(n \log n)$ time has been established only in Rustogi and Strusevich (2012b). The same can be said about optimality of the LPT rule for the a general positional deterioration effect (7.10) and about optimality of the SPT rule for a general positional learning effect (7.11). For special cases of the general position deterioration effect, optimality of the LPT rule has been established earlier: for the polynomial deterioration (7.12) by Mosheiov (2005) and for the exponential deterioration (7.13) by Gordon et al. (2008). Similarly, optimality of the SPT rule has been established for the polynomial learning (7.14) by Mosheiov (2001) and for the exponential learning (7.15) by Gordon et al. (2008).

Notice that some authors study scheduling problems with alternative forms of position-dependent processing times. For example, Bachman and Janiak (2004) consider a single machine problem to minimize the makespan in which the processing time of job $j$ scheduled in position $r$ is given by $p_j(r) = A_j + b_j r$, where $A_j$ is the normal processing time and $b_j$ is a job-dependent rate (positive in the case of deterioration and negative in the case of learning). As shown in Rustogi and Strusevich (2012b), even for a more general situation, e.g., when the actual time of job $j$ scheduled in position $r$ is defined by $p_j(r) = A_j\left(a_j + b_j g(r)\right)$, for an arbitrary permutation $\pi$ of jobs, the makespan can be written as

$$C_{\max}(\pi) = \sum_{r=1}^{n} A_{\pi(r)} a_{\pi(r)} + \sum_{r=1}^{n} A_{\pi(r)} b_{\pi(r)} g(r),$$

which satisfies (7.3) with $W(r) = g(r)$, $1 \le r \le n$, $p_j = A_j b_j$, $j \in N$, and $\Gamma = \sum_{j=1}^{n} A_j a_j$, so that the problem of minimizing the makespan is solvable by Algorithm Match in $O(n \log n)$ time.

Rustogi and Strusevich (2012a, b) solve problem $1\left|p_j(r) = p_j g(r)\right| \sum C_j$ for an arbitrary array $g(r)$, $1 \le r \le n$, of positional factors by Algorithm Match in $O(n \log n)$ time. They also prove optimality of the SPT rule under a general positional learning effect (7.11). Optimality of the SPT rule for the case of a polynomial learning effect (7.14) has been proved by Biskup (1999) and for an exponential learning effect (7.15) by Gordon et al. (2008). Surprisingly, for problem $1\left|p_j(r) = p_j g(r)\right| \sum C_j$ with a deterioration effect (7.10), no polynomial algorithm faster than $O\left(n^3\right)$ time has been known prior to Rustogi and Strusevich (2012a, b). Theorem 7.4 which resolves the issue of the existence of a 1-priority function for the problem under an exponential positional effect (7.13) is proved in Gordon et al. (2008). Mosheiov (2005) demonstrates that for the problem with a polynomial deterioration effect defined by (7.12), an optimal permutation is $V$-shaped; his proof is different from the reasoning given in Sect. 7.2.2. Yang and Yang (2010) claim that the latter problem can be solved by a technique used by Biskup (1999), which, however, cannot be transferred to deterioration.

As mentioned above, the single machine problem to minimize the makespan with the effect $p_j(r) = A + b_j r$, introduced by Bachman and Janiak (2004), is solvable in $O(n \log n)$ time, even in more general settings. By contrast, if the objective is the total completion time, the problem does not reduce to minimizing (7.3) and it is essential to use a full form linear assignment problem for its solution (see Yang and Yang (2010)).

The material of Sect. 7.3 is based on paper Gordon et al. (2008).

Regarding related objective functions, notice that Mosheiov (2001) shows that problem $1\left|p_j(r) = p_j g(r)\right| \sum w_j C_j$ under a polynomial learning effect (7.14) cannot be solved by the WSPT rule, even for a two-job instance. The exact status of the problem remains unresolved.

## References

Bachman A, Janiak A (2004) Scheduling jobs with position-dependent processing times. J Oper Res Soc 55:257–264

Biskup D (1999) Single-machine scheduling with learning considerations. Eur J Oper Res 11:173–178

Gordon VS, Potts CN, Strusevich VA, Whitehead JD (2008) Single machine scheduling models with deterioration and learning: handling precedence constraints via priority generation. J Sched 11:357–370

Janiak A, Rudek R (2006) Scheduling problems with position-dependent processing times. In: Janiak A (ed) Scheduling in computer and manufacturing systems. Wydawnictwa Komuniikacji i Łączności, Warsaw, pp 26–38

Mosheiov G (2001) Scheduling problems with a learning effect. Eur J Oper Res 132:687–693

Mosheiov G (2005) A note on scheduling deteriorating jobs. Math Comput Model 41:883–886

Mosheiov G, Sarig A (2009) Scheduling a maintenance activity and due-window assignment on a single machine. Comp Oper Res 36:2541–2545

Mosheiov G, Sidney JB (2003) Scheduling with general job-dependent learning curves. Eur J Oper Res 147:665–670

Rustogi K, Strusevich VA (2012a) Single machine scheduling with general positional deterioration and rate-modifying maintenance. Omega 40:791–804

Rustogi K, Strusevich VA (2012b) Simple matching vs linear assignment in scheduling models with positional effects: A critical review. Eur J Oper Res 22:393–407

Yang SJ, Yang DL (2010) Minimizing the total completion time in single-machine scheduling with ageing/deteriorating effects and deteriorating maintenance activities. Comp Math Appl 60:2161–2169

# Chapter 8
# Scheduling with Pure and Combined Additive Start-Time-Dependent Effects

In this chapter, we study the problems of minimizing the makespan and the total completion time on a single machine, provided that the actual processing times of the jobs are subject to a special form of a start-time-dependent effect. We also study effects in which such a start-time-dependent effect is combined with a positional effect.

For a job $j \in N = \{1, 2, \ldots, n\}$, its normal processing time $p_j$ is given. Suppose that the jobs are processed on a single machine in accordance with a permutation $\pi = (\pi(1), \ldots, \pi(n))$. As defined in Sect. 6.2, if the actual processing time of a job depends on its normal processing time and its start time in the schedule, we call such an effect start-time-dependent.

Following Sect. 6.2, we distinguish between two types of a start-time-dependent effect: additive and multiplicative. Let $p_j(\tau)$ denote the actual processing time of job $j \in N$ that starts at time $\tau \geq 0$. Then, under a general additive effect, we define

$$p_j(\tau) = p_j + f_j(\tau), \tag{8.1}$$

while under a general multiplicative effect, we define

$$p_j(\tau) = p_j f_j(\tau), \tag{8.2}$$

where $f_j$ is a job-dependent function of the start-time.

Scheduling problems with a multiplicative start-time-dependent effect are considered in Chap. 9. In this chapter, we focus on the problems with an additive effect of the form (8.1). We also study additive models in which the function $f_j(\tau)$ is job-independent, so that the additive effect is of the form

$$p_j(\tau) = p_j + f(\tau). \tag{8.3}$$

V.A. Strusevich and K. Rustogi, *Scheduling with Times-Changing Effects and Rate-Modifying Activities*, International Series in Operations Research & Management Science 243, DOI 10.1007/978-3-319-39574-6_8

In fact, we shall study a more general form of the effect (8.3) that combines a job-independent start-time-dependent effect with a general job-independent positional effect, so that the actual processing time of job $j$ that is scheduled in the $r$th position and starts at time $\tau$ is given by

$$p_j(\tau; r) = \big(p_j + f(\tau)\big)g(r), \tag{8.4}$$

where

- $f : [0, +\infty) \to \mathbb{R}$ is a continuous differentiable function, common to all jobs, that depends on the start time $\tau$ of the job in the $r$th position;
- Array $g(r)$, $1 \le r \le n$, is a monotone sequence of positional factors and defines a positional effect.

Notice that if function $f$ is non-negative and non-decreasing (non-positive and non-increasing), we deal with a start-time-dependent deterioration (learning) effect. In the case of a learning effect, an additional assumption

$$f(\tau) < \min\{p_1, p_2, \dots, p_n\},\ \tau > 0, \tag{8.5}$$

is required. This assumption guarantees that under a learning effect, the actual processing times remain non-negative for each job $j \in N$.

Problems with positional effects are considered in Chap. 7. Recall that if the sequence $g(r)$, $1 \le r \le n$, of positional factors is non-decreasing (non-increasing), we deal with a positional deterioration (learning) effect.

In this chapter, we assume that $f(0) = 0$ and $g(1) = 1$, which guarantees that for the job which is the first in the processing sequence, the actual processing time is equal to its normal time.

As adopted throughout this book, if job $j$ is sequenced in position $\pi(r)$ of permutation $\pi$, its completion time is denoted either by $C_j(\pi)$ or by $C_{\pi(r)}$, whichever is more convenient.

Many problems from the considered range admit a solution by a priority rule. Recall that if the jobs are numbered in accordance with the LPT rule, then

$$p_1 \ge p_2 \ge \cdots \ge p_n, \tag{8.6}$$

while if they are numbered in accordance with the SPT rule, then

$$p_1 \le p_2 \le \cdots \le p_n. \tag{8.7}$$

This chapter is structured as follows. Section 8.1 studies single machine problems with no precedence constraints under various effects, including a combined effect (8.4) and linear job-dependent and job-independent effects. Section 8.2 considers problems with series-parallel precedence constraints, mainly under pure start-time-dependent linear effects.

## 8.1 Scheduling Independent Jobs

In this section, we address various versions of single machine scheduling problems, provided that no precedence constraint is imposed on the set of jobs and the actual processing times are subject to an additive start-time-dependent effect of the form (8.1) or a combined effect of the form (8.4).

### *8.1.1 Combined Effects*

Consider a job-independent nonlinear additive start-time-dependent effect which is combined with a positional effect so that the actual processing time of a job is given by (8.4). For the combined effect (8.4), the problems of minimizing the makespan and the total completion time on a single machine are denoted by $1\left|p_j(\tau; r) = \left(p_j + f(\tau)\right)g(r)\right|C_{\max}$ and by $1\left|p_j(\tau; r) = \left(p_j + f(\tau)\right)g(r)\right| \sum C_j$, respectively.

Function $f$ can take both positive and negative values; in the latter case, it satisfies (8.5).

We start with a rather general statement, which should be seen as an extension of Theorem 2.3.

**Theorem 8.1** *Let $\pi = (\pi(1), \ldots, \pi(n))$ be a permutation, in which two jobs $u$ and $v$ such that*

$$p_u > p_v, \tag{8.8}$$

*occupy two consecutive positions $r$ and $r + 1$, i.e., $u = \pi(r)$ and $v = \pi(r + 1)$. Let permutation $\pi'$ be obtained from $\pi$ by swapping the jobs $u$ and $v$. Then for a single machine problem with a combined effect (8.4) the inequality*

$$C_{\pi(h)} \geq C_{\pi'(h)} \tag{8.9}$$

*holds for all $h$, $1 \leq h \leq n$, provided that function $f$ is non-decreasing and the array $g(r)$, $1 \leq r \leq n$, is non-increasing, i.e., it follows*

$$1 = g(1) \geq g(2) \geq \cdots \geq g(n). \tag{8.10}$$

*Proof* It is convenient to represent permutation $\pi$ as $\pi = (\pi_1, u, v, \pi_2)$, where $\pi_1$ and $\pi_2$ are subsequences of jobs that precede job $u$ and follow job $v$ in permutation $\pi$, respectively. Then, $\pi' = (\pi_1, v, u, \pi_2)$.

We present the proof assuming that both sequences $\pi_1$ and $\pi_2$ are non-empty; otherwise, the corresponding part of the proof can be skipped.

The actual processing times and the completion times of all jobs in sequence $\pi_1$ are not affected by the swap of jobs $u$ and $v$, i.e., (8.9) holds as equality for each $h$, $1 \leq h \leq r - 1$.

Define $X$ as the completion time of the job in the $(r-1)$th position in sequence $\pi$ (or, equivalently, in $\pi'$), i.e., $X = C_{\pi(r-1)} = C_{\pi'(r-1)}$. For $h = r$, we derive that

$$\begin{aligned} C_{\pi(r)} &= C_u(\pi) = X + (p_u + f(X))g(r); \\ C_{\pi'(r)} &= C_v(\pi') = X + (p_v + f(X))g(r). \end{aligned}$$

Due to (8.8), we see that inequality (8.9) holds for $h = r$.

For $h = r + 1$, we derive that

$$\begin{aligned} C_{\pi(r+1)} &= C_v(\pi) = X + (p_u + f(X))g(r) \\ &\quad + (p_v + f(X + (p_u + f(X))g(r)))g(r+1); \\ C_{\pi'(r+1)} &= C_u(\pi') = X + (p_v + f(X))g(r) \\ &\quad + (p_u + f(X + (p_v + f(X))g(r)))g(r+1). \end{aligned}$$

Define

$$\Delta := C_{\pi'(r+1)} - C_{\pi(r+1)}.$$

Writing out the actual processing times of jobs $u$ and $v$ in permutations $\pi$ and $\pi'$, we obtain

$$\begin{aligned} \Delta &= (p_v - p_u)(g(r) - g(r+1)) \\ &\quad + (f(X + (p_v + f(X))g(r)) - f(X + (p_u + f(X))g(r)))g(r+1). \end{aligned} \tag{8.11}$$

Due to (8.8) and (8.10), we have that

$$(p_v - p_u)(g(r) - g(r+1)) < 0.$$

Besides, $X + (p_v + f(X))g(r) = C_{\pi'(r)} \le C_{\pi(r)} = X + (p_u + f(X))g(r)$, so that

$$f(X + (p_v + f(X))g(r)) - f(X + (p_u + f(X))g(r)) \le 0,$$

since $f$ is non-decreasing. Thus, $\Delta \le 0$, i.e., (8.9) holds for $h = r + 1$.

The jobs that follow the position $r + 1$ form the same sequence $\pi_2$ in both permutations $\pi$ and $\pi'$. Each of these jobs starts in permutation $\pi'$ no later than in permutation $\pi$, and therefore, (8.9) holds for each $h$, $r + 2 \le h \le n$.

This proves the theorem. □

Since the positional factors satisfy (8.10), it follows that Theorem 8.1 addresses a single machine model in which a start-time-dependent deterioration effect is combined with a positional learning effect. Theorem 8.1 demonstrates that in a sequence that minimizes any objective function $\Phi(\pi)$ that depends only on the completion times, the jobs may be arranged in such a way that a job with a larger normal processing time is not followed by a job with a smaller normal processing time. Examples of such a function include, but not limited to $C_{\max}$, $\sum C_j^z$, where $z$ is a given positive number, and their linear combination $\xi C_{\max} + \eta \sum C_j^z$.

**Theorem 8.2** *For problem* $1\left|p_j(\tau;r) = \left(p_j + f(\tau)\right)g(r)\right|\Phi$, *where* $\Phi \in \left\{C_{\max}, \sum C_j^z, \xi C_{\max} + \eta \sum C_j^z\right\}$, *under an effect (8.4) that combines an additive start-time-dependent deterioration effect and a positional learning effect an optimal permutation can be found in* $O(n \log n)$ *time by sorting the jobs in accordance with the SPT rule (8.7), provided that function* $f$ *is non-decreasing and the positional factors* $g(r)$, $1 \le r \le n$, *are non-increasing, i.e., (8.10) holds.*

Reformulating Theorem 8.2, we conclude that problem $1\left|p_j(\tau;r) = \left(p_j + f(\tau)\right)g(r)\right|\Phi$, where $\Phi \in \left\{C_{\max}, \sum C_j^z, \xi C_{\max} + \eta \sum C_j^z\right\}$, under a combined effect (8.4) with a non-decreasing function $f$ and non-increasing positional factors $g(r)$, $1 \le r \le n$, admits the 1-priority $\omega(j) = 1/p_j$.

It appears that the analysis of a combined effect (8.4) with an additive start-time-dependent learning effect and a positional deterioration effect, defined by a non-increasing function $f$ and non-decreasing positional factors $g(r)$, $1 \le r \le n$, is not fully symmetric to that presented in Theorem 8.1. We need additional conditions on the derivative of $f$, and even then only a less general statement can be proved.

Before we present the next result, we reproduce a classical statement, known in mathematical analysis as the *Lagrange's mean value theorem.*

**Theorem 8.3** *If a function* $f$ *is continuous on a closed interval* $[a, b]$, *where* $a < b$, *and differentiable on the open interval* $(a, b)$, *then there exists a point* $\zeta \in (a, b)$ *such that*

$$f(a) - f(b) = f'(\zeta)(a - b).$$

Theorem 8.3 is used in the proof of the theorem below.

**Theorem 8.4** *Let* $\pi = (\pi(1), \dots, \pi(n))$ *be a permutation, in which two jobs* $u$ *and* $v$ *such that*

$$p_u < p_v,$$

*occupy two consecutive positions* $r$ *and* $r + 1$, *i.e.,* $u = \pi(r)$ *and* $v = \pi(r + 1)$. *Let permutation* $\pi'$ *be obtained from* $\pi$ *by swapping the jobs* $u$ *and* $v$. *Then for problem* $1\left|p_j(\tau;r) = \left(p_j + f(\tau)\right)g(r)\right|C_{\max}$ *with a combined effect (8.4) the inequality* $C_{\max}\left(\pi'\right) \le C_{\max}(\pi)$ *holds, provided that*

**(i)** $f$ *is differentiable and non-increasing on* $[0, +\infty)$;
**(ii)** *positional factors* $g(r)$, $1 \le r \le n$, *are non-decreasing, i.e.,*

$$1 = g(1) \le g(2) \le \cdots \le g(n), \tag{8.12}$$

*and*

**(iii)** $|f'(\tau)| \le 1/g(n)$ *for* $\tau \in [0, +\infty)$.

*Proof* The first part of the proof follows that of Theorem 8.1. We represent permutations $\pi$ and $\pi'$ as $\pi = (\pi_1, u, v, \pi_2)$, where $\pi_1$ and $\pi_2$ are subsequences of jobs that precede job $u$ and follow job $v$ in permutation $\pi$, respectively. We present the proof assuming that both sequences $\pi_1$ and $\pi_2$ are non-empty; otherwise, the corresponding part of the proof can be skipped.

The actual processing times and the completion times of all jobs in sequence $\pi_1$ are not affected by the swap of jobs $u$ and $v$. Define $X$ as the completion time of the job in the $(r-1)$th position in sequence $\pi$ (or, equivalently, in $\pi'$), i.e., $X := C_{\pi(r-1)} = C_{\pi'(r-1)}$. For $h = r$, we have that $C_{\pi(r)} = X + (p_u + f(X))g(r)$ and $C_{\pi'(r)} = X + (p_v + f(X))g(r)$. Notice that $p_u < p_v$, which implies that $C_{\pi(r)} < C_{\pi'(r)}$. This explains why under the conditions of Theorem 8.4 it is not possible to prove a more general statement, similar to Theorem 8.1.

Further, for the jobs in position $r+1$, as in the proof of Theorem 8.1, define $\Delta := C_{\pi'(r+1)} - C_{\pi(r+1)}$, which can be rewritten as (8.11). Since $p_u < p_v$, $g(r) \le g(r+1)$ due to (8.12), and function $f$ is non-increasing, we deduce that $\Delta \le 0$, i.e., $C_{\pi'(r+1)} \le C_{\pi(r+1)}$.

The rest of the proof relies on condition (iii) and is done by induction. Assume that the inequality (8.9) holds for each $h$, where $r+1 \le h \le q-1 \le n-1$. We prove that (8.9) holds for $h = q$.

The jobs that follow the position $r+1$ form the same sequence $\pi_2$ in both permutations $\pi$ and $\pi'$. Let $x$ be the job scheduled in position $q$, i.e., $x = \pi(q) = \pi'(q)$. We deduce that

$$C_{\pi(q)} = C_x(\pi) = C_{\pi(q-1)} + \big(p_x + f\big(C_{\pi(q-1)}\big)\big)g(q);$$
$$C_{\pi'(q)} = C_x\big(\pi'\big) = C_{\pi'(q-1)} + \big(p_x + f\big(C_{\pi'(q-1)}\big)\big)g(q).$$

Define $\delta := C_{\pi(q-1)} - C_{\pi'(q-1)}$. If $\delta = 0$, then $C_{\pi(q)} = C_{\pi'(q)}$; otherwise, by the induction assumption, $\delta > 0$.

By Theorem 8.3 and due to the fact that $f$ is non-increasing, there exists a point $\zeta \in \big[C_{\pi'(q-1)}, C_{\pi(q-1)}\big]$ such that

$$f\big(C_{\pi'(q-1)}\big) - f\big(C_{\pi'(q-1)} + \delta\big) = -\delta f'(\zeta) = \delta\big|f'(\zeta)\big|.$$

By condition (iii), we have that $\big|f'(\zeta)\big| \le 1/g(n)$, which implies that $f\big(C_{\pi'(q-1)}\big) - f\big(C_{\pi'(q-1)} + \delta\big) \le \delta/g(n)$. Thus, we deduce that

$$\begin{aligned} C_{\pi'(q)} - C_{\pi(q)} &= \big(C_{\pi'(q-1)} - C_{\pi(q-1)}\big) + \big(f\big(C_{\pi'(q-1)}\big) - f\big(C_{\pi(q-1)}\big)\big)g(q) \\ &= -\delta + \big(f\big(C_{\pi'(q-1)}\big) - f\big(C_{\pi'(q-1)} + \delta\big)\big)g(q) \le -\delta + \frac{\delta}{g(n)}g(q) \le 0, \end{aligned}$$

where the last inequality holds because $g(q) \le g(n)$ due to (8.12).

We conclude that the inequality (8.9) holds for each $h$, $r+1 \le h \le n$. In particular, $C_{\pi'(n)} \le C_{\pi(n)}$, i.e., $C_{\max}\big(\pi'\big) \le C_{\max}(\pi)$. This proves the theorem. □

Notice that if $f$ is additionally non-positive, the start-time-dependent component of the combined effect (8.4) represents a learning effect. Condition (iii) of Theorem 8.4 ensures that the learning rate is not too large, so that the actual processing times of jobs never become negative.

Theorem 8.4 immediately leads to the following statement regarding single machine scheduling problems to minimize the makespan.

**Theorem 8.5** *For problem $1\left|p_j(\tau;r)=\left(p_j+f(\tau)\right)g(r)\right|C_{\max}$ under the combined effect (8.4), an optimal permutation can be found in $O(n\log n)$ time by sorting the jobs in accordance with the LPT rule (8.6), provided that the conditions of Theorem 8.4 are satisfied.*

Notice that the proof of Theorem 8.4 in fact demonstrates that the inequality (8.9) holds for each $h$ other than $r$, while $C_{\pi(r)} < C_{\pi'(r)}$. This fact does not allow us to derive any conclusions regarding the status of problem $1\left|p_j(\tau;r)=\left(p_j+f(\tau)\right)\right.$ $g(r)|\sum C_j$, provided that function $f$ is non-increasing on $[0,+\infty)$. Below, we present a counterexample that demonstrates that the problem can be solved neither by the SPT rule nor by the LPT rule, even if a pure additive start-time-dependent effect is considered, i.e., even if $g(r)=1,\ 1\le r\le n$. Notice that if $g(r)=1,\ 1\le r\le n$, then condition (iii) of Theorem 8.4 simply becomes $\left|f'(\tau)\right|\le 1$ for $\tau\in[0,+\infty)$.

*Example 8.1* Consider an instance of problem $1\left|p_j(\tau)=p_j+f(\tau)\right|\sum C_j$ to minimize the sum of completion times under the additive effect (8.3) with the function $f(\tau)=-1+1/(1+\tau),\ \tau>0$. There are 8 jobs with the normal processing times listed below

$$p_1=1,\ p_2=2,\ p_3=3,\ p_4=4,\ p_5=5,\ p_6=5,\ p_7=6,\ p_8=6.$$

It is clear that $f(0)=0$ and function $f$ is decreasing. Also, its derivative $\left|f'(\tau)\right|=\frac{1}{(\tau+1)^2}\le 1$, for $\tau\in[0,+\infty)$. Thus, $f$ satisfies the conditions of Theorem 8.4. For this instance, the total completion time $\sum C_j$ is minimized neither by the SPT sequence,

**Table 8.1** Computations for Example 8.1

| | $\pi=(1,2,3,4,5,6,7,8)$ $[SPT]$ | $\pi=(8,7,6,5,4,4,3,2,1)$ $[LPT]$ | $\pi=(2,1,3,4,5,6,7,8)$ |
|---|---|---|---|
| $C_{\pi(1)}$ | 1.00 | 6.00 | 2.00 |
| $C_{\pi(2)}$ | 2.50 | 11.14 | 2.33 |
| $C_{\pi(3)}$ | 4.79 | 15.23 | 4.63 |
| $C_{\pi(4)}$ | 7.96 | 19.29 | 7.81 |
| $C_{\pi(5)}$ | 12.07 | 22.34 | 11.92 |
| $C_{\pi(6)}$ | 16.15 | 24.38 | 16.00 |
| $C_{\pi(7)}$ | 21.21 | 25.42 | 21.06 |
| $C_{\pi(8)}=C_{\max}(\pi)$ | 26.25 | 25.46 | 26.11 |
| $\sum C_{\pi(j)}$ | 91.92 | 149.24 | 91.87 |

**Table 8.2** Results for scheduling independent jobs on a single machine with an additive start-time-dependent effect

| Condition on $g$ | Condition on $f$ | Condition on $f'$ | Objective | Rule | Statement |
|---|---|---|---|---|---|
| $g \searrow$ | $f \nearrow$ | – | $C_{\max}$ | SPT | Theorem 8.2 |
| $g \searrow$ | $f \nearrow$ | – | $\sum C_j^z$ | SPT | Theorem 8.2 |
| $g \searrow$ | $f \nearrow$ | – | $\xi C_{\max} + \eta \sum C_j^z$ | SPT | Theorem 8.2 |
| $g \nearrow$ | $f \searrow, \exists f'$ | $\lvert f' \rvert \le 1/g(n)$ | $C_{\max}$ | LPT | Theorem 8.5 |
| $g = 1$ | $f \searrow, \exists f'$ | $\lvert f' \rvert \le 1$ | $\sum C_j$ | Open | Example 8.1 |

nor by the LPT sequence. The corresponding computations (accurate to 2 decimal places) are shown in Table 8.1. We see that permutation (2, 1, 3, 4, 5, 6, 7, 8) delivers a smaller value of the total completion time than that produced by each of the SPT and the LPT sequences.

The results presented in Sect. 8.1.1 are summarized in Table 8.2. Here, we write $g = 1$ to indicate that $g(r) = 1$, $1 \le r \le n$; besides, in the first and the second columns, we use symbols $\nearrow$ and $\searrow$ to indicate that either the sequence $g(r)$ or the function $f$ is non-decreasing or non-increasing, respectively.

## 8.1.2 *Job-Dependent Linear Effects*

In this subsection, we address a single machine scheduling problem under pure linear additive job-dependent start-time-dependent effects. Each job $j \in N$ is associated with a normal processing time $p_j$ and a non-negative rate $a_j$. We study the effects, under which the actual processing time $p_j(\tau)$ of job $j$ that starts at time $\tau$ is given either by

$$p_j(\tau) = p_j + a_j \tau, \tag{8.13}$$

or by

$$p_j(\tau) = p_j - a_j \tau. \tag{8.14}$$

Here, $a_j$ is a positive number that represents either a deterioration rate (in the case of effect (8.13)) or a learning rate (in the case of effect (8.14)) of job $j \in N$.

It is clear that both effects (8.13) and (8.14) are versions of the general additive effect (8.1) with either $f_j(\tau) = a_j \tau$ or $f_j(\tau) = -a_j \tau$, respectively. Notice that in the case of the learning effect (8.14), an additional assumption

$$a_j < 1, \quad a_j \left( \sum_{i=1}^{n} p_i - p_j \right) < p_j, \, j \in N, \tag{8.15}$$

is required. The first part of this assumption guarantees that for minimizing a regular objective function, an unnecessary delay in starting a job is counterproductive, and the second part which follows from (8.5) guarantees that the actual processing times are kept non-negative.

First, we provide a closed form formula for computing the completion times on a single machine under the effects (8.13) and (8.14).

**Lemma 8.1** *Let the jobs be processed on a single machine in accordance with a permutation* $\pi = (\pi(1), \pi(2), \dots, \pi(n))$. *Under the effect (8.13) the completion times can be computed by the formula*

$$C_{\pi(k)} = \sum_{j=1}^{k} p_{\pi(j)} \prod_{i=j+1}^{k} \left(1 + a_{\pi(i)}\right),\ 1 \le k \le n, \tag{8.16}$$

*while under the effect (8.14) they can be computed as*

$$C_{\pi(k)} = \sum_{j=1}^{k} p_{\pi(j)} \prod_{i=j+1}^{k} \left(1 - a_{\pi(i)}\right),\ 1 \le k \le n. \tag{8.17}$$

*Proof* We present the derivation of (8.16); the derivation of (8.17) is similar. The proof is by induction. Recall that by a standard agreement, $\prod_{i=u}^{v}(\cdot) = 1$, provided that $u > v$.

For $k = 1$, we see that the actual processing time of job $\pi(1)$ is $p_{\pi(1)}$, so that $C_{\pi(1)} = p_{\pi(1)}$, as required for the basis of induction.

Assume that (8.16) holds for all $k$, where $1 \le k \le q - 1 < n - 1$. We prove that (8.16) holds for $k = q$. By definition and the induction assumption, we have that

$$\begin{aligned} C_{\pi(q)} &= C_{\pi(q-1)} + p_{\pi(q)} + a_{\pi(q)} C_{\pi(q-1)} = p_{\pi(q)} + \left(1 + a_{\pi(q)}\right) C_{\pi(q-1)} \\ &= p_{\pi(q)} + \left(1 + a_{\pi(q)}\right) \sum_{j=1}^{q-1} p_{\pi(k)} \prod_{i=j+1}^{q-1} \left(1 + a_{\pi(i)}\right) \\ &= p_{\pi(q)} + \sum_{j=1}^{q-1} p_{\pi(k)} \prod_{i=j+1}^{q} \left(1 + a_{\pi(i)}\right) = \sum_{j=1}^{q} p_{\pi(k)} \prod_{i=j+1}^{q} \left(1 + a_{\pi(i)}\right), \end{aligned}$$

which proves the lemma. □

We denote the single machine problems for minimizing a function $\Phi$ under the effects (8.13) and (8.14) by $1\left|p_j(\tau) = p_j + a_j \tau\right|\Phi$ and $1\left|p_j(\tau) = p_j - a_j \tau\right|\Phi$, respectively.

Lemma 8.1 together with Theorem 2.5 on minimizing the sum of products implies the following statement.

**Theorem 8.6** *For problem $1\left|p_j(\tau) = p_j + a_j\tau\right|C_{\max}$, with a deterioration effect (8.13), an optimal permutation can be found in $O(n \log n)$ time by sorting the jobs in non-decreasing order of the ratios $p_j/a_j$. For problem $1\left|p_j(\tau) = p_j - a_j\tau\right|C_{\max}$, with a learning effect (8.14) under the assumption (8.15), an optimal permutation can be found in $O(n \log n)$ time by sorting the jobs in non-increasing order of the ratios $p_j/a_j$.*

*Proof* It follows from Lemma 8.1 that for a given permutation $\pi$, we have

$$C_{\max}(\pi) = C_{\pi(n)} = \sum_{j=1}^{n} p_{\pi(j)} \prod_{i=j+1}^{n} \left(1 + a_{\pi(i)}\right)$$

for problem $1\left|p_j(\tau) = p_j + a_j\tau\right|C_{\max}$, while

$$C_{\max}(\pi) = C_{\pi(n)} = \sum_{j=1}^{n} p_{\pi(j)} \prod_{i=j+1}^{n} \left(1 - a_{\pi(i)}\right)$$

for problem $1\left|p_j(\tau) = p_j - a_j\tau\right|C_{\max}$. In either case, minimizing $C_{\max}(\pi)$ is equivalent to minimizing the sum of products $K(\pi)$ defined by (2.14). The conditions of Theorem 2.5 regarding components $b_j$ hold, since in the case of problem $1\left|p_j(\tau) = p_j + a_j\tau\right|C_{\max}$, all differences $b_j - 1 = \left(1 + a_j\right) - 1 = a_j$ are non-negative, while in the case of problem $1\left|p_j(\tau) = p_j - a_j\tau\right|C_{\max}$, all differences $b_j - 1 = \left(1 - a_j\right) - 1 = -a_j$ are non-positive. □

Reformulating Theorem 8.6 in terms of 1-priorities, we conclude that problem $1\left|p_j(\tau) = p_j + a_j\tau\right|C_{\max}$ admits a 1-priority function either $\omega(j) = a_j/p_j$ or $\omega(j) = -p_j/a_j$, while problem $1\left|p_j(\tau) = p_j - a_j\tau\right|C_{\max}$ admits a 1-priority function $\omega(j) = p_j/a_j$. Sequencing jobs in non-increasing order of 1-priorities solves the corresponding problem.

We now pass to problems of minimizing functions related to the sum of the completion times.

**Theorem 8.7** *Problems $1\left|p_j(\tau) = p_j + a_j\tau\right|\sum C_j$ and $1\left|p_j(\tau) = p_j - a_j\tau\right|\sum C_j$ do not admit a 1-priority function.*

*Proof* We present the proof for problem $1\left|p_j(\tau) = p_j + a_j\tau\right|\sum C_j$; the proof for problem $1\left|p_j(\tau) = p_j - a_j\tau\right|\sum C_j$ is similar. To prove the theorem, we use Recipe 2.2 from Sect. 2.1.1. Consider an instance of problem $1\left|p_j(\tau) = p_j + a_j\tau\right|\sum C_j$ with four jobs such that $p_1 = \cdots = p_4 = 1$ and

$$a_1 = 1,\ a_2 = 2,\ a_3 = 1,\ a_4 = 1.$$

Let $F(\pi)$ denote the sum of the completion times of the jobs sequenced in accordance with a permutation $\pi$. If $F(\pi)$ admitted a 1-priority function $\omega(j)$, then for any pair of jobs $u$ and $v$ such that $\omega(u) \geq w(v)$, the value of $F$ for any permutation

**Table 8.3** Computation for the proof of Theorem 8.7

| | $\pi = ((1, 2), 3, 4)$ | $\pi = ((2, 1), 3, 4)$ | $\pi = (3, 4, (1, 2))$ | $\pi = 3, 4, (2, 1)$ |
|---|---|---|---|---|
| $C_{\pi(1)}$ | 1 | 1 | 1 | 1 |
| $C_{\pi(2)}$ | 4 | 3 | 3 | 3 |
| $C_{\pi(3)}$ | 9 | 7 | 7 | 10 |
| $C_{\pi(4)}$ | 19 | 15 | 22 | 21 |
| $F(\pi)$ | 33 | 26 | 33 | 35 |

$\pi$ in which $u$ precedes $v$ should be no larger than the value of $F$ for permutation $\pi'$ obtained from $\pi$ by swapping the jobs $u$ and $v$.

Table 8.3 presents the details of relevant computation that is based on (8.16). We see that

$$F((1, 2), 3, 4) = 33 > F((2, 1), 3, 4) = 26;$$
$$F(3, 4, (1, 2)) = 33 < F(3, 4, (2, 1)) = 35.$$

For the first pair of permutations (those which finish with 3, 4), ordering the jobs 1 and 2 as $(1, 2)$ gives a better value of the function compared to the order $(2, 1)$ of these jobs. On the other hand, for the second pair of permutations (those which start with 3, 4), the opposite phenomenon is observed. We conclude that no 1-priority function exists. □

The problem of minimizing the weighted sum of the completion times $\sum w_j C_j$ under the effect (8.13) is NP-hard, as proved below by polynomial reduction in the following problem.

NON-NUMERICAL 3-PARTITION. Given positive integers $e_1, e_2, \ldots, e_{3r}$ and the index set $R = \{1, \ldots, 3r\}$ such that

- $e_i$ is bounded by a polynomial of $r$, $i \in R$,
- $\frac{1}{4}E < e_i < \frac{1}{2}E$, $i \in R$, and
- $e(R) = \sum_{i \in R} e_i = rR$,

does there exist a partition of set $R$ into $r$ disjoint subsets $R_k$ such that $e(R_k) = \sum_{i \in R_k} e_i = E$ for each $k$, $1 \le k \le r$?

**Theorem 8.8** *Problem* $1\left|p_j(\tau) = p_j + a_j\tau\right|\sum w_j C_j$ *is NP-hard in the ordinary sense.*

*Proof* We show that NON-NUMERICAL 3-PARTITION reduces to the decision version of problem $1\left|p_j(\tau) = p_j + a_j\tau\right|\sum w_j C_j$. The difference between 3-PARTITION formulated in Sect. 1.3.2 and NON-NUMERICAL 3-PARTITION is that in the latter problem, it is additionally assumed that each $e_i$ is bounded by a polynomial of $r$.

Given an instance of Non-numerical 3-Partition, define the following instance of problem $1|p_j(\tau) = p_j + a_j\tau|\sum w_jC_j$. Compute

$$D = r(r+3),\ y = r(r+3)D^{(r+1)E}.$$

There are $n = 4r + 1$ jobs, which are split into two classes: the $U$-jobs, denoted by $U_i$, $1 \le i \le 3r$, and the $V$-jobs, denoted by $V_k$, $k \in \{0, 1, \dots, r\}$. For the $U$-jobs, the parameters are set equal to

$$p_{U_i} = 0,\ a_{U_i} = D^{e_i} - 1,\ w_{U_i} = 1,\ \ 1 \le i \le 3r.$$

For the $V$-jobs, define

$$\begin{aligned} &p_{V_0} = 1,\ a_{V_0} = 0,\ w_{V_0} = y + 1; \\ &p_{V_k} = D^{kE},\ a_{V_k} = 0,\ w_{V_k} = D^{(r+1-k)E}. \end{aligned}$$

We show that Non-numerical 3-Partition has a solution if and only if for the constructed instance of problem $1|p_j(\tau) = p_j + a_j\tau|\sum w_jC_j$ there exists a schedule $S_0$ for which the value of the objective function is at most $2y + 1$.

First, assume that Non-numerical 3-Partition has a solution, and $R_k$, $1 \le k \le r$, are the found sets. Then, a required schedule $S_0$ exists and can be found as follows: The $V$-jobs are scheduled in the sequence $V_0, V_1, \dots V_k$, and between each pair of jobs $V_{k-1}$ and $V_k$, a triple of jobs $U_i$ with $i \in R_k$ are processed.

To demonstrate that for schedule $S_0$, the value of the objective function is at most $y + 1$, compute the contribution $F_k = w_{V_k}C_{V_k}$ for each $V$-job.

It is clear that job $V_0$ completes at time 1, so that $F_0 = y + 1$. Suppose that $R_1 = \{i_1, i_2, i_3\}$ and assume that in schedule $S_0$, jobs $U_{i_1}$, $U_{i_2}$ and $U_{i_3}$ are processed in this order. Finally, job $U_{i_1}$ starts at time 1, its actual processing time is $D^{e_{i_1}} - 1$ and it completes at time $D^{e_{i_1}}$. Similarly, job $U_{i_2}$ starts at time $D^{e_{i_1}}$, its actual processing time is $(D^{e_{i_2}} - 1)\ D^{e_{i_1}}$, and it completes at time $D^{e_{i_1}}D^{e_{i_2}}$. Finally, job $U_{i_3}$ completes at time $D^{e_{i_1}}D^{e_{i_2}}D^{e_{i_3}} = D^{e(R_1)}$. Notice that the completion time of the block of these three jobs, i.e., the start time of job $V_1$, does not depend on the order of the jobs within the block. Thus, we deduce

$$F_1 = \left(D^{e(R_1)} + D^E\right)D^{rE} = D^{e(R_1)+rE} + D^{(r+1)E}.$$

Similarly, we obtain that the start time of job $V_2$ is $\left(D^{e(R_1)} + D^E\right)D^{e(R_2)}$, so that

$$\begin{aligned} F_2 &= \left(\left(D^{e(R_1)} + D^E\right)D^{e(R_2)} + D^{2E}\right)D^{(r-1)E} \\ &= D^{e(R_1)+e(R_2)+(r-1)E} + D^{e(R_2)+rE} + D^{(r+1)E}. \end{aligned}$$

Extending, we derive that for an arbitrary $k$, $1 \le k \le r$, the equality

$$F_k = \sum_{u=1}^{k} D^{\left(\sum_{v=u}^{k} e(R_v)\right)+(r+u-k)E} + D^{(r+1)E} \tag{8.18}$$

holds for an arbitrary $k$, $1 \le k \le r$.

Since Non-numerical 3-Partition has a solution, we know that $e(R_k) = E$, $1 \le k \le n$, which implies that

$$F_k = (k+1)D^{(r+1)E}, \ 1 \le k \le r,$$

so that

$$\sum_{k=1}^{r} F_k = \frac{1}{2} r(r+3)D^{(r+1)E}.$$

Each $U$-job completed between jobs $V_{k-1}$ and $V_k$ makes a contribution to the objective function which is no larger than the completion time of job $V_k$, i.e., $F_k/w_{V_k}$. Since $F_k/w_{V_k} < F_k/w_{V_m}$, it follows that a triple of the $U$-jobs completed between jobs $V_{k-1}$ and $V_k$ makes a total contribution that is no larger than $3F_k/D^E$.

For schedule $S_0$, the objective function is the sum of $\sum_{k=0}^{r} F_k$ plus the total contribution of the $U$-jobs which does not exceed

$$(y+1) + \sum_{k=1}^{r} F_k + \frac{3}{D^E}\sum_{k=1}^{r} F_k < (y+1) + 2\sum_{k=1}^{r} F_k = 2y+1,$$

as required.

Now assume that there exists a required schedule $S_0$, in which the sum of weighted completion times does not exceed $2y+1$.

It immediately follows that job $V_0$ must be scheduled in the time interval [0, 1]; otherwise, its contribution $F_0 > 2w_{V_0}$ is at least $2y+2$.

Using Theorem 2.6 on optimality of the WSPT rule for problem $1||\sum w_j C_j$, it is easy to show that in schedule $S_0$, the $V$-jobs should be sequenced in non-decreasing order of the ratios $p_{V_k}/w_{V_k}$, i.e., in the order of their numbering $F_{V_1}, F_{V_2}, \ldots, F_{V_m}$.

Let $R_k$ denote the index set of the $U$-jobs sequenced between the jobs $F_{k-1}$ and $F_k$, $1 \le k \le r$. Suppose that for some $k$, we have that $e(R_k) > E$. Then, it follows from (8.18) that

$$F_k > D^{e(R_k)+rE} + D^{(r+1)E} \ge D^{(E+1)+rE} + D^{(r+1)E} = D^{(r+1)E}(D+1) > y.$$

Therefore, we must have that $e(R_k) \le E$ for each $k$, $1 \le k \le r$. This together with $e(R) = rE$ yields $e(R_k) = E$, $1 \le k \le r$; i.e., the sets $R_k$ form a solution to Non-numerical 3-Partition, as required.

The presented reduction requires time that is polynomial in $r$ and $E$. Although Non-numerical 3-Partition is NP-complete in the strong sense, the presented reduction is polynomial, not pseudopolynomial, which only allows us to conclude that problem $1\left|p_j(\tau) = p_j + a_j\tau\right|\sum w_j C_j$ is NP-hard in the ordinary sense. $\square$

### 8.1.3 Job-Independent Linear Effects

In this subsection, we address a single machine scheduling problem under linear additive job-independent start-time-dependent effects. We study these effects in combination with positional effects. If each job $j \in N$ is associated with a normal processing time $p_j$, then under the studied effect, the actual processing time $p_j(\tau)$ of job $j$ that starts at time $\tau$ and is placed in position $r$, $1 \le r \le n$, is given either by

$$p_j(\tau; r) = \left(p_j + a\tau\right)g(r) \tag{8.19}$$

or by

$$p_j(\tau; r) = \left(p_j - a\tau\right)g(r). \tag{8.20}$$

Here, $a$ is a positive number that represents either a deterioration rate (in the case of effect (8.19)) or a learning rate (in the case of effect (8.20)), and this rate is common for all jobs $j \in N$. Array $g(r)$, $1 \le r \le n$, is a possibly non-monotone sequence of positional factors and defines an arbitrary positional effect. In general, the sequence $g(r)$, $1 \le r \le n$, need not be monotone; however, if it is non-decreasing (non-increasing), it represents a positional deterioration (learning) effect. Notice that both effects (8.19) and (8.20) are special cases of (8.4).

As usual, it is assumed that $g(1) = 1$, which guarantees that for the job which is the first in the processing sequence, the actual processing time is equal to its normal time. Notice that in the case of a start-time-dependent learning effect (8.20), we must also adopt the additional assumption

$$a < \frac{1}{\max\{g(1), g(2), \ldots, g(n)\}}, \ a\tau < \min\{p_1, p_2, \ldots, p_n\}, \ \tau > 0, \tag{8.21}$$

which follows from (8.5) and (8.15) and guarantees that there is no idle time before the processing of a job and the actual processing times do not assume negative values.

For the combined effect (8.19), the following statement immediately follows from Theorem 8.2.

**Theorem 8.9** *For problem* $1\left|p_j(\tau; r) = \left(p_j + a\tau\right)g(r)\right|\Phi$, *where* $\Phi \in \left\{C_{\max}, \sum C_j^z, \xi C_{\max} + \eta \sum C_j^z\right\}$, *under an effect (8.19) that combines an additive start-time-dependent deterioration effect and a positional learning effect, an optimal permutation can be found in* $O(n \log n)$ *time by sorting the jobs in accordance with the SPT rule, provided that the array* $g(r)$, $1 \le r \le n$, *is non-increasing, i.e., (8.10) holds.*

Reformulating Theorem 8.9 in terms of 1-priorities, we conclude that problem $1\left|p_j(\tau; r) = \left(p_j + a\tau\right)g(r)\right|\Phi$ for $\Phi \in \left\{C_{\max}, \sum C_j^z, \xi C_{\max} + \eta \sum C_j^z\right\}$ admits a 1-priority function either $\omega(j) = 1/p_j$ or $\omega(j) = -p_j$, provided that (8.10) holds.

For the combined effect (8.20) with a negative sign, the following statement follows from Theorem 8.5.

**Theorem 8.10** *For problem $1\left|p_j(\tau;r) = \left(p_j - a\tau\right)g(r)\right|C_{\max}$, under an effect (8.20) that combines an additive start-time-dependent learning effect and a positional deterioration effect, an optimal permutation can be found in $O(n \log n)$ time by sorting the jobs in accordance with the LPT rule, provided that the condition (8.21) holds, and the array $g(r)$, $1 \le r \le n$, is non-decreasing, i.e., (8.12) holds.*

Theorem 8.5 is applicable because for problem $1\left|p_j(\tau;r) = \left(p_j - a\tau\right)\right.$ $g(r)|C_{\max}$, the start-time-dependent function $f(\tau) = -a\tau$ decreases on $[0, \infty)$, and due to the assumption (8.21) we have that $\left|f'(\tau)\right| = a < 1/g(n)$.

Reformulating Theorem 8.10 in terms of 1-priorities, we conclude that problem $1\left|p_j(\tau;r) = \left(p_j - a\tau\right)g(r)\right|C_{\max}$ admits a 1-priority function either $\omega(j) = p_j$ or $\omega(j) = -1/p_j$, provided that (8.12) holds.

Now, we show that single machine problems with the combined effects (8.19) and (8.20) are polynomially solvable for a range of objective functions, even if the sequence $g(r)$, $1 \le r \le n$, is not monotone. As demonstrated in Chap. 7, many single machine scheduling problems with a positional job-independent effect reduce to minimizing a function

$$\Phi(\pi) = \sum_{r=1}^{n} W(r)p_{\pi(r)}, \tag{8.22}$$

where $W(r)$, $1 \le r \le n$, is a suitably defined positional weight which denotes the contribution of a job scheduled in the $r$th position to the objective function. Function (8.22) is in a linear form, and an optimal permutation can be found in $O(n \log n)$ time by Algorithm Match given in Sect. 2.1.

**Theorem 8.11** *Problems $1\left|p_j(\tau;r) = \left(p_j + a\tau\right)g(r)\right|\Phi$ and $1\left|p_j(\tau;r) = \left(p_j - a\tau\right)g(r)\right|\Phi$, where $\Phi \in \left\{C_{\max}, \sum C_j, \xi C_{\max} + \eta \sum C_j\right\}$, under no assumption on the array $g(r)$, $1 \le r \le n$, can be reduced to minimizing a generic objective function (8.22). Each of the problems can be solved by Algorithm Match in $O(n \log n)$ time.*

*Proof* We present a detailed proof for effect (8.19); effect (8.20) can be treated similarly. Let $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ be an arbitrary permutation of jobs. It is easy to deduce from Lemma 8.1 that the completion times of jobs can computed as

$$\begin{aligned}
C_{\pi(1)} &= p_{\pi(1)}g(1);\\
C_{\pi(2)} &= C_{\pi(1)} + \left(p_{\pi(2)} + aC_{\pi(1)}\right)g(2) = p_{\pi(2)}g(2) + C_{\pi(1)}(1 + ag(2))\\
&= p_{\pi(1)}g(1)(1 + ag(2)) + p_{\pi(2)}g(2);\\
&\ \ \vdots\\
C_{\pi(r)} &= \sum_{k=1}^{r} p_{\pi(k)}g(k) \prod_{i=k+1}^{r} (1 + ag(i)),\ 1 \le r \le n,
\end{aligned} \tag{8.23}$$

where, as before, for $k + 1 > r$, an empty product $\prod_{i=k+1}^{r} (\cdot)$ equals one.

For the makespan objective, i.e., for problem $1\left|p_j(\tau;r) = \left(p_j + a\tau\right)g(r)\right|C_{\max}$, adopting (8.23), we deduce that

$$C_{\max}(\pi) = \sum_{r=1}^{n} p_{\pi(r)}\left(g(r)\prod_{i=r+1}^{n}(1 + ag(i))\right),$$

which can be rewritten as (8.22) with

$$W(r) = g(r)\prod_{i=r+1}^{n}(1 + ag(i)),\ 1 \le r \le n. \tag{8.24}$$

Clearly, problem $1\left|p_j(\tau;r) = \left(p_j - a\tau\right)g(r)\right|C_{\max}$ reduces to minimizing (8.22) with the positional weights

$$W(r) = g(r)\prod_{i=r+1}^{n}(1 - ag(i)),\ 1 \le r \le n. \tag{8.25}$$

For the total completion time objective, i.e., for problem $1\left|p_j(\tau;r) = \left(p_j + a\tau\right)\right.$ $g(r)| \sum C_j$, adopting (8.23), we deduce that

$$\sum C_j(\pi) = \sum_{r=1}^{n}\left(\sum_{k=1}^{r} p_{\pi(k)}g(k)\prod_{i=k+1}^{r}(1 + ag(i))\right).$$

Changing the order of summation, we get

$$\sum C_j(\pi) = \sum_{r=1}^{n} p_{\pi(r)}g(r)\left(\sum_{k=r}^{n}\prod_{i=r+1}^{k}(1 + ag(i))\right)$$

which can be rewritten as (8.22) with

$$W(r) = g(r)\left(\sum_{k=r}^{n}\prod_{i=r+1}^{k}(1 + ag(i))\right),\ 1 \le r \le n. \tag{8.26}$$

Clearly, problem $1\left|p_j(\tau;r) = \left(p_j - a\tau\right)g(r)\right|\sum C_j$ reduces to minimizing (8.22) with the positional weights

$$W(r) = g(r)\left(\sum_{k=r}^{n}\prod_{i=r+1}^{k}(1 - ag(i))\right),\ 1 \le r \le n. \tag{8.27}$$

Finally, using the above relations together with (8.24) and (8.25), we reduce problems $1\left|p_j(\tau;r) = \left(p_j + a\tau\right)g(r)\right|\xi C_{\max} + \eta\sum C_j$ and $1\left|p_j(\tau;r) = \left(p_j - a\tau\right)\right.$

$g(r)|\xi C_{\max} + \eta \sum C_j$ to minimizing a generic objective function of the form (8.22) with the positional weights defined by

$$W(r) = \xi\left[g(r)\prod_{i=r+1}^{n}(1+ag(i))\right] + \eta\left[g(r)\left(\sum_{k=r}^{n}\prod_{i=r+1}^{k}(1+ag(i))\right)\right],\ 1 \le r \le n, \quad (8.28)$$

and

$$W(r) = \xi\left[g(r)\prod_{i=r+1}^{n}(1-ag(i))\right] + \eta\left[g(r)\left(\sum_{k=r}^{n}\prod_{i=r+1}^{k}(1-ag(i))\right)\right],\ 1 \le r \le n,$$

respectively.

As stated in Sect. 2.1, a permutation that minimizes function $\Phi(\pi)$ of the form (8.22) over all permutations of jobs of set $N$ can be found by Algorithm Match which requires $O(n \log n)$ time. □

Theorem 8.11 guarantees that an optimal solution to problem $1\left|p_j(\tau; r) = \left(p_j \pm a\tau\right)g(r)\right|\Phi$, where $\Phi \in \left\{C_{\max}, \sum C_j, \xi C_{\max} + \eta \sum C_j\right\}$ can be obtained in $O(n \log n)$ time under much more general assumptions that those imposed in Theorems 8.9 and 8.10. In general, for problem $1\left|p_j(\tau; r) = \left(p_j \pm a\tau\right)g(r)\right|\Phi$ an optimal permutation need not be obtained by a simple priority rule, such as SPT or LPT.

According to Theorem 8.9, an optimal permutation for problem $1\left|p_j(\tau; r) = \left(p_j + a\tau\right)g(r)\right|\Phi$, where $\Phi \in \left\{C_{\max}, \sum C_j, \xi C_{\max} + \eta \sum C_j\right\}$ and the array $g(r)$, $1 \le r \le n$, are non-increasing, can be found by the SPT rule. The same conclusion can be derived from Theorem 8.11, which reduces the problem to minimizing a function (8.22) by Algorithm Match. Indeed, it is easy to verify that for each of these problems, the sequence of the positional weights $W(r)$, $1 \le r \le n$, is non-increasing. For illustration, observe that in the case of the makespan objective, it follows from (8.24) that for any $r$, $1 \le r \le n-1$, the inequality

$$\frac{W(r)}{W(r+1)} = \frac{g(r)}{g(r+1)}(1 + ag(r+1)) > 1$$

holds. Recall that according to Algorithm Match, in order to minimize the objective function (8.22), the jobs must be sequenced in non-decreasing order of their processing times, i.e., in accordance with the SPT rule.

On the other hand, notice that Theorem 8.9 guarantees that a permutation that is optimal for problem $1\left|p_j(\tau; r) = \left(p_j + a\tau\right)g(r)\right|\sum C_j^z$ can be found by the SPT rule, while such a conclusion does not follow from Theorem 8.11.

Notice that Theorem 8.10 does not resolve the status of problem $1\left|p_j(\tau) = \left(p_j - a\tau\right)g(r)\right|\sum C_j$, while this can be easily done by applying Theorem 8.11, even if the array $g(r)$, $1 \le r \le n$, is not monotone. Moreover, if $g(r) = 1$, $1 \le r \le n$, Theorem 8.11 implies the following statement.

**Corollary 8.1** *For problem $1\left|p_j(\tau) = p_j - a\tau\right|\sum C_j$, under a pure additive start-time-dependent learning effect, an optimal permutation can be found in $O(n \log n)$ time by sorting the jobs in accordance with the SPT rule, provided that (8.21) holds, i.e., $a < 1$.*

*Proof* According to Theorem 8.11, for problem $1\left|p_j(\tau) = p_j - a\tau\right|\sum C_j$, the objective function can be written in the form of a generic objective function (8.22) with the positional weights given by (8.27), applied with $g(r) = 1$, $1 \le r \le n$. We get

$$W(r) = \sum_{k=r}^{n} \prod_{i=r+1}^{k} (1 - a) = \sum_{k=r}^{n} (1 - a)^{k-r} = \frac{1 - (1 - a)^{n-r+1}}{a}, \ 1 \le r \le n. \tag{8.29}$$

Notice that since $a < 1$, the positional weights $W(r)$, $1 \le r \le n$, given by (8.29), form a non-increasing sequence. Thus, in an optimal permutation generated by Algorithm Match, the jobs are sequenced in non-decreasing order of their normal processing times, i.e., in accordance with the SPT rule. □

Reformulating Corollary 8.1 in terms of 1-priorities, we conclude that problem $1\left|p_j(\tau) = p_j - a\tau\right|\sum C_j$ admits a 1-priority function $\omega(j) = 1/p_j$.

Theorem 8.11 reduces several problems to an assignment problem with a product matrix, a tool which is extensively used in Chap. 7 to study problems with positional effects. In fact, for an objective function $\Phi \in \left\{C_{\max}, \sum C_j\right\}$, there are interesting links between problems $1\left|p_j(\tau) = p_j - a\tau\right|\Phi$ and $1\left|p_j(\tau) = p_j + a\tau\right|\Phi$, on one hand, and problems of minimizing $C_{\max}$ with positional effects (see Sect. 7.2.1). Theorem 8.11 implies the following statement.

**Corollary 8.2** *The following problems are equivalent:*

**(a)** *problem $1\left|p_j(\tau) = p_j - a\tau\right|C_{\max}$ under a pure additive start-time-dependent learning effect and problem $1\left|p_j(r) = p_j g(r)\right|C_{\max}$ with a positional exponential deterioration effect (7.13); both problems are solvable by the LPT rule;*

**(b)** *problem $1\left|p_j(\tau) = p_j + a\tau\right|C_{\max}$ under a pure additive start-time-dependent deterioration effect and problem $1\left|p_j(r) = p_j g(r)\right|C_{\max}$ with a positional exponential learning effect (7.15); both problems are solvable by the SPT rule;*

**(c)** *problem $1\left|p_j(\tau) = p_j - a\tau\right|\sum C_j$ and problem $1\left|p_j(\tau) = p_j + a\tau\right|\sum C_j$, on one hand, and problem $1\left|p_j(r) = p_j g(r)\right|C_{\max}$ with a positional learning effect, on the other hand; all these problems are solvable by the SPT rule.*

*Proof* In order to prove statement (a), take problem $1\left|p_j(\tau) = p_j - a\tau\right|C_{\max}$. According to Theorem 8.11, for problem $1\left|p_j(\tau) = p_j - a\tau\right|C_{\max}$, the objective function can be written in the form of a generic objective function (8.22) with the positional weights given by (8.25). Substituting $g(r) = 1$, $1 \le r \le n$, we get

$$W(r) = \prod_{i=r+1}^{n}(1-a) = (1-a)^{n-r},\ 1 \le r \le n.$$

Define $\gamma := 1/(1-a)$ and rewrite

$$W(r) = (1-a)^{n-1}\gamma^{r-1},\ 1 \le r \le n.$$

Notice that $\gamma > 1$, since $0 < a < 1$. Disregarding the sequence-independent multiplicative constant $(1-a)^{n-1}$, we see that problem $1\left|p_j(\tau) = p_j - a\tau\right|C_{\max}$ is equivalent to the problem of minimizing the function $\sum_{r=1}^{n} p_{\pi(r)}\gamma^{r-1}$. The latter function is the objective function in problem $1\left|p_j(r) = p_j g(r)\right|C_{\max}$ with a positional exponential deterioration effect of the form (7.13). According to Theorem 7.2, the latter problem is solvable by the LPT rule, which complies with the earlier proved Theorem 8.10.

The proof of statement (b) is similar. In this case, problem $1\left|p_j(\tau) = p_j + a\tau\right| C_{\max}$ is equivalent to the problem of minimizing the function $\sum_{r=1}^{n} p_{\pi(r)}\gamma^{r-1}$, where $\gamma = 1/(1+a) < 1$. The latter function is the objective function in problem $1\left|p_j(r) = p_j g(r)\right|C_{\max}$ with a positional exponential learning effect (7.15). According to Theorem 7.2, the latter problem is solvable by the SPT rule, which complies with the earlier proved Theorem 8.9.

The proof of statement (c) follows from Corollary 8.1. Problem $1\left|p_j(\tau) = p_j + a\tau\right|\sum C_j$ is equivalent to the problem of minimizing the function (8.22), where

$$W(r) = \frac{(1+a)^{n-r+1} - 1}{a},\ 1 \le r \le n,$$

which form a non-increasing sequence, i.e., $W(1) \ge W(2) \ge \cdots \ge W(n)$. For $g(r) = W(r)$, $1 \le r \le n$, problem $1\left|p_j(\tau) = p_j + a\tau\right|\sum C_j$ is equivalent to problem $1\left|p_j(r) = p_j g(r)\right|C_{\max}$ with a positional learning effect that satisfies (7.2) and (7.11). According to Theorem 7.2, the latter problem is solvable by the SPT rule, which complies with the earlier proved Theorem 8.9.

Problem $1\left|p_j(\tau) = p_j - a\tau\right|\sum C_j$ can be handled similarly, with non-increasing sequence of the positional weights given by (8.29) (see the proof of Corollary 8.1). Thus, problem $1\left|p_j(\tau) = p_j - a\tau\right|\sum C_j$ reduces to problem $1\left|p_j(r) = p_j g(r)\right|C_{\max}$ with $g(r) = W(r)$, $1 \le r \le n$. The latter problem is solvable by the SPT rule, which complies with Corollary 8.1. □

The results of Sect. 8.1.3 are summarized in Table 8.4.

**Table 8.4** Results for scheduling independent jobs on a single machine with additive start-time-dependent job-independent linear effects

| Effect | Condition on $g$ | Objective | Rule | Statement |
|---|---|---|---|---|
| $(p_j + a\tau)g(r)$ | $g \searrow$ | $C_{\max}$ | SPT | Theorem 8.9 |
| $(p_j + a\tau)g(r)$ | $g \searrow$ | $\sum C_j^z$ | SPT | Theorem 8.9 |
| $(p_j + a\tau)g(r)$ | $g \searrow$ | $\xi C_{\max} + \eta \sum C_j^z$ | SPT | Theorem 8.9 |
| $(p_j - a\tau)g(r)$ | $g \nearrow$ | $C_{\max}$ | LPT | Theorem 8.10 |
| $p_j - a\tau$ | – | $\sum C_j$ | SPT | Corollary 8.1 |
| $(p_j \pm a\tau)g(r)$ | Arbitrary | $C_{\max}$ | Algorithm Match | Theorem 8.11 |
| $(p_j \pm a\tau)g(r)$ | Arbitrary | $\sum C_j$ | Algorithm Match | Theorem 8.11 |
| $(p_j \pm a\tau)g(r)$ | Arbitrary | $\xi C_{\max} + \eta \sum C_j$ | Algorithm Match | Theorem 8.11 |

## 8.2 Scheduling Under Precedence Constraints

In this section, we consider single machine problems under additive start-time-dependent effects. Unlike in Sect. 8.1, here we assume that the jobs of set $N$ are not independent and a precedence relation given by a series-parallel reduction graph $G = (N, U)$ is imposed over the set $N$ of jobs.

In the remainder of this section, we focus on problems $1\left|p_j(\tau) = p_j + a_j\tau, SP - prec\right|C_{\max}$ and $1\left|p_j(\tau) = p_j - a_j\tau, SP - prec\right|C_{\max}$, as well as on problems $1\left|p_j(\tau) = p_j + a\tau, SP - prec\right|\sum C_j$ and $1\left|p_j(\tau) = p_j - a\tau, SP - prec\right|\sum C_j$. We show that in these problems, the objective function is priority-generating, which means that each of these problems is solvable in $O(n \log n)$ time. See Chap. 3 for definitions and main results on scheduling under precedence constraints.

Given a scheduling problem with an additive start-time-dependent effect, let $\pi$ be a (partial) permutation of jobs contained as a subsequence in some schedule. The length of a permutation $\pi$, i.e., the number of elements in $\pi$, is denoted by $|\pi|$.

Assuming that the first job in a partial permutation $\pi$ starts at time $t \geq 0$, let $C_{\pi(k)}^{(t)}$ denote the completion time of the job sequenced in the $k$th position. Let $C_{\max}(\pi; t)$ denote the maximum completion time of the jobs in $\pi$.

### 8.2.1 *Job-Dependent Linear Effects*

We present our reasoning for problem $1\left|p_j(\tau) = p_j + a_j\tau, SP - prec\right|C_{\max}$; the reasoning for problem $1\left|p_j(\tau) = p_j - a_j\tau, SP - prec\right|C_{\max}$ is similar. For a partial permutation $\pi$, rewrite (8.16) in an equivalent form

$$C_k^{(0)}(\pi) = \sum_{j=1}^{k} p_{\pi(j)} \prod_{i=j+1}^{k} \left(1 + a_{\pi(i)}\right), \ 1 \leq k \leq |\pi|. \tag{8.30}$$

For our analysis, we need the following auxiliary result.

**Lemma 8.2** *For problem* $1\left|p_j(\tau) = p_j + a_j\tau\right|C_{\max}$, *if the jobs are sequenced in accordance with a permutation* $\pi$ *and the first job starts at time* $t$, *then the completion time* $C^{(t)}_{\pi(k)}$ *of the job in position* $k$, $1 \le k \le |\pi|$, *is given by*

$$\begin{aligned} C^{(t)}_{\pi(k)} &= C^{(0)}_{\pi(k)} + t\prod_{i=1}^{k}\left(1 + a_{\pi(i)}\right) \qquad (8.31)\\ &= \sum_{j=1}^{k} p_{\pi(j)} \prod_{i=j+1}^{k}\left(1 + a_{\pi(i)}\right) + t\prod_{i=1}^{k}\left(1 + a_{\pi(i)}\right). \end{aligned}$$

*Proof* The proof is by induction. First, observe that

$$C^{(t)}_{\pi(1)} = t + (p_{\pi(1)} + a_{\pi(1)}t) = p_{\pi(1)} + t(1 + a_{\pi(1)}) = C^{(0)}_{\pi(1)} + t(1 + a_{\pi(1)}),$$

which corresponds to (8.31) for $k = 1$.

Assume that (8.31) holds for all $k$ such that $k \le q - 1 \le n - 1$. We derive

$$C^{(t)}_{\pi(q)} = C^{(t)}_{\pi(q-1)} + \left(p_{\pi(q)} + a_{\pi(q)}C^{(t)}_{\pi(q-1)}\right) = C^{(t)}_{\pi(q-1)}(1 + a_{\pi(q)}) + p_{\pi(q)}.$$

By the induction hypothesis,

$$\begin{aligned} C^{(t)}_{\pi(q)} &= (1 + a_{\pi(q)})\left(\sum_{j=1}^{q-1} p_{\pi(j)} \prod_{i=j+1}^{q-1}\left(1 + a_{\pi(i)}\right) + t\prod_{i=1}^{q-1}\left(1 + a_{\pi(i)}\right)\right) + p_{\pi(q)}\\ &= \sum_{j=1}^{q-1} p_{\pi(j)} \prod_{i=j+1}^{q}\left(1 + a_{\pi(i)}\right) + p_{\pi(q)} + t\prod_{i=1}^{q}\left(1 + a_{\pi(i)}\right) = C^{(0)}_{\pi(q)} + t\prod_{i=1}^{q}\left(1 + a_{\pi(i)}\right), \end{aligned}$$

as required. □

We now demonstrate that the makespan under the deterioration effect (8.13) is priority-generating. Let $\pi^{\alpha\beta} = (\pi_1\alpha\beta\pi_2)$ and $\pi^{\beta\alpha} = (\pi_1\beta\alpha\pi_2)$ be two permutations of all jobs that only differ in the order of the subsequences $\alpha$ (containing $u$ jobs) and $\beta$ (containing $v$ jobs). Define

$$\Delta := C_{\max}(\pi^{\alpha\beta}) - C_{\max}(\pi^{\beta\alpha}) = C_{\max}(\pi^{\alpha\beta}; 0) - C_{\max}(\pi^{\beta\alpha}; 0).$$

In order to verify that the objective function is priority-generating, we need to determine a sufficient condition for the inequality $\Delta \le 0$ to hold. From Lemma 8.2, we obtain

$$\begin{aligned} C_{\max}(\pi_1\alpha\beta\pi_2;0) &= C_{\max}(\pi_2; C_{\max}(\pi_1\alpha\beta;0)) \\ &= C_{\max}(\pi_2;0) + C_{\max}(\pi_1\alpha\beta;0)\prod_{j=1}^{|\pi_2|}\left(1+a_{\pi_2(j)}\right); \\ C_{\max}(\pi_1\beta\alpha\pi_2;0) &= C_{\max}(\pi_2; C(\pi_1\beta\alpha;0)) \\ &= C_{\max}(\pi_2;0) + C_{\max}(\pi_1\beta\alpha;0)\prod_{j=1}^{|\pi_2|}\left(1+a_{\pi_2(j)}\right), \end{aligned}$$

so that $\Delta = (C_{\max}(\pi_1\alpha\beta;0) - C_{\max}(\pi_1\beta\alpha;0))\prod_{j=1}^{|\pi_2|}\left(1+a_{\pi_2(j)}\right)$. Applying Lemma 8.2 again, we obtain

$$\begin{aligned} C_{\max}(\pi_1\alpha\beta;0) &= C_{\max}(\alpha\beta;0) + C_{\max}(\pi_1;0)\prod_{j=1}^{u}\left(1+a_{\alpha(j)}\right)\prod_{j=1}^{v}\left(1+a_{\beta(j)}\right); \\ C_{\max}(\pi_1\beta\alpha,0) &= C_{\max}(\beta\alpha;0) + C_{\max}(\pi_1;0)\prod_{j=1}^{u}\left(1+a_{\alpha(j)}\right)\prod_{j=1}^{v}\left(1+a_{\beta(j)}\right), \end{aligned}$$

so that $\Delta = (C_{\max}(\alpha\beta;0) - C_{\max}(\beta\alpha;0))\prod_{j=1}^{|\pi_2|}\left(1+a_{\pi_2(j)}\right)$. Another application of Lemma 8.2 gives

$$\begin{aligned} C_{\max}(\alpha\beta;0) &= C_{\max}(\beta;0) + C_{\max}(\alpha;0)\prod_{j=1}^{v}\left(1+a_{\beta(j)}\right); \\ C_{\max}(\beta\alpha;0) &= C_{\max}(\alpha;0) + C_{\max}(\beta;0)\prod_{j=1}^{u}\left(1+a_{\alpha(j)}\right), \end{aligned}$$

and this yields

$$\begin{aligned} \Delta = {} & \prod_{j=1}^{|\pi_2|}\left(1+a_{\pi_2(j)}\right) \\ & \times\left(C_{\max}(\alpha,0)\left(\prod_{j=1}^{v}\left(1+a_{\beta(j)}\right)-1\right) - C_{\max}(\beta,0)\left(\prod_{j=1}^{u}\left(1+a_{\alpha(j)}\right)-1\right)\right). \end{aligned} \tag{8.32}$$

Using (8.32), we can establish the following result.

**Theorem 8.12** *For problem $1\left|p_j(\tau) = p_j + a_j\tau, SP - prec\right|C_{\max}$, the objective function is priority-generating and*

$$\omega(\pi) = \frac{\prod_{j=1}^{|\pi|}\left(1 + a_{\pi(j)}\right) - 1}{C_{\max}(\pi; 0)} = \frac{\prod_{j=1}^{|\pi|}\left(1 + a_{\pi(j)}\right) - 1}{\sum_{j=1}^{|\pi|} p_{\pi(j)} \prod_{i=j+1}^{|\pi|}\left(1 + a_{\pi(i)}\right)} \tag{8.33}$$

*is its priority function. Problem $1\left|p_j(\tau) = p_j + a_j\tau, SP - prec\right|C_{\max}$ is solvable in $O(n \log n)$ time.*

*Proof* Dividing (8.32) by $C_{\max}(\alpha; 0)C_{\max}(\beta; 0)$, we deduce that $\Delta \le 0$, provided that

$$\frac{\prod_{j=1}^{u}\left(1 + a_{\alpha(j)}\right) - 1}{C_{\max}(\alpha; 0)} \ge \frac{\prod_{j=1}^{v}\left(1 + a_{\beta(j)}\right) - 1}{C_{\max}(\beta; 0)}.$$

For an arbitrary (partial) permutation $\pi$, define the function $\omega(\pi)$ by (8.33). It is easily verified that $\omega(\alpha) > \omega(\beta)$ implies $C_{\max}(\pi^{\alpha\beta}) \le C_{\max}(\pi^{\beta\alpha})$, while $\omega(\alpha) = \omega(\beta)$ implies $C_{\max}(\pi^{\alpha\beta}) = C_{\max}(\pi^{\beta\alpha})$, as required. □

Observe that if (8.33) is applied to a single job $j$, then for the corresponding problem with independent jobs, $\omega(j) = a_j/p_j$ is a 1-priority function, which complies with Theorem 8.6.

For problem $1\left|p_j(\tau) = p_j - a_j\tau, SP - prec\right|C_{\max}$, a result similar to Theorem 8.12 is easy to derive. The corresponding priority function is

$$\omega(\pi) = \frac{\prod_{j=1}^{|\pi|}\left(1 - a_{\pi(j)}\right) - 1}{C_{\max}(\pi; 0)} = \frac{\prod_{j=1}^{|\pi|}\left(1 - a_{\pi(j)}\right) - 1}{\sum_{j=1}^{|\pi|} p_{\pi(j)} \prod_{i=j+1}^{|\pi|}\left(1 - a_{\pi(i)}\right)},$$

which in the case of a single job complies with Theorem 8.6.

The results obtained for the makespan cannot be extended to the minsum objective functions of completion times. Indeed, Theorem 8.7 states that problem $1\left|p_j(\tau) = p_j + a_j\tau\right|\sum C_j$ does not admit a 1-priority function.

### 8.2.2 Job-Independent Linear Effects

We present our reasoning for problem $1\left|p_j(\tau) = p_j + a\tau, SP - prec\right|\sum C_j$; the reasoning for problem $1\left|p_j(\tau) = p_j - a\tau, SP - prec\right|\sum C_j$ is similar. First, we derive a preliminary result on the function and

$$F(\pi;t) = \sum_{k=1}^{|\pi|} C_k^{(t)}(\pi)$$

which represents the total completion time for a partial permutation $\pi$, provided that the first job in this permutation starts at time $t$. Notice that it directly follows from Lemma 8.2 that

$$C_{\pi(k)}^{(t)} = C_{\pi(k)}^{(0)} + t(1+a)^k = \sum_{j=1}^{k} p_{\pi(j)}(1+a)^{k-j} + t(1+a)^k. \tag{8.34}$$

**Lemma 8.3** *For problem $1\left|p_j(\tau) = p_j + a\tau\right|\sum C_j$, if the jobs are sequenced from time 0 according to a permutation $(\pi'\pi''\pi''')$, then*

$$\begin{aligned} F(\pi'\pi''\pi''';0) = {} & F(\pi';0) + F(\pi'';0) + F(\pi''';0) + C_{\max}(\pi';0)\sum_{k=1}^{|\pi''|}(1+a)^k \\ & + C_{\max}(\pi',0)(1+a)^{|\pi_2|}\sum_{k=1}^{|\pi'''|}(1+a)^k + C_{\max}(\pi'',0)\sum_{k=1}^{|\pi'''|}(1+a)^k. \end{aligned} \tag{8.35}$$

*Proof* Denote $t' := C_{\max}(\pi';0)$. Applying Lemma 8.2 successively, we obtain

$$\begin{aligned} F(\pi'\pi''\pi''';0) &= F(\pi';0) + F(\pi'';t') + F(\pi''', C_{\max}(\pi'\pi'';0)) \\ &= F(\pi';0) + \left(F(\pi'';0) + t'\sum_{k=1}^{|\pi''|}(1+a)^k\right) \\ &\quad + \left(F(\pi''';0) + C_{\max}(\pi'\pi'';0)\sum_{k=1}^{|\pi'''|}(1+a)^k\right). \end{aligned}$$

Further, since

$$C_{\max}(\pi'\pi'';0) = C_{\max}(\pi'';t') = C_{\max}(\pi'';0) + t'(1+a)^{|\pi''|}$$

we obtain the desired expression (8.35) for $F(\pi'\pi''\pi''';0)$. □

It can be verified that Theorem 3.4 formulated for constant processing times holds for a deterioration effect (8.13). Thus, if a priority function exists for the objective $\sum C_j$, it must be given by the function $\omega$ which serves as the priority function for minimizing the makespan. In our case, the required function can be obtained by adapting (8.33) to the case of equal rates $a_j = a$, $j \in N$, and can be written as

$$\omega(\pi) = \frac{(1+a)^{|\pi|}-1}{C_{\max}(\pi;0)} = \frac{(1+a)^{|\pi|}-1}{\sum_{j=1}^{|\pi|} p_{\pi(j)}(1+a)^{|\pi|-j}}. \tag{8.36}$$

We now establish that $\omega$ given by (8.36) is indeed the required priority function.

**Theorem 8.13** *For problem $1\left|p_j(\tau) = p_j + a\tau, SP - prec\right|\sum C_j$, the objective function is priority-generating and (8.36) is its priority function. Problem $1\left|p_j(\tau) = p_j + a\tau, SP - prec\right|\sum C_j$ is solvable in $O(n \log n)$ time.*

*Proof* Let $\pi^{\alpha\beta} = (\pi_1\alpha\beta\pi_2)$ and $\pi^{\beta\alpha} = (\pi_1\beta\alpha\pi_2)$ be two permutations of all jobs that only differ in the order of the subsequences $\alpha$ (containing $u$ jobs) and $\beta$ (containing $v$ jobs). Define

$$\Delta := F(\pi^{\alpha\beta}; 0) - F(\pi^{\beta\alpha}; 0) = F(\pi_1\alpha\beta\pi_2; 0) - F(\pi_1\beta\alpha\pi_2; 0).$$

Using (8.35) for $\pi' = \pi_1$, $\pi'' = \alpha\beta$, and $\pi''' = \pi_2$, and for $\pi' = \pi_1$, $\pi'' = \beta\alpha$, and $\pi''' = \pi_2$, we obtain

$$\Delta = (F(\alpha\beta; 0) - F(\alpha\beta; 0)) + (C_{\max}(\alpha\beta; 0) - C_{\max}(\beta\alpha; 0)) \sum_{k=1}^{|\pi_2|}(1+a)^k. \tag{8.37}$$

A further application of (8.35) for $\pi' = \alpha$, $\pi'' = \beta$, and for $\pi' = \beta$, $\pi'' = \alpha$, where $\pi'''$ is empty in both cases, yields

$$\begin{aligned} F(\alpha\beta; 0) - F(\alpha\beta; 0) &= C_{\max}(\alpha; 0)\sum_{k=1}^{v}(1+a)^k - C_{\max}(\beta; 0)\sum_{k=1}^{u}(1+a)^k \qquad (8.38) \\ &= \left(\frac{(1+a)C_{\max}(\alpha; 0)C_{\max}(\beta; 0)}{a}\right)\left(\frac{(1+a)^v - 1}{C_{\max}(\beta; 0)} - \frac{(1+a)^u - 1}{C_{\max}(\alpha; 0)}\right). \end{aligned}$$

Suppose that $\omega(\alpha) > \omega(\beta)$. From (8.38), the value of $F(\alpha\beta, 0) - F(\alpha\beta, 0)$ is negative, while the value of $C_{\max}(\alpha\beta, 0) - C_{\max}(\beta\alpha, 0)$ is non-positive since $\omega$ is a priority function for the makespan. Thus, we deduce from (8.38), that $\Delta$ is negative. Similarly, for $\omega(\alpha) = \omega(\beta)$, we obtain $F(\alpha\beta, 0) = F(\alpha\beta, 0)$ and $C_{\max}(\alpha\beta, 0) = C_{\max}(\beta\alpha, 0)$, so we deduce from (8.38) that $\Delta = 0$. This completes the proof. □

Observe that if (8.36) is applied to a single job $j$, then $\omega(\pi)$ reduces to $a/p_j$, i.e., $\omega(j) = 1/p_j$ is a 1-priority function, which complies with Theorem 8.9.

For problem $1\left|p_j(\tau) = p_j - a_j\tau, SP - prec\right|C_{\max}$, a result similar to Theorem 8.13 is easy to derive. The corresponding priority function is

$$\omega(\pi) = \frac{(1-a)^{|\pi|} - 1}{\sum_{j=1}^{|\pi|} p_{\pi(j)}(1-a)^{|\pi|-j}},$$

which in the case of a single job complies with Theorem 8.10.

## 8.3 Bibliographic Notes

Papers by Shafransky (1978) and Melnikov and Shafransky (1980) must be seen as historically the first ones that introduce not only scheduling models with start-time-dependent effects, but open up the whole area of scheduling with changing times.

Start-time-dependent effects are probably the most studied among known effects that affect the actual processing time of jobs in a schedule. There are several influential surveys, such as Alidaee and Womer (1999) and Cheng et al. (2004) that review the developments in the area. Reviews of start-time-dependent models with learning effects and deterioration effects are provided in Biskup (2008) and Janiak and Kovalyov (2006), respectively. A systematic exposition of various aspects of scheduling with start-time-dependent effects has been undertaken in the monograph by Gawiejnowicz (2008).

### *8.3.1 General Additive Job-Independent Effects*

We are not aware of any prior work on the models that combine a general additive start-time-dependent effect with a positional effect. However, models that combine a linear additive start-time-dependent effect with a positional effect have been studied before.

Wang (2006) and Yang and Kuo (2009) consider a model of the form (8.19) with time-dependent deterioration and polynomial positional learning, i.e., $g(r) = r^b$, $b < 0$, and prove that each of the problems of minimizing the makespan and the total completion time is solvable by the SPT rule. Theorem 8.9 is a generalization of the results presented in these papers.

Yang (2010) consider a model of the form (8.20) with time-dependent learning and polynomial positional deterioration, i.e., $g(r) = r^b$, $b > 0$, and prove that the LPT rule is optimal for the problem of minimizing the makespan. Theorem 8.10 is a generalization of this result. For the problem of minimizing the total completion time, Yang (2010) reduces the problem to minimizing (8.22), so that the results of Theorem 8.11 are applicable.

A pure additive start-time-dependent effect (8.3) is introduced by Melnikov and Shafransky (1980). For the pure effect, simplified versions of Theorems 8.1 and 8.4 follow from Melnikov and Shafransky (1980) (in the case of the makespan). For minimizing the total completion time, an analogue of Theorem 8.1 is formulated as Theorem 6.146 in the book Gawiejnowicz (2008). However, the provided sketch of its proof refers to a statement similar to Theorem 2.5 on minimization of the sum of products, and it remains unclear how that statement may be used.

The results presented in Kuo and Yang (2007) should be seen as a special case of Theorem 8.1 applied to a particular function $f(\tau) = \sum_{i=1}^{m} \lambda_i \tau^{r_i}$, where for some integer $m$, the values $\lambda_i$ and $r_i$, $1 \leq i \leq m$, are given non-negative constants.

### *8.3.2 Linear Additive Job-Dependent Effects*

The linear additive job-dependent deterioration effect (8.13) has been introduced by Shafransky (1978), while its learning counterpart (8.14) together with the assumption (8.15) is due to Ho et al. (1993) and Ng et al. (2002).

For problem $1\left|p_j(\tau) = p_j + a_j\tau\right|C_{\max}$, analogues of Theorem 8.6 have been independently proved by many authors, including Shafransky (1978), Tanaev et al. (1984), Gupta and Gupta (1988), Brownie and Yechiali (1990) (in the stochastic settings), and Gawiejnowicz and Pankowska (1995). In Sect. 6.1.3 of the book by Gawiejnowicz (2008), several alternative proofs of the part of Theorem 8.6 that concerns problem $1\left|p_j(\tau) = p_j + a_j\tau\right|C_{\max}$ with a deterioration effect are presented.

An analogue of Theorem 8.6 for problem $1\left|p_j(\tau) = p_j - a_j\tau\right|C_{\max}$ with a learning effect is given in Ho et al. (1993), where it is interpreted as a feasibility problem, in which all jobs have a common deadline.

The presented proof of Theorem 8.8 on the NP-hardness of problem $1\left|p_j(\tau) = p_j + a_j\tau\right|\sum w_jC_j$ is based on Bachman et al. (2002).

The fact that problem $1\left|p_j(\tau) = p_j + a_j\tau\right|\sum C_j$ does not admit a 1-priority function is established in Gordon et al. (2008) see Theorem 8.7. It is stated in Cheng et al. (2004) that problem $1\left|p_j(\tau) = p_j + a_j\tau\right|\sum C_j$ is NP-hard in the strong sense, provided that the makespan is bounded by a given value (or, equivalently, all jobs have to be complete by a common deadline). Various properties of an optimal schedule for problem $1\left|p_j(\tau) = 1 + a_j\tau\right|\sum C_j$ are established in Mosheiov (1991), where it is proved that an optimal sequence is $V$-shaped with respect to the $a_j$ values. However, still the complexity status of problem $1\left|p_j(\tau) = p_j + a_j\tau\right|\sum C_j$ remains unknown, even if $p_j = 1,\ j \in N$, which is confirmed in Cheng et al. (2004), Gawiejnowicz (2008), and Janiak and Kovalyov (2006). Kubale and Ocetkiewicz (2009) show that problem $1\left|p_j(\tau) = p_j + a_j\tau\right|\sum C_j$ is solvable in polynomial time, provided that all deterioration rates $a_j$ are different and the difference between any two rates is bounded from below. Ocetkiewicz (2009) presents an algorithm that behaves as an FPTAS for problem $1\left|p_j(\tau) = p_j + a_j\tau\right|\sum C_j$, provided that deterioration rates are different and no smaller than a certain constant.

### *8.3.3 Linear Additive Job-Independent Effects*

Most of the presented results for single machine problems with an additive linear job-independent effect are special cases of more general results. For problem $1\left|p_j(\tau) = p_j - a\tau\right|\sum C_j$, Corollary 8.1 is due to Ng et al. (2002) (notice the misleading words "deteriorating jobs" in the title of that paper which studies only learning models). Their proof is based on the equality (8.29); however, no working is provided in the original paper that demonstrates that the equality holds. An analogue of Corollary 8.1 presented in the book Gawiejnowicz (2008) is not accompanied by a detailed proof, either.

For problem $1\left|p_j(\tau) = p_j + a\tau\right|\sum w_j C_j$ of minimizing the sum of the weighted completion times, Mosheiov (1996) gives a characterization of an optimal permutation, provided that the weights $w_j$ are proportionate to the normal processing times $p_j$; i.e., the ratio $p_j/w_j$ is constant. An optimal permutation is unique and is an alternating $\Lambda$-shaped sequence with respect to normal processing times $p_j$; i.e., if the jobs are numbered in accordance with the SPT rule, then the optimal sequence is $(1, 3, 5, \ldots, n, \ldots, 4, 2)$. Such a permutation can be found in $O(n \log n)$ time, but not by a single sorting of jobs. This implies that problem $1\left|p_j(\tau) = p_j + a\tau\right|\sum w_j C_j$ under the additional condition on the constant ratio $p_j/w_j$ does not admit a 1-priority function.

### 8.3.4 Scheduling with Precedence Constraints

Theorem 8.12 is proved in Shafransky (1978) (see also Tanaev et al. (1984)). Theorem 8.13 is due to Gordon et al. (2008).

## References

Alidaee B, Womer NK (1999) Scheduling with time dependent processing times: review and extensions. J Oper Res Soc 50:711–720

Bachman A, Janiak A, Kovalyov MY (2002) Minimizing the total weighted completion time of deteriorating jobs. Inf Process Lett 81:81–84

Biskup D (2008) A state-of-the-art review on scheduling with learning effects. Eur J Oper Res 188:315–329

Browne S, Yechiali U (1990) Scheduling deteriorating jobs on a single processor. Oper Res 38:495–498

Cheng TCE, Ding Q, Lin BMT (2004) A concise survey of scheduling with time-dependent processing times. Eur J Oper Res 152:1–13

Gawiejnowicz S (2008) Time-dependent scheduling. Springer, Berlin

Gawiejnowicz S, Pankovska L (1995) Scheduling jobs with varying processing times. Inf Process Lett 54:175–178

Gordon VS, Potts CN, Strusevich VA, Whitehead JD (2008) Single machine scheduling models with deterioration and learning: handling precedence constraints via priority generation. J Sched 11:357–370

Gupta JND, Gupta SK (1988) Single facility scheduling with nonlinear processing times. Comput Ind Engine 14:387–393

Ho KI-J, Leung JY-T, Wei W-D (1993) Complexity of scheduling tasks with time dependent execution times. Inf Process Lett 48:315–320

Janiak A, Kovalyov MY (2006) Scheduling deteriorating jobs. In: Janiak A (ed) Scheduling in computer and manufacturing systems. Wydawnictwa Komuniikacji i Łączności, Warsaw, pp 12–25

Kubale M, Ocetkiewicz KM (2009) A new optimal algorithm for a time-dependent scheduling problem, Contr Cybern 38:713–21

Kuo W-H, Yang D-L (2007) Single-machine scheduling problems with start-time dependent processing time. Comp Math Appl 53:1658–1664

Melnikov OI, Shafransky YM (1980) Parametric problem of scheduling theory. Cybernetics 15:352–357

Mosheiov G (1991) $V$-shaped policies for scheduling deteriorating jobs. Oper Res 39:979–991

Mosheiov G (1996) $\Lambda$-shaped policies to schedule deteriorating jobs. J Oper Res Soc 47:1184–1191

Ng CT, Cheng CTE, Bachman A, Janiak A (2002) Three scheduling problems with deteriorating jobs to minimize the total completion time. Inf Process Lett 81:327–333

Ocetkiewicz KM (2009) A FPTAS for minimizing total completion time in a single machine time-dependent scheduling problem. Eur J Oper Res 203:316–320

Shafransky YM (1978) On optimal sequencing for deterministic systems with a tree-like partial order. Vestsi Akad Navuk BSSR, Ser Fiz-Mat Navuk, vol 2, 120 (in Russian)

Tanaev VS, Gordon VS, Shafransky YM (1984) Scheduling theory. Single-stage systems. Moscow, Nauka (in Russian); Kluwer, Dordrecht, 1994 (in English)

Wang JB (2006) A note on scheduling problems with learning effects and deteriorating jobs. Int J Syst Sci 37:827–832

Yang SJ (2010) Single-machine scheduling problems with both start-time dependent learning and position dependent aging effects under deteriorating maintenance consideration. Appl Math Comput 217:3321–3329

Yang DL, Kuo WH (2009) Single-machine scheduling with both deterioration and learning effects. Ann Oper Res 172:315–327

# Chapter 9
# Scheduling with Pure and Combined Multiplicative Start-Time-Dependent Effects

In this chapter, we study the problems of minimizing the makespan and the total completion time on a single machine, provided that the actual processing times of the jobs are subject to a special form of a start-time-dependent effect. We also study effects in which such a start-time-dependent effect is combined with a positional effect.

For a job $j \in N = \{1, 2, \ldots, n\}$, its normal processing time $p_j$ is given. Suppose that the jobs are processed on a single machine in accordance with a permutation $\pi = (\pi(1), \ldots, \pi(n))$. As defined in Sect. 6.2, if the actual processing time of a job depends on its normal processing time and its start time in the schedule, we call such an effect start-time-dependent.

Following Sect. 6.2, we distinguish between two types of a start-time effect: additive and multiplicative. Let $p_j(\tau)$ denote the actual processing time of job $j \in N$ that starts at time $\tau \geq 0$. Then, under a general additive effect, we define

$$p_j(\tau) = p_j + f_j(\tau), \tag{9.1}$$

while under a general multiplicative effect, we define

$$p_j(\tau) = p_j f_j(\tau), \tag{9.2}$$

where $f_j$ is a job-dependent function of start time.

Scheduling problems with an additive start-time-dependent effect are considered in Chap. 8. In this chapter, we focus on the problems with a multiplicative effect of the form (9.2). In the case of *learning*, $f_j : [0, +\infty) \to (0, 1]$ is a non-increasing function, while in the case of *deterioration*, $f_j : [0, +\infty) \to [1, +\infty)$ is a non-decreasing function.

We also study multiplicative models in which the function $f(\tau)$ is job-independent, so that the multiplicative effect is of the form

V.A. Strusevich and K. Rustogi, *Scheduling with Times-Changing Effects and Rate-Modifying Activities*, International Series in Operations Research & Management Science 243, DOI 10.1007/978-3-319-39574-6_9

$$p_j(\tau) = p_j f(\tau). \tag{9.3}$$

In fact, we shall study a more general form of the effect (9.3) that combines a job-independent start-time-dependent effect with a general job-independent positional effect, so that the actual processing time of job $j$ that is scheduled in the $r$th position and starts at time $\tau$ is given by

$$p_j(\tau; r) = p_j f(\tau) g(r), \tag{9.4}$$

where

- $f$ is a continuous differentiable function, common to all jobs, that depends on the start time $\tau$ of the job and takes positive values;
- array $g(r)$, $1 \leq r \leq n$, is a monotone sequence of positional factors and defines a positional effect.

It is assumed that $f(0) = 1$ and $g(1) = 1$, which guarantee that for the job which is the first in the processing sequence, the actual processing time is equal to its normal time.

As adopted throughout this book, if job $j$ is sequenced in position $\pi(r)$ of permutation $\pi$, its completion time is denoted either by $C_j(\pi)$ or by $C_{\pi(r)}$, whichever is more convenient.

Many problems from the considered range admit a solution by a priority rule. Recall that if the jobs are numbered in accordance with the LPT rule, then

$$p_1 \geq p_2 \geq \cdots \geq p_n, \tag{9.5}$$

while if they are numbered in accordance with the SPT rule, then

$$p_1 \leq p_2 \leq \cdots \leq p_n. \tag{9.6}$$

This chapter is structured as follows. Section 9.1 studies single machine problems with no precedence constraints under various effects, including a combined effect (9.4) and linear job-dependent and job-independent effects. Section 9.2 considers problems with series-parallel precedence constraints, mainly under pure start-time-dependent linear effects.

## 9.1 Scheduling Independent Jobs

In this section, we address various versions of single machine scheduling problems, provided that no precedence constraint is imposed on the set of jobs and the actual processing times are either subject to a combined effect of the form (9.4) or a multiplicative job-dependent start-time-dependent effect of the form (9.2).

### *9.1.1 Job-Independent Combined Multiplicative Nonlinear Effects*

In this section, we provide results for a combined effect given by (9.4). For the combined effect (9.4), the problems of minimizing a regular objective function $\Phi$ on a single machine are denoted by $1\left|p_j(\tau) = p_j f(\tau) g(r)\right|\Phi$.

For our reasoning, the behavior of the following auxiliary function

$$\varphi(t) = (1-\lambda) f(X) + \lambda\mu f(X+t) - \mu f(X+\lambda t), \tag{9.7}$$

is important.

**Lemma 9.1** *For function $\varphi(t)$ defined by (9.7) with $X \geq 0$, the inequality*

$$\varphi(t) \leq 0$$

*holds for all $t \geq 0$ if*

***(a)*** $\lambda \geq 1$, $0 < \mu \leq 1$ *and function $f$ is convex on* $[0, +\infty)$, *or*
***(b)*** $0 < \lambda \leq 1$, $\mu \geq 1$ *and function $f$ is concave on* $[0, +\infty)$.

*Proof* Notice that

$$\varphi(0) = (1-\lambda) f(X) + \lambda\mu f(X) - \mu f(X) = (\mu - 1)(\lambda - 1) f(X),$$

which is non-positive, since the differences $\mu - 1$ and $\lambda - 1$ are of the opposite signs.

To prove the lemma, we show that its conditions (a) and (b) imply that function $\varphi(t)$ is monotone non-increasing on $[0, +\infty)$. Differentiating function $\varphi(t)$, we obtain

$$\varphi'(t) = \lambda\mu\big(f'(X+t) - f'(X+\lambda t)\big).$$

Under conditions (a), $t \leq \lambda t$ and since $f$ is convex, i.e., $f'$ is non-decreasing, we have that $\varphi'(t) \leq 0$. Under conditions (b), $t \geq \lambda t$ and since $f$ is concave, i.e., $f'$ is non-increasing, we again have $\varphi'(t) \leq 0$. This means that in any case $\varphi'(t)$ is monotone non-increasing, i.e.,

$$\varphi(t) \leq \varphi(0) = f(X)(\mu - 1)(\lambda - 1) < 0,$$

as required. □

Apart from the above result, we need the Lagrange mean value theorem used in Sect. 8.1.1. For completeness, we reproduce it below.

**Theorem 9.1** *If a function $f$ is continuous on a closed interval $[a, b]$, where $a < b$, and differentiable on the open interval $(a, b)$, then there exists a point $\zeta \in (a, b)$ such that*

$$f(a) - f(b) = f'(\zeta)(a - b).$$

We start with an effect that combines a start-time-dependent effect with positional learning. Define $p_{\max} := \max\{\mathrm{p_j} | \mathrm{j} \in \mathrm{N}\}$.

**Theorem 9.2** *Let $\pi = (\pi(1), \ldots, \pi(n))$ be a permutation, in which two jobs $u$ and $v$ such that*

$$p_u > p_v, \tag{9.8}$$

*occupy two consecutive positions $r$ and $r + 1$, i.e., $u = \pi(r)$ and $v = \pi(r + 1)$. Let permutation $\pi'$ be obtained from $\pi$ by swapping the jobs $u$ and $v$. Then for a single machine problem with a combined effect (9.4) the inequality*

$$C_{\pi(h)} \geq C_{\pi'(h)} \tag{9.9}$$

*holds for all $h$, $1 \leq h \leq n$, provided that*

***(i)*** *function $f$ is convex on $[0, +\infty)$,*
***(ii)*** *the array $g(r)$, $1 \leq r \leq n$, is non-increasing, i.e., it follows*

$$1 = g(1) \geq g(2) \geq \cdots \geq g(n), \tag{9.10}$$

***(iii)*** *either $f'(0) \geq -1/p_{\max}$ or function $f$ is non-decreasing on $[0, +\infty)$.*

*Proof* We represent permutations $\pi$ and $\pi'$ as $\pi = (\pi_1, u, v, \pi_2)$ and as $\pi' = (\pi_1, v, u, \pi_2)$, respectively, where $\pi_1$ and $\pi_2$ are subsequences of jobs that precede job $u$ and follow job $v$ in permutation $\pi$, respectively.

We present the proof assuming that both sequences $\pi_1$ and $\pi_2$ are non-empty; otherwise, the corresponding part of the proof can be skipped.

The actual processing times and the completion times of all jobs in sequence $\pi_1$ are not affected by the swap of jobs $u$ and $v$, i.e., (9.9) holds as equality for each $h$, $1 \leq h \leq r - 1$.

Define $X$ as the completion time of the job in the $(r - 1)$th position in sequence $\pi$ (or, equivalently, in $\pi'$), i.e., $X := C_{\pi(r-1)} = C_{\pi'(r-1)}$. For $h = r$, we derive that

$$C_{\pi(r)} = C_u(\pi) = X + p_u f(X) g(r);$$
$$C_{\pi'(r)} = C_v\left(\pi'\right) = X + p_v f(X) g(r).$$

Due to (9.8), we see that inequality (9.9) holds for $h = r$.

For $h = r + 1$, we derive that

$$C_{\pi(r+1)} = C_v(\pi) = X + p_u f(X) g(r) + p_v f(X + p_u f(X) g(r)) g(r + 1);$$
$$C_{\pi'(r+1)} = C_u\left(\pi'\right) = X + p_v f(X) g(r) + p_u f(X + p_v f(X) g(r)) g(r + 1).$$

Define

$$\Delta := C_{\pi'(r+1)} - C_{\pi(r+1)}.$$

We show that $\Delta \le 0$. Writing out the actual processing times of jobs $u$ and $v$ in permutations $\pi$ and $\pi'$, we obtain

$$\begin{aligned}\Delta = p_v f(X)g(r) + p_u f(X + p_v f(X)g(r))g(r+1) - p_u f(X)g(r) \\ - p_v f(X + p_u f(X)g(r))g(r+1).\end{aligned}$$

Define

$$\lambda := p_u/p_v \tag{9.11}$$

and

$$\mu := \frac{g(r+1)}{g(r)} \tag{9.12}$$

and rewrite the expression for $\Delta$ as

$$\begin{aligned}\Delta = p_v g(r)((1-\lambda)f(X) + \lambda\mu f(X + p_v f(X)g(r)) \\ - \mu p_v f(X + \lambda p_v f(X)g(r))).\end{aligned}$$

Applying (9.7) with $t = p_v f(X)g(r)$, we deduce that

$$\Delta = p_v g(r)\varphi(p_v f(X)g(r)). \tag{9.13}$$

Since $\lambda > 1$, $\mu < 1$, and $f$ is convex, it follows that condition (a) of Lemma 9.1 is valid. Thus, $\varphi(p_v f(X)g(r)) \le 0$ and $\Delta \le 0$, as required.

The rest of the theorem is proved by induction with respect to $h$. Assume that the inequality (9.9) holds for all $h$, $1 \le h \le q-1$, where $r+2 \le q \le n$. We prove that it holds for $h = q$.

Let $x$ be the job scheduled in position $q$, i.e., $x = \pi(q) = \pi'(q)$. We deduce that

$$\begin{aligned}C_{\pi(q)} &= C_x(\pi) = C_{\pi(q-1)} + p_x f\big(C_{\pi(q-1)}\big)g(q);\\ C_{\pi'(q)} &= C_x\big(\pi'\big) = C_{\pi'(q-1)} + p_x f\big(C_{\pi'(q-1)}\big)g(q),\end{aligned}$$

where by the induction assumption $C_{\pi(q-1)} \ge C_{\pi'(q-1)}$.

If $C_{\pi(q-1)} = C_{\pi'(q-1)}$, then it follows that $C_{\pi(q)} = C_{\pi'(q)}$, and (9.9) holds as equality for $h = q$.

Assume now that $C_{\pi(q-1)} > C_{\pi'(q-1)}$. If function $f$ is non-decreasing, then $f\big(C_{\pi(q-1)}\big) \ge f\big(C_{\pi'(q-1)}\big)$ and (9.9) holds for $h = q$.

In the remainder of this proof, assume that $f$ is not necessarily non-decreasing, so that the theorem's condition (iii) means that $f'(0) \ge -1/p_{\max}$. Since the derivative $f'$ is non-decreasing, we deduce that $f'(\tau) \ge -1/p_{\max}$ for all $\tau \in [0, +\infty)$.

By Theorem 9.1, there exists a point $\zeta \in \big[C_{\pi'(q-1)}, C_{\pi(q-1)}\big]$ such that

$$f\big(C_{\pi(q-1)}\big) - f\big(C_{\pi'(q-1)}\big) = f'(\zeta)\big(C_{\pi(q-1)} - C_{\pi'(q-1)}\big).$$

Then

$$\begin{aligned}C_{\pi(q)} - C_{\pi'(q)} &= \left(C_{\pi(q-1)} - C_{\pi'(q-1)}\right) + p_x\left(f\left(C_{\pi(q-1)}\right) - f\left(C_{\pi'(q-1)}\right)\right)g(q)\\ &= \left(C_{\pi(q-1)} - C_{\pi'(q-1)}\right) + p_x g(q) f'(\zeta)\left(C_{\pi(q-1)} - C_{\pi'(q-1)}\right).\end{aligned}$$

Since $g(q) \le 1$, we deduce from $f'(\zeta) \ge -1/p_{\max}$ that $1 + p_x g(q) f'(\zeta) \ge 0$. The theorem is fully proved. □

For single machine problems, to minimize a regular objective function $\Phi(\pi)$ under a combined effect (9.4), Theorem 9.2 sets conditions on functions $f$ and $g$ that define the effect, which guarantee that in a sequence that minimizes $\Phi(\pi)$ the jobs may be arranged in such a way that a job with a larger normal processing time is not followed by a job with a smaller normal processing time.

**Theorem 9.3** *For problem* $1\left|p_j(\tau; r) = p_j f(\tau) g(r)\right|\Phi$ *with* $\Phi \in \left\{C_{\max}, \sum C_j^z, \xi C_{\max} + \eta \sum C_j^z\right\}$ *under an effect (9.4) that combines a start-time dependent effect with a positional learning effect, an optimal permutation can be found in* $O(n \log n)$ *time by sorting the jobs in accordance with the SPT rule, provided that the conditions of Theorem 9.2 hold.*

Reformulating Theorem 9.3, we conclude that problem $1\left|p_j(\tau; r) = p_j f(\tau) g(r)\right|\Phi$, where $\Phi \in \left\{C_{\max}, \sum C_j^z, \xi C_{\max} + \eta \sum C_j^z\right\}$ with an effect (9.4) admits the 1-priority $\omega(j) = 1/p_j$, provided that $f$ and $g$ satisfy the conditions of Theorem 9.2.

Notice that the analysis of the effect (9.4) defined by a concave function $f$ is not fully symmetric to that presented in Theorem 9.2, and only a less general statement can be proved.

**Theorem 9.4** *Let* $\pi = (\pi(1), \ldots, \pi(n))$ *be a permutation, in which two jobs* $u$ *and* $v$ *such that*

$$p_u < p_v,$$

*occupy two consecutive positions* $r$ *and* $r+1$*, i.e.,* $u = \pi(r)$ *and* $v = \pi(r+1)$*. Let permutation* $\pi'$ *be obtained from* $\pi$ *by swapping the jobs* $u$ *and* $v$*. Then for problem* $1\left|p_j(\tau; r) = p_j f(\tau) g(r)\right|C_{\max}$ *with an effect (9.4) the inequality* $C_{\max}\left(\pi'\right) \le C_{\max}(\pi)$ *holds, provided that*

***(i)*** *function* $f$ *is concave and non-decreasing on* $[0, +\infty)$*,*
***(ii)*** *the array* $g(r)$*,* $1 \le r \le n$*, is non-decreasing, i.e., it follows*

$$1 = g(1) \le g(2) \le \cdots \le g(n). \tag{9.14}$$

*Proof* As in the proof of Theorem 9.2, we represent permutations $\pi$ and $\pi'$ as $\pi = (\pi_1, u, v, \pi_2)$ and as $\pi' = (\pi_1, v, u, \pi_2)$, respectively, where $\pi_1$ and $\pi_2$ are subsequences of jobs that precede job $u$ and follow job $v$ in permutation $\pi$, respectively.

We present the proof assuming that both sequences $\pi_1$ and $\pi_2$ are non-empty; otherwise, the corresponding part of the proof can be skipped.

The actual processing times and the completion times of all jobs in sequence $\pi_1$ are not affected by the swap of jobs $u$ and $v$. Define $X$ as the completion time of the job in the $(r-1)$th position in sequence $\pi$ (or, equivalently, in $\pi'$), i.e., $X = C_{\pi(r-1)} = C_{\pi'(r-1)}$. It is clear that $C_{\pi(r)} = X + p_u f(X)g(r)$ and $C_{\pi'(r)} = X + p_v f(X)g(r)$. Notice that $p_u < p_v$, which implies that $C_{\pi(r)} < C_{\pi'(r)}$. This explains why under the conditions of Theorem 9.4 it is not possible to prove a more general statement, similar to Theorem 9.2.

We now prove that $\Delta = C_{\pi'(r+1)} - C_{\pi(r+1)} \le 0$. As in the proof of Theorem 9.2, $\Delta$ can be rewritten as (9.13), where $\lambda$ and $\mu$ are defined by (9.11) and (9.12), respectively.

Since $\lambda < 1$, $\mu \ge 1$, and $f$ is concave by the theorem's condition, it follows that condition (b) of Lemma 9.1 is valid. Thus, $\varphi(p_v f(X)g(r)) \le 0$ and $\Delta \le 0$, as required.

Notice that function $f$ is non-decreasing, as in one of the conditions (iii) of Theorem 9.2. Following the proof of that theorem, we deduce that the inequality $C_{\pi'(q)} \le C_{\pi(q)}$ holds for each $q$, $r+2 \le q \le n$. In particular, $C_{\pi'(n)} \le C_{\pi(n)}$, i.e., $C_{\max}(\pi') \le C_{\max}(\pi)$, as required. □

For a single machine problem with a combined effect (9.4), Theorem 9.4 sets conditions on functions $f$ and $g$, which guarantee that in a sequence that minimizes the makespan $C_{\max}(\pi)$, the jobs may be arranged in such a way that a job with a larger normal processing time is not followed by a job with a smaller normal processing time.

**Theorem 9.5** *For problem* $1\left|p_j(\tau;r) = p_j f(\tau)g(r)\right|C_{\max}$, *with a multiplicative start-time dependent effect (9.4) an optimal permutation can be found in* $O(n\log n)$ *time by sorting the jobs in accordance with the LPT rule, provided that functions* $f$ *and* $g$ *satisfy the conditions of Theorem 9.4.*

Reformulating Theorem 9.5, we conclude that problem $1\left|p_j(\tau;r) = p_j f(\tau)g(r)\right| C_{\max}$ with an effect (9.4) admits the 1-priority $\omega(j) = p_j$, provided that functions $f$ and $g$ satisfy the conditions of Theorem 9.4.

Notice that the proof of Theorem 9.5 demonstrates that the inequality (9.9) holds for each $h$ other than $r$, while $C_{\pi(r)} < C_{\pi'(r)}$. This fact does not allow us to derive any conclusions regarding the status of the problem of minimizing the total completion time for an effect that satisfies the conditions of Theorem 9.4. The status of this problem remains open even if $g(r) = 1$, $1 \le r \le n$.

Below we present a counterexample that demonstrates that for problem $1\left|p_j(\tau) = p_j f(\tau)\right|\sum C_j$ with a function $f$ that satisfies the conditions of Theorem 9.4 neither the LPT sequence nor the SPT sequence is optimal.

*Example 9.1* Consider an instance of a single machine problem to minimize the sum of the completion times under effect (9.3) with the function $f(\tau) = (1+\tau)^{\frac{1}{2}}$. It is clear that function $f$ is concave and increasing on $[0, +\infty)$, i.e., the conditions of Theorem 9.4 hold. There are 3 jobs with the normal processing times listed below

$$p_1 = 6, \ p_2 = 7, \ p_3 = 9.$$

For this instance, the total completion time $\sum C_j$ is minimized neither by the SPT sequence, nor by the LPT sequence. The corresponding computations, accurate up to three decimal places, are shown in Table 9.1. We see that permutation (2, 1, 3) delivers a smaller value of the total completion time than that produced by the SPT and the LPT sequences.

In Sect. 9.1.2, we present another example of a similar nature.

The results on single machine scheduling problem with an effect that combines a multiplicative time-dependent effect with a positional effect are summarized in Table 9.2. Here, in the first and the third columns, we use symbols $\nearrow$ and $\searrow$ to indicate whether the corresponding function or arrays are non-decreasing or non-increasing, respectively. Additionally, we write $g = 1$ if $g(r) = 1, 1 \le r \le n$.

**Table 9.1** Computations for Example 9.1

| | $\pi = (1, 2, 3)$ [SPT] | $\pi = (3, 2, 1)$ [LPT] | $\pi = (2, 1, 3)$ |
|---|---|---|---|
| $C_{\pi(1)}$ | 6.000 | 9.000 | 7.000 |
| $C_{\pi(2)}$ | 24.520 | 31.136 | 23.971 |
| $C_{\pi(3)} = C_{\max}(\pi)$ | 69.986 | 65.149 | 68.944 |
| $\sum C_{\pi(j)}$ | 100.506 | 105.285 | 99.914 |

**Table 9.2** Results for scheduling on a single machine with a combined effect (9.4)

| Condition on $f$ | Condition on $f'$ | Condition on $g$ | Objective | Rule | Statement |
|---|---|---|---|---|---|
| $f$ convex $\nearrow$ | – | $g \searrow$ | $C_{\max}$ | SPT | Theorem 9.3 |
| $f$ convex $\nearrow$ | – | $g \searrow$ | $\sum C_j^z$ | SPT | Theorem 9.3 |
| $f$ convex $\nearrow$ | – | $g \searrow$ | $\xi C_{\max} + \eta \sum C_j^z$ | SPT | Theorem 9.3 |
| $f$ convex | $f'(0) \ge -1/p_{\max}$ | $g \searrow$ | $C_{\max}$ | SPT | Theorem 9.3 |
| $f$ convex | $f'(0) \ge -1/p_{\max}$ | $g \searrow$ | $\sum C_j^z$ | SPT | Theorem 9.3 |
| $f$ convex | $f'(0) \ge -1/p_{\max}$ | $g \searrow$ | $\xi C_{\max} + \eta \sum C_j^z$ | SPT | Theorem 9.3 |
| $f$ concave $\nearrow$ | – | $g \nearrow$ | $C_{\max}$ | LPT | Theorem 9.5 |
| $f$ concave $\nearrow$ | – | $g = 1$ | $\sum C_j$ | Open | Example 9.1 |

## *9.1.2 Job-Independent Combined Multiplicative Polynomial Effects*

In this subsection, we consider single machine scheduling problems with an effect that combines a multiplicative start-time-dependent polynomial effect of the form

$$f(\tau) = (1 + b\tau)^A \tag{9.15}$$

or

$$f(\tau) = (1 - b\tau)^A, \tag{9.16}$$

with a positional effect given by an array $g(r)$, $1 \le r \le n$. In both (9.15) and (9.16), $b$ is a positive job-independent rate. For the effect (9.16), we additionally assume that the inequality

$$b\tau < 1, \tag{9.17}$$

holds for all $\tau > 0$. This inequality guarantees that the actual processing times of the jobs will remain non-negative. Recall that an assumption of a similar nature has been made regarding learning effects studied in Chap. 8.

A direct application of Theorems 9.3 and 9.5 yields the following statement.

**Theorem 9.6** *Given problem* $1\left|p_j(\tau; r) = p_j(1 + b\tau)^A g(r)\right|\Phi$, *where* $\Phi \in \left\{C_{\max}, \sum C_j^z, \xi C_{\max} + \eta \sum C_j^z\right\}$, *with an effect of the form (9.4) that combines a start-time-dependent effect (9.15) with a positional learning effect defined by an array* $g(r)$, $1 \le r \le n$, *an optimal permutation can be found in* $O(n \log n)$ *time by sorting the jobs in accordance with the SPT rule, provided that (9.10) holds and either* $A \ge 1$ *or* $-1/(bp_{\max}) \le A < 0$. *Besides, for problem* $1\left|p_j(\tau; r) = p_j(1 + b\tau)^A g(r)\right|C_{\max}$, *with a positional deterioration effect, an optimal permutation can be found in* $O(n \log n)$ *time by sorting the jobs in accordance with the LPT rule, provided that (9.14) holds and* $0 < A < 1$.

*Proof* The proof is by checking the conditions of Theorems 9.3 and 9.5.

It is clear that for $A > 0$ (a time-dependent deterioration effect), function $f(\tau) = (1 + b\tau)^A$ is increasing. Its second-order derivative $f'' = A(A - 1)b^2(1 + b\tau)^{A-2}$ is non-negative for $A \ge 1$, so that function $f$ is convex, i.e., the conditions of Theorem 9.2 hold and Theorem 9.3 applies. On the other hand, if $0 < A < 1$, then $f''$ is negative, so that function $f$ is concave, i.e., the conditions of Theorem 9.4 hold and Theorem 9.5 applies.

For $A < 0$ (a time-dependent learning effect), function $f(\tau) = (1 + b\tau)^A$ is decreasing. Its second-order derivative $f''$ is positive, so that function $f$ is convex. Since $f'(0) = Ab$, it follows that for $Ab \ge -1p_{\max}$, the conditions of Theorem 9.2 hold and Theorem 9.3 applies. $\square$

Notice that Theorem 9.6 guarantees that no priority rule exists for problems $1\left|p_j(\tau; r) = p_j(1 + b\tau)^A g(r)\right|C_{\max}$ and $1\left|p_j(\tau; r) = p_j(1 + b\tau)^A g(r)\right|\sum C_j$,

**Table 9.3** Computations for Example 9.2

| | $\pi = (1, 2, 3)$ [SPT] | $\pi = (2, 1, 3)$ |
|---|---|---|
| $C_{\pi(1)}$ | 10.0000 | 11.0000 |
| $C_{\pi(2)}$ | 10.0909 | 11.0694 |
| $C_{\pi(3)} = C_{\max}(\pi)$ | 30.4146 | 28.2314 |
| $\sum C_{\pi(j)}$ | 50.5055 | 50.3008 |

provided that $A < -1/bp_{\max}$. Example 9.2 below demonstrates that for effect (9.15) with $g(r) = 1$, $1 \le r \le n$, and $A = -2$, an optimal permutation is obtained neither by the SPT rule nor by the LPT rule.

*Example 9.2* Consider an instance of a single machine problem to minimize the sum of completion times under effect (9.3) with the function $f(\tau) = (1 + \tau)^{-2}$. It is clear that function $f$ is concave and is decreasing on $[0, +\infty)$, i.e., neither the conditions of Theorem 9.3 nor those of Theorem 9.5 are satisfied. There are 3 jobs with the normal processing times listed below

$$p_1 = 10,\ p_2 = 11,\ p_3 = 2500.$$

For this instance, the total completion time $\sum C_j$ is minimized by neither the SPT sequence, nor the LPT sequence. The corresponding computations, accurate up to four decimal places, are shown in Table 9.3. The computation for the LPT sequence is not presented, since in this sequence job 3 is scheduled first to complete at time 2500. We see that permutation (2, 1, 3) delivers smaller values of both the makespan and the total completion time as compared to those produced by the SPT and the LPT sequences. In fact, (2, 1, 3) is an optimal permutation for both these functions.

For the problems with the effect (9.16), a statement similar to Theorem 9.6 can be proved.

**Theorem 9.7** *Given problem* $1\left|p_j(\tau; r) = p_j(1 - b\tau)^A g(r)\right|\Phi$, *where* $\Phi \in \left\{C_{\max}, \sum C_j^z, \xi C_{\max} + \eta \sum C_j^z\right\}$, *with an effect of the form (9.4) that combines a start-time-dependent effect (9.16) with a positional learning effect given by an array* $g(r)$, $1 \le r \le n$, *an optimal permutation can be found in* $O(n \log n)$ *time by sorting the jobs in accordance with the SPT rule, provided that (9.10) holds and either* $A < 0$ *or* $1 \le A \le 1/(bp_{\max})$.

## 9.1.3 Pure Multiplicative Linear Effects

In this section, we consider single machine scheduling problems with a pure multiplicative linear effect, so that in the main formula (9.2) for non-negative rate

$b_j,\ j \in N$, either

$$f_j(\tau) = 1 + b_j\tau \tag{9.18}$$

(in the case of deterioration) or

$$f_j(\tau) = 1 - b_j\tau \tag{9.19}$$

(in the case of learning). We also consider special cases of the above effects with constant rates, i.e., for $b_j = b,\ j \in N$.

First, observe that the multiplicative effect (9.18) should be seen as a special case of the additive effect (8.13). Indeed, under the effect (8.13), the actual processing time of job $j \in N$ that starts at time $\tau$ is equal to $p_j(\tau) = p_j + a_j\tau$, while under the effect (9.18), we have that $p_j(\tau) = p_j\left(1 + b_j\tau\right) = p_j + p_jb_j\tau$. A similar equivalence is observed between the learning effects (9.19) and (8.14). Thus, the results derived in Sect. 8.1.2 applied with $a_j = p_jb_j$ carry over for the problems with the effects (9.18) and (9.19), respectively. Notice also that for learning effect (9.19) to be meaningful, we adapt the assumption (8.15) to become

$$b_j\left(\sum_{i=1}^{n} p_i - p_j\right) < 1,\ \ 1 \le j \le n, \tag{9.20}$$

which makes the learning rates $b_j$ rather small numbers. See also the earlier used assumption (9.17) for the case of equal $b_j$.

It follows from Sect. 8.1.2 that for a permutation $\pi = (\pi(1), \dots, \pi(n))$, the makespan under the effect (9.18) can be written as

$$C_{\max}(\pi) = C_{\pi(n)} = \sum_{j=1}^{n} p_{\pi(j)} \prod_{i=j+1}^{n} \left(1 + p_{\pi(i)}b_{\pi(i)}\right),$$

while under the effect (9.19)

$$C_{\max}(\pi) = C_{\pi(n)} = \sum_{j=1}^{n} p_{\pi(j)} \prod_{i=j+1}^{n} \left(1 - p_{\pi(i)}b_{\pi(i)}\right).$$

Adapting Theorem 8.6 for $a_j = p_jb_j$, we obtain the following statement.

**Theorem 9.8** *For problem $1\left|p_j(\tau) = p_j\left(1 + b_j\tau\right)\right|C_{\max}$, with a deterioration effect (9.18), an optimal permutation can be found in $O(n \log n)$ time by sorting the jobs in non-increasing order of $b_j$. For problem $1\left|p_j(\tau) = p_j\left(1 - b_j\tau\right)\right|C_{\max}$, with a learning effect (9.19) under the assumption (9.20), an optimal permutation can be found in $O(n \log n)$ time by sorting the jobs in non-decreasing order of $b_j$.*

Reformulating Theorem 9.8 in terms of 1-priorities, we conclude that problem $1\left|p_j(\tau) = p_j\left(1 + b_j\tau\right)\right|C_{\max}$ admits a 1-priority function $\omega(j) = b_j$, while problem $1\left|p_j(\tau) = p_j\left(1 - b_j\tau\right)\right|C_{\max}$ admits a 1-priority function either $\omega(j) = 1/b_j$ or $\omega(j) = -b_j$. Sequencing jobs in non-increasing order of 1-priorities solves the corresponding problem.

Notice that problem $1\left|p_j(\tau) = p_j\left(1 + b_j\tau\right)\right|\sum C_j$ with $p_j = 1,\ j \in N$, becomes $1\left|p_j(\tau) = 1 + b_j\tau\right|\sum C_j$, i.e., for $a_j = b_j$, problem $1\left|p_j(\tau) = p_j\left(1 + b_j\tau\right)\right|\sum C_j$ coincides with problem $1\left|p_j(\tau) = 1 + a_j\tau\right|\sum C_j$ considered in Chap. 8. The complexity status of the latter problem remains open; see Sect. 8.3.

In the remainder of this section, we consider the case of a constant rate, i.e., we assume that effects (9.18) and (9.19) are applied with $b_j = b$ for all $j \in N$.

**Lemma 9.2** *Let the jobs be processed on a single machine in accordance with a permutation* $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$. *Under the effect (9.18) applied with* $b_j = b,\ j \in N$, *the completion times can be computed by the formula*

$$C_{\pi(k)} = \frac{1}{b}\prod_{i=1}^{k}\left(1 + bp_{\pi(i)}\right) - \frac{1}{b},\ 1 \le k \le n, \tag{9.21}$$

*while for the effect (9.19) applied with* $b_j = b,\ j \in N$, *the completion times can be computed by the formula*

$$C_{\pi(k)} = \frac{1}{b} - \frac{1}{b}\prod_{i=1}^{k}\left(1 - bp_{\pi(i)}\right),\ 1 \le k \le n, \tag{9.22}$$

*Proof* We present the proof of (9.21) for the deterioration effect; the proof for the learning counterpart is similar. The proof is by induction.

By definition, $C_{\pi(1)} = p_{\pi(1)}$. On the other hand, from (9.21) with $k = 1$, we get $C_{\pi(1)} = \frac{1}{b}\left(1 + bp_{\pi(1)}) - \frac{1}{b}\right) = p_{\pi(1)}$. This establishes the basis of induction.

Assume that (9.21) holds for all $k$, where $1 \le k \le q - 1 < n - 1$. We prove that (9.21) holds for $k = q$. By definition and the induction assumption, we have that

$$\begin{aligned} C_{\pi(q)} &= C_{\pi(q-1)} + p_{\pi(q)}(1 + bC_{\pi(q-1)}) = p_{\pi(q)} + C_{\pi(q-1)}\left(1 + bp_{\pi(q)}\right) \\ &= p_{\pi(q)} + \left(\frac{1}{b}\prod_{i=1}^{q-1}\left(1 + bp_{\pi(i)}\right) - \frac{1}{b}\right)\left(1 + bp_{\pi(q)}\right) \\ &= p_{\pi(q)} + \frac{1}{b}\prod_{i=1}^{q}\left(1 + bp_{\pi(i)}\right) - \frac{1}{b}\left(1 + bp_{\pi(q)}\right) = \frac{1}{b}\prod_{i=1}^{q}\left(1 + bp_{\pi(i)}\right) - \frac{1}{b}, \end{aligned}$$

as required. □

Applying (9.21) with $k = n$, we immediately deduce the following statement.

**Corollary 9.1** *For problem* $1\left|p_j(\tau) = p_j(1 + b\tau)\right|C_{\max}$ *the makespan can be written as*

$$C_{\max}(\pi) = \frac{1}{b}\prod_{i=1}^{n}\left(1 + bp_{\pi(i)}\right) - \frac{1}{b},$$

*while for problem* $1\left|p_j(\tau) = p_j(1 - b\tau)\right|C_{\max}$ *the makespan is*

$$C_{\max}(\pi) = \frac{1}{b} - \frac{1}{b}\prod_{i=1}^{n}\left(1 - bp_{\pi(i)}\right).$$

*In either case, the makespan is sequence independent.*

We now pass to considering problems with the objective functions related to the sum of the completion times. First, Theorems 9.6 and 9.7 applied with $A = 1$ yield the following statement.

**Corollary 9.2** *For each problem* $1\left|p_j(\tau) = p_j(1 + b\tau)\right|\Phi$ *and* $1\left|p_j(\tau) = p_j(1 - b\tau)\right|\Phi$*, where* $\Phi \in \left\{\sum C_j^z, \xi C_{\max} + \eta \sum C_j^z\right\}$*, an optimal permutation can be found in* $O(n \log n)$ *time by the SPT rule.*

Reformulating Corollary 9.2 in terms of 1-priorities, we conclude that for $\Phi \in \left\{\sum C_j^z, \xi C_{\max} + \eta \sum C_j^z\right\}$ each problem $1\left|p_j(\tau) = p_j(1 + b\tau)\right|\Phi$ and $\left|p_j(\tau) = p_j(1 - b\tau)\right|\Phi$ admits a 1-priority function either $\omega(j) = 1/p_j$ or $\omega(j) = -p_j$. Sequencing jobs in non-increasing order of 1-priorities solves the corresponding problem.

Unlike many other models with changing times, for the model under consideration, it is possible to deduce a positive result regarding minimization of the weighted sum of completion times.

**Theorem 9.9** *Problem* $1\left|p_j(\tau) = p_j(1 + b\tau)\right|\sum w_j C_j$ *is solvable in* $O(n \log n)$ *time by sorting the jobs in non-increasing order of the ratios* $\frac{w_j(1+bp_j)}{bp_j}$*. Problem* $1\left|p_j(\tau) = p_j(1 - b\tau)\right|\sum w_j C_j$ *is solvable in* $O(n \log n)$ *time by sorting the jobs in non-increasing order of the ratios* $\frac{w_j(1-bp_j)}{bp_j}$*.*

*Proof* We present the proof for problem $1\left|p_j(\tau) = p_j(1 + b\tau)\right|\sum w_j C_j$ with a deterioration effect; the proof for its learning counterpart is similar. Both proofs follow Recipe 2.1.

For an arbitrary permutation of jobs $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$, using (9.21) we can write out the objective function as

$$\sum_{k=1}^{n} w_{\pi(k)} C_{\pi(k)} = \frac{1}{b}\sum_{k=1}^{n} w_{\pi(k)}\left(\prod_{i=1}^{k}\left(1 + bp_{\pi(i)}\right) - 1\right).$$

Define

$$\Psi(\pi) = \left(\prod_{i=1}^{n}(1+bp_i)^{-1}\right)\left(\sum_{k=1}^{n} w_{\pi(k)} \prod_{i=1}^{k}\left(1+bp_{\pi(i)}\right)\right).$$

The problem of minimizing $\sum_{k=1}^{n} w_{\pi(k)}C_{\pi(k)}$ is equivalent to that of minimizing $\Psi(\pi)$. We rewrite the expression for $\Psi(\pi)$ in the form that is similar to the sum of products (2.14):

$$\Psi(\pi) = \sum_{k=1}^{n} w_{\pi(k)} \prod_{i=k+1}^{n} \left(1+bp_{\pi(i)}\right)^{-1}.$$

It follows from Theorem 2.5 that a permutation that minimizes $\Psi(\pi)$ the jobs can be sorted in non-decreasing order of

$$\kappa(j) = \frac{w_j}{\left(1+bp_j\right)^{-1}-1} = -\frac{w_j\left(1+bp_j\right)}{bp_j},$$

as required.

For problem $1\left|p_j(\tau)=p_j(1-b\tau)\right|\sum w_jC_j$, the sequence-dependent term can written as

$$\Psi(\pi) = -\sum_{k=1}^{n} w_{\pi(k)} \prod_{i=k+1}^{n} \left(1-bp_{\pi(i)}\right)^{-1},$$

and a permutation that minimizes $\Psi(\pi)$ the jobs can be sorted in non-decreasing order of

$$\kappa(j) = \frac{-w_j}{\left(1-bp_j\right)^{-1}-1} = \frac{-w_j\left(1-bp_j\right)}{bp_j}.$$

The theorem is proved. □

Reformulating Theorem 9.9 in terms of 1-priorities, we conclude that problem $1\left|p_j(\tau)=p_j(1+b\tau)\right|\sum w_jC_j$ admits a 1-priority function $\omega(j) = \frac{w_j\left(1+bp_j\right)}{bp_j}$, while problem $1\left|p_j(\tau)=p_j(1-b\tau)\right|\sum w_jC_j$ admits a 1-priority function $\omega(j) = \frac{w_j\left(1-bp_j\right)}{bp_j}$. Sequencing jobs in non-increasing order of 1-priorities solves the corresponding problem. Notice that since $b>0$, it can be removed from the denominators in the expressions for the 1-priorities.

The results presented in Sect. 9.1.3 are summarized in Table 9.4.

**Table 9.4** Results for scheduling independent jobs on a single machine with a multiplicative linear start-time-dependent effect

| Effect | Objective | 1-Priority | Statement |
|---|---|---|---|
| $p_j(\tau) = p_j(1 + b_j\tau)$ | $C_{\max}$ | $b_j$ | Theorem 9.8 |
| $p_j(\tau) = p_j(1 - b_j\tau)$ | $C_{\max}$ | $1/b_j$ | Theorem 9.8 |
| $p_j(\tau) = p_j(1 + b_j\tau)$ | $\sum C_j$ | Open | |
| $p_j(\tau) = p_j(1 - b_j\tau)$ | $\sum C_j$ | Open | |
| $p_j(\tau) = p_j(1 + b\tau)$ | $C_{\max}$ | $\forall$ | Corollary 9.1 |
| $p_j(\tau) = p_j(1 - b\tau)$ | $C_{\max}$ | $\forall$ | Corollary 9.1 |
| $p_j(\tau) = p_j(1 + b\tau)$ | $\sum C_j^z$ | $1/p_j$ | Corollary 9.2 |
| $p_j(\tau) = p_j(1 + b\tau)$ | $\xi C_{\max} + \eta \sum C_j^z$ | $1/p_j$ | Corollary 9.2 |
| $p_j(\tau) = p_j(1 - b\tau)$ | $\sum C_j^z$ | $1/p_j$ | Corollary 9.2 |
| $p_j(\tau) = p_j(1 - b\tau)$ | $\xi C_{\max} + \eta \sum C_j^z$ | $1/p_j$ | Corollary 9.2 |
| $p_j(\tau) = p_j(1 + b\tau)$ | $\sum w_j C_j$ | $\frac{w_j(1+bp_j)}{bp_j}$ | Theorem 9.9 |
| $p_j(\tau) = p_j(1 - b\tau)$ | $\sum w_j C_j$ | $\frac{w_j(1-bp_j)}{bp_j}$ | Theorem 9.9 |

## 9.2 Scheduling Under Precedence Constraints

In this section, we consider single machine problems under the multiplicative start-time-dependent effects (9.18) and (9.19) applied with $b_j = b,\ j \in N$. Unlike in Sect. 9.1, here we assume that the jobs of set $N$ are not independent and a precedence relation given by a reduction graph $G = (N, U)$ is imposed over the set $N$ of jobs.

It follows immediately from Corollary 9.1 that for each problem $1\left|p_j(\tau) = p_j(1 + b\tau), prec\right|C_{\max}$ and $1\left|p_j(\tau) = p_j(1 - b\tau), prec\right|C_{\max}$ with arbitrary precedence constraints, any permutation that is feasible with respect to graph $G$ delivers an optimal solution. Problems $1\left|p_j(\tau) = p_j\left(1 + b_j\tau\right), SP - prec\right|C_{\max}$ and $1\left|p_j(\tau) = p_j\left(1 - b_j\tau\right), SP - prec\right|C_{\max}$ are special cases of problems $1\left|p_j(\tau) = p_j + a_j\tau, SP - prec\right|C_{\max}$ and $1\left|p_j(\tau) = p_j - a_j\tau, SP - prec\right|C_{\max}$ with an additive start-time-dependent effect. The latter problems are proved solvable in $O(n \log n)$ time in Sect. 8.2.1.

In the remainder of this section, we focus on problems $1\left|p_j(\tau) = p_j(1 + b\tau), SP - prec\right|\sum w_j C_j$ and $1\left|p_j(\tau) = p_j(1 - b\tau), SP - prec\right|\sum w_j C_j$ with precedence constraints given by a series-parallel graph. We show that in these problems, the objective function is priority-generating, which means that each of these problems is solvable in $O(n \log n)$ time. See Chap. 3 for definitions and main results on scheduling under precedence constraints.

Given a scheduling problem with a multiplicative start-time-dependent effect, let $\pi$ be a (partial) permutation of jobs contained as a subsequence in some schedule. The length of a permutation $\pi$, i.e., the number of elements in $\pi$, is denoted by $|\pi|$.

Assuming that the first job in partial permutation $\pi$ starts at time $t \geq 0$, let $C_{\pi(k)}^{(t)}$ denote the completion time of the job sequenced in the $k$th position. Also, let $C_{\max}(\pi; t)$ denote the maximum completion time of the jobs in $\pi$, and

$$Z(\pi; t) = \sum_{k=1}^{|\pi|} w_{\pi(k)} C_k^{(t)}(\pi)$$

denote the sum of weighted completion times for permutation $\pi$.

We present our reasoning for problem $1\left|p_j(\tau) = p_j(1 + b\tau), SP - prec\right| \sum w_j C_j$ with a deterioration effect; the proof for the learning counterpart is similar.

First, it is straightforward to extend Lemma 9.2 and to prove that for any $t \geq 0$ and any partial permutation $\pi$, the completion times of jobs can be computed as

$$C_{\pi(k)}^{(t)} = \frac{1 + bt}{b} \prod_{i=1}^{k} \left(1 + bp_{\pi(i)}\right) - \frac{1}{b}, \; 1 \leq k \leq |\pi|. \tag{9.23}$$

Indeed, for $k = 1$, we have that $C_{\pi(1)} = t + p_{\pi(1)}(1 + bt) = p_{\pi(1)} + \left(1 + bp_{\pi(1)}\right)t$, while it follows from (9.23) that $C_{\pi(1)} = \frac{1+bt}{b}\left(1 + bp_{\pi(1)}\right) - \frac{1}{b} = p_{\pi(1)} + \left(1 + bp_{\pi(1)}\right)t$, as required. Then, assuming that (9.23) holds for all $k$, $1 \leq k \leq q - 1 \leq |\pi| - 1$, we derive for $k = q$ that

$$\begin{aligned} C_{\pi(q)}^{(t)} &= p_{\pi(q)} + C_{\pi(q-1)}^{(t)}\left(1 + bp_{\pi(q)}\right) \\ &= p_{\pi(q)} + \left(\frac{1 + bt}{b} \prod_{i=1}^{q-1} \left(1 + bp_{\pi(i)}\right) - \frac{1}{b}\right)\left(1 + bp_{\pi(q)}\right) \\ &= p_{\pi(q)} + \frac{1 + bt}{b} \prod_{i=1}^{q} \left(1 + bp_{\pi(i)}\right) - \frac{1}{b}\left(1 + bp_{\pi(q)}\right) = \frac{1 + bt}{b} \prod_{i=1}^{q} \left(1 + bp_{\pi(i)}\right) - \frac{1}{b}. \end{aligned}$$

In particular,

$$C_{\max}(\pi; t) = \frac{1 + bt}{b} \prod_{i=1}^{|\pi|} \left(1 + bp_{\pi(i)}\right) - \frac{1}{b}. \tag{9.24}$$

We prove that function $Z(\pi; t_0)$ is priority-generating for any positive $t_0$. The proof follows Recipe 3.1. Let $\pi^{\alpha\beta} = (\pi_1\alpha\beta\pi_2)$ and $\pi^{\beta\alpha} = (\pi_1\beta\alpha\pi_2)$ be two permutations of all jobs that only differ in the order of the subsequences $\alpha = (\alpha(1), \ldots, \alpha(u))$ of $u$ jobs and $\beta = (\beta(1), \ldots, \beta(v))$ of $v$ jobs. Define

$$\Delta := Z(\pi^{\alpha\beta}; t_0) - Z(\pi^{\beta\alpha}; t_0) = Z(\pi_1\alpha\beta\pi_2; t_0) - Z(\pi_1\beta\alpha\pi_2; t_0).$$

We need to determine a sufficient condition for the inequality $\Delta \leq 0$. Let $t'$ denote the completion time of the last job in permutation $\pi_1$, i.e., $t' = C_{\max}(\pi_1; t_0)$. Then,

$$Z(\pi_1\alpha\beta\pi_2; t_0) = Z(\pi_1; t_0) + Z(\alpha; t') + Z(\beta; C_{\max}(\alpha; t')) + Z(\pi_2; C_{\max}(\alpha\beta; t')),$$

and

$$Z(\pi_1\beta\alpha\pi_2; t_0) = Z(\pi_1; t_0) + Z(\beta; t') + Z(\alpha; C_{\max}(\beta, t')) + Z(\pi_2; C_{\max}(\beta\alpha; t')).$$

Since for the effect under consideration the makespan is sequence-independent, we deduce that $C_{\max}(\alpha\beta, t') = C_{\max}(\beta\alpha, t')$, and therefore,

$$\Delta = Z(\alpha; t') + Z(\beta; C_{\max}(\alpha; t')) - Z(\beta; t') - Z(\alpha; C_{\max}(\beta; t')).$$

For simplicity, for an arbitrary partial permutation, define

$$x_{\pi(k)} = \prod_{j=1}^{k}\left(1 + bp_{\pi(j)}\right),\ 1 \le k \le |\pi|.$$

Using this definition and (9.23), we obtain

$$\Delta = \left(\frac{1+bt'}{b}\sum_{k=1}^{u} w_{\alpha(k)}x_{\alpha(k)} - \frac{1}{b}\right) + \left(\frac{1+bC_{\max}(\alpha; t')}{b}\sum_{k=1}^{v} w_{\beta(k)}x_{\beta(k)} - \frac{1}{b}\right)$$
$$-\left(\frac{1+bt'}{b}\sum_{k=1}^{v} w_{\beta(k)}x_{\beta(k)} - \frac{1}{b}\right) - \left(\frac{1+bC_{\max}(\beta; t')}{b}\sum_{k=1}^{u} w_{\alpha(k)}x_{\alpha(k)} - \frac{1}{b}\right).$$

From (9.24), we have

$$C_{\max}(\alpha; t') = \frac{1+bt'}{b}\prod_{j=1}^{u}\left(1 + a_{\alpha(j)}\right) - \frac{1}{b} = \frac{1+bt'}{b}x_{\alpha(u)} - \frac{1}{b};$$
$$C_{\max}(\beta; t') = \frac{1+bt'}{b}x_{\beta(v)} - \frac{1}{b},$$

so that

$$\frac{1+bC_{\max}(\alpha; t')}{b} = \frac{1 + b\left(\frac{1+bt'}{b}x_{\alpha(u)} - \frac{1}{b}\right)}{b} = \frac{1+bt'}{b}x_{\alpha(u)};$$
$$\frac{1+bC_{\max}(\beta; t')}{b} = \frac{1 + b\left(\frac{1+bt'}{b}x_{\beta(v)} - \frac{1}{b}\right)}{b} = \frac{1+bt'}{b}x_{\beta(v)}.$$

Thus,

$$\Delta = \frac{1+bt'}{b}\left(\left(1 - x_{\beta(v)}\right)\sum_{k=1}^{u} w_{\alpha(k)}x_{\alpha(k)} - \left(1 - x_{\alpha(u)}\right)\sum_{k=1}^{v} w_{\beta(k)}x_{\beta(k)}\right), \qquad (9.25)$$

from which we can derive the following result.

**Theorem 9.10** *For problem* $1\left|p_j(\tau) = p_j(1+b\tau), SP - prec\right|\sum w_jC_j$, *the objective function is priority-generating for any positive start time* $t_0$ *and*

$$\omega(\pi) = \frac{1 - \prod_{j=1}^{|\pi|}\left(1 + bp_{\pi(j)}\right)}{\sum_{k=1}^{|\pi|} w_{\pi(k)}\prod_{j=1}^{k}\left(1 + bp_{\pi(j)}\right)} \qquad (9.26)$$

*is its priority function. Problem* $1\left|p_j(\tau) = p_j(1+b\tau), SP - prec\right|\sum w_jC_j$ *is solvable in* $O(n\log n)$ *time.*

*Proof* Dividing (9.25) by $\sum_{k=1}^{u} w_{\alpha(k)}x_{\alpha(k)}\sum_{k=1}^{v} w_{\beta(k)}x_{\beta(k)}$, we deduce that $\Delta \le 0$, provided that

$$\frac{1 - x_{\alpha(u)}}{\sum_{k=1}^{u} w_{\alpha(k)}x_{\alpha(k)}} \ge \frac{1 - x_{\beta(v)}}{\sum_{k=1}^{v} w_{\beta(k)}x_{\beta(k)}}.$$

For an arbitrary (partial) permutation $\pi$, define the function $\omega(\pi)$ by (9.26). It is easily verified that $\omega(\alpha) > \omega(\beta)$ implies $Z(\pi^{\alpha\beta}; t_0) \le Z(\pi^{\beta\alpha}; t_0)$, while $\omega(\alpha) = \omega(\beta)$ implies $Z(\pi^{\alpha\beta}; t_0) = Z(\pi^{\beta\alpha}; t_0)$ for any $t_0$, as required. □

Observe that if (9.26) is applied to a single job $j$, then it follows that for the corresponding problem with independent jobs, $\omega(j) = -bp_j/(w_j(1+bp_j))$ is a 1-priority function, which can also be written as $\omega(j) = w_j(1+bp_j)/bp_j$, which complies with Theorem 9.9.

By applying symmetric reasoning, it can be proved that for problem $1\left|p_j(\tau) = p_j(1-b\tau), SP - prec\right|\sum w_jC_j$, the objective function is priority-generating for any positive start time $t_0$ and its priority function is given by

$$\omega(\pi) = \frac{1 - \prod_{j=1}^{|\pi|}\left(1 - bp_{\pi(j)}\right)}{\sum_{k=1}^{|\pi|} w_{\pi(k)}\prod_{j=1}^{k}\left(1 - bp_{\pi(j)}\right)}.$$

## 9.3 Bibliographic Notes

The results presented in Sect. 9.1.1 on the general multiplicative start-time-dependent effect of the form (9.4) build up on the work by Yin et al. (2010) where this effect (in the learning form) is combined with a general positional learning effect. The assumption that function $f$ is non-increasingly used in Yin et al. (2010) is not always needed, and therefore, more general statements Theorems 9.2 and 9.4 can be proved.

Not all published results related to pure multiplicative start-time-dependent effect appear to be correct, and this section fixes some of the discovered flaws.

The polynomial effect (9.6) with $A < 0$ (learning) has been introduced by Yang and Kuo (2010) who claim that each problem $1\left|p_j(\tau) = p_j(1+b\tau)^A\right|\Phi$ with $\Phi \in \left\{\sum C_j^z, \xi C_{\max} + \eta \sum C_j^z\right\}$ is solvable by the SPT rule for all $A < 0$. Unfortunately, this is wrong; see Sect. 9.1.2 and, in particular, Theorems 9.6 and 9.7, as well as Example 9.2.

Statements similar to Theorem 9.9 are presented in Kononov (1998) and Zhao et al. (2003) (for problem $1\left|p_j(\tau) = p_j(1+b\tau)\right|\sum w_j C_j$). However, Kononov (1998) simply states without a proof that for problem $1\left|p_j(\tau) = p_j(1+b\tau)\right|\sum w_j C_j$, the 1-priority function is $\omega(j) = w_j\left(\frac{1}{p_j}+1\right)$. This expression is also quoted in Theorem 6.197 of the book Gawiejnowicz (2008), but in fact, it needs to be corrected as given in Theorem 9.9. A simple counterexample demonstrates that the job sequences obtained by sorting the jobs in non-increasing order of the 1-priorities computed as suggested in Kononov (1998) are not optimal. Consider an instance of problem $1\left|p_j(\tau) = p_j(1+b\tau)\right|\sum w_j C_j$ with $b = 0.05$ and two jobs such that

$$p_1 = 5, w_1 = 2;\;\; p_2 = 2, w_2 = 1.$$

Computing the 1-priorities as suggested in Kononov (1998), we obtain

$$\omega(1) = 2.4, \omega(2) = 1.5,$$

which implies that the sequence $(1, 2)$ should be optimal with the value of the objective function 17.5. However, computation in accordance with Theorem 9.9 yields

$$\omega(1) = 10, \omega(2) = 11,$$

so that the optimal permutation is $(2, 1)$ with the value of the objective function 17.

Statements similar to Corollary 9.1 are independently proved in Kononov (1998) and Zhao et al. (2003) (for problem $1\left|p_j(\tau) = p_j(1+b\tau)\right|C_{\max}$) and in Wang and Xia (2005) (for problem $1\left|p_j(\tau) = p_j(1-b\tau)\right|C_{\max}$).

The results presented in Kuo and Yang (2007) should be seen as a special case of Theorem 9.3 applied to a particular function $f(\tau) = \sum_{i=1}^{m} \lambda_i \tau^{r_i}$, where for some integer $m$, the values $\lambda_i$ and $r_i$, $1 \le i \le m$, are given non negative constants.

Theorem 9.10 is presented in Wang et al. (2008) with no detailed proof. The proof given in Sect. 9.2 follows the proof techniques outlined in Tanaev et al. (1984) and Gordon et al. (2008).

## References

Gawiejnowicz S (2008) Time-dependent scheduling. Springer, Berlin

Gordon VS, Potts CN, Strusevich VA, Whitehead JD (2008) Single machine scheduling models with deterioration and learning: handling precedence constraints via priority generation. J Sched 11:357–370

Kononov A (1998) Single machine scheduling problems with processing times proportional to an arbitrary function. Diskretnyj Analiz i Issledovanie Operatsij, Seria 1 5(3):17–37 (in Russian)

Kuo W-H, Yang D-L (2007) Single-machine scheduling problems with start-time dependent processing time. Comp Math Appl 53:1658–1664

Tanaev VS, Gordon VS, Shafransky YM (1984) Scheduling theory. Single-stage systems. Moscow, Nauka (in Russian); Kluwer, Dordrecht, 1994 (in English)

Wang J-B, Xia Z-Q (2005) Scheduling jobs under decreasing linear deterioration. Inf Proc Lett 94:63–69

Wang J-B, Ng CT, Cheng TCE (2008) Single-machine scheduling with deteriorating jobs under a series-parallel graph constraint. Comput Oper Res 35:2684–2693

Yang DL, Kuo WH (2007) Single-machine scheduling with an actual time-dependent learning effect. J Oper Res Soc 58:1348–1353

Yin Y, Xu D, Wang J (2010) Single-machine scheduling problems with a general sum-of-actual-processing-times-based and job-position-based learning effect. Appl Math Model 34:3623–3630

Zhao C-L, Zhang Q-L, Tang H-Y (2003) Scheduling problems under linear deterioration. Acta Autom Sinica 29:531–535

# Chapter 10
# Scheduling with Pure and Combined Cumulative Effects

In this chapter, we study single machine scheduling problems, provided that the actual processing times of the jobs are subject to a cumulative effect. We also study effects in which a cumulative effect is combined with a positional effect.

For a job $j \in N = \{1, 2, \ldots, n\}$, its normal processing time $p_j$ is given. Suppose that the jobs are processed on a single machine in accordance with a permutation $\pi = (\pi(1), \ldots, \pi(n))$. As stated in Sect. 6.3, under a general cumulative effect, the actual processing time $p_j(r)$ of a job $j = \pi(r)$ sequenced in position $r$, $1 \leq r \leq n$, depends on the normal times of the previously sequenced jobs, which is given by

$$p_j(r) = p_j f(P_r), \tag{10.1}$$

where

$$P_r = \sum_{h=1}^{r-1} p_{\pi(h)}$$

is the sum of the normal processing times of the earlier sequenced jobs. In the case of learning, $f : [0, +\infty) \to (0, 1]$ is a non-increasing function, while in the case of deterioration, $f : [0, +\infty) \to [1, +\infty)$ is a non-decreasing function.

As adopted throughout this book, if job $j$ is sequenced in position $\pi(r)$ of permutation $\pi$, its completion time is denoted either by $C_j(\pi)$ or by $C_{\pi(r)}$, whichever is more convenient.

We denote the problems of minimizing an objective function $\Phi$ on a single machine subject to an effect (10.1) by $1\left|p_j(r) = p_j f(P_r)\right|\Phi$. Typically, we will have either $\Phi = C_{\max}$ in the case of minimizing makespan or $\Phi = \sum C_j$ in the case of minimizing total completion time. Whenever possible, we also consider the problems of minimizing more general objective functions, such as $\sum C_j^z$, where $z$ is a given positive number, and the linear combination $\xi C_{\max} + \eta \sum C_j^z$ with positive coefficients.

In this book, we often turn to the most popular cumulative effect given by

$$p_j(r) = p_j\left(1 + b\sum_{h=1}^{r-1} p_{\pi(h)}\right)^A, \tag{10.2}$$

where $b$ is either a positive or a negative constant suitably assigned to ensure that the actual processing times do not become too high, or too low or negative. Provided that $b > 0$, the constant $A$ is negative in the case of learning and is positive in the case of deterioration. We denote the problems of minimizing an objective function $\Phi$ on a single machine subject to an effect (10.2) by $1\left|p_j(r) = p_j(1 + bP_r)^A\right|\Phi$.

In this chapter, we also study more general effects, e.g., an effect that combines a general cumulative effect with a general monotone positional effect, so that the actual processing time of job $j$ scheduled in the $r$th position of a permutation $\pi$ is given by

$$p_j(r) = p_j f(P_r)g(r), \tag{10.3}$$

where

- $f$ is a continuous differentiable function, common to all jobs, that depends on the sum $P_r$ of normal processing times of jobs scheduled in the previous positions and takes positive values;
- array $g(r)$, $1 \le r \le n$, is a monotone sequence and defines a positional effect.

It is assumed that $f(0) = 1$ and $g(1) = 1$, which guarantees that for the job which is the first in the processing sequence, the actual processing time is equal to its normal time.

Many problems from the considered range admit a solution by a priority rule. Recall that if the jobs are numbered in accordance with the LPT rule, then

$$p_1 \ge p_2 \ge \cdots \ge p_n, \tag{10.4}$$

while if they are numbered in accordance with the SPT rule, then

$$p_1 \le p_2 \le \cdots \le p_n. \tag{10.5}$$

This chapter is structured as follows. Section 10.1 studies single machine problems with no precedence constraints under various forms of a combined effect (10.3). In Sect. 10.2, we consider problems with pure cumulative effects, including a generalized linear effect (6.9), under which the actual processing time depends on accumulated values of a parameter different from the normal processing time. Section 10.3 considers problems with series-parallel precedence constraints, mainly under pure cumulative effects.

## 10.1 Scheduling Independent Jobs with a Combined Job-Independent Cumulative Effect

In this section, we consider single machine problems under the combined effect (10.3). We prove general statements that compare the completion times of jobs in two permutations that differ only by the positions of two consecutive jobs. Then, we show its implications for problems of minimizing the makespan, the total completion time, and their generalizations.

Similarly to Sect. 9.1 that considers a combined multiplicative start-time-dependent effect of the form (9.4), which resembles (10.3), in this section our reasoning relies on the behavior of function

$$\varphi(t) = (1-\lambda)f(P_r) + \lambda\mu f(P_r + t) - \mu f(P_r + \lambda t), \tag{10.6}$$

which is established in Lemma 9.1. To make this chapter self-consistent, below we repeat the formulation of that lemma.

**Lemma 10.1** *For function $\varphi(t)$ defined by (9.7) with $P_r \geq 0$, the inequality*

$$\varphi(t) \leq 0$$

*holds for all $t \geq 0$ if*

*(a)* $\lambda \geq 1$, $0 < \mu \leq 1$ *and $f$ is convex on* $[0, +\infty)$*, or*
*(b)* $0 < \lambda \leq 1$, $\mu \geq 1$ *and $f$ is concave on* $[0, +\infty)$.

We split our further consideration into two parts. Our analysis and the final results depend on the behavior of function $f$ that defines a cumulative effect (10.1) and whether the array of positional factors $g(r)$ represents a learning effect or a deterioration effect.

### *10.1.1 Combining General Cumulative Effects with Positional Effects*

Let us first analyze scheduling problems with a combined effect (10.3), provided that function $f$ is convex and the array $g(r)$, $1 \leq r \leq n$, is non-increasing, i.e., represents a positional learning effect. Under this assumption, it is possible to prove a rather general statement, which should be seen as an extension of Theorem 2.3.

**Theorem 10.1** *Let $\pi$ be a permutation, in which two jobs $u$ and $v$ such that*

$$p_u > p_v, \tag{10.7}$$

*occupy two consecutive positions $r$ and $r+1$, i.e., $u = \pi(r)$ and $v = \pi(r+1)$, where $1 \leq r \leq n-1$. Let permutation $\pi'$ be obtained from $\pi$ by swapping the jobs $u$ and $v$. Then for a single machine problem with a combined effect (10.3) the inequality*

$$C_{\pi(h)} \geq C_{\pi'(h)} \tag{10.8}$$

*holds for all $h$, $1 \leq h \leq n$, provided that*

$$1 = g(1) \geq g(2) \geq \cdots \geq g(n), \tag{10.9}$$

*and $f$ is convex on $[0, +\infty)$.*

*Proof* It is convenient to represent permutation $\pi$ as $\pi = (\pi_1, u, v, \pi_2)$, where $\pi_1$ and $\pi_2$ are subsequences of jobs that precede job $u$ and follow job $v$ in permutation $\pi$, respectively. Then, $\pi' = (\pi_1, v, u, \pi_2)$.

Define
$$\lambda := p_u/p_v. \tag{10.10}$$

We present the proof assuming that both sequences $\pi_1$ and $\pi_2$ are non-empty; otherwise, the corresponding part of the proof can be skipped.

The actual processing times and the completion times of all jobs in sequence $\pi_1$ are not affected by the swap of jobs $u$ and $v$, i.e., the inequality (10.7) holds as equality for each $h$, $1 \leq h \leq r-1$.

Denote $P_r = \sum_{h=1}^{r-1} p_{\pi(h)}$ and define $Y$ as the completion time of the job in the $(r-1)$th position in sequence $\pi$ (or, equivalently, in $\pi'$), i.e., $Y = C_{\pi(r-1)} = C_{\pi'(r-1)}$. For $h = r$, we derive that

$$C_{\pi(r)} = C_u(\pi) = Y + p_u f(P_r) g(r);$$
$$C_{\pi'(r)} = C_v(\pi') = Y + p_v f(P_r) g(r).$$

Due to (10.7), we see that inequality (10.8) holds for $h = r$.

For $h = r+1$, we derive that

$$C_{\pi(r+1)} = C_v(\pi) = C_u(\pi) + p_v f(P_r + p_u) g(r+1)$$
$$C_{\pi'(r+1)} = C_u(\pi') = C_v(\pi') + p_u f(P_r + p_v) g(r+1).$$

Define
$$\Delta := C_{\pi'(r+1)} - C_{\pi(r+1)}. \tag{10.11}$$

We show that $\Delta \leq 0$. Writing out the actual processing times of jobs $u$ and $v$ in permutations $\pi$ and $\pi'$, we obtain

$$\Delta = p_v f(P_r) g(r) + p_u f(P_r + p_v) g(r+1) - p_u f(P_r) g(r) \\ - p_v f(P_r + p_u) g(r+1).$$

Using (10.10), $\Delta$ can be rewritten as

$$\begin{aligned}\Delta &= p_v\Bigg(f(P_r)g(r) + \frac{p_u}{p_v} f(P_r + p_v)g(r+1) \\ &\quad -\frac{p_u}{p_v} f(P_r)g(r) - f(P_r + p_u)g(r+1)\Bigg) \\ &= p_v(f(P_r)g(r) + \lambda f(P_r + p_v)g(r+1) \\ &\quad -\lambda f(P_r)g(r) - f(P_r + \lambda p_v)g(r+1)) \\ &= p_v((1-\lambda)f(P_r)g(r) + \lambda f(P_r + p_v)g(r+1) - f(P_r + \lambda p_v)g(r+1)).\end{aligned} \tag{10.12}$$

Further, denote

$$\mu := \frac{g(r+1)}{g(r)}, \tag{10.13}$$

and rewrite (10.12) as

$$\Delta = p_v g(r)((1-\lambda)f(P_r) + \lambda\mu f(P_r + p_v) - \mu f(P_r + \lambda p_v))$$

Applying (10.6) with $t = p_v$, we deduce that

$$\Delta = p_v g(r)\varphi(p_v). \tag{10.14}$$

By assumption (10.7), we have that $\lambda > 1$, and due to (10.9), the inequality $\mu < 1$ holds. Since $f$ is convex by the theorem's condition, it follows that condition (a) of Lemma 10.1 is valid. Thus, $\varphi(p_v) \le 0$ and $\Delta \le 0$, as required.

The actual processing times of all jobs in the sequence $\pi_2$ are not affected by the swap of jobs $u$ and $v$. Besides, due to (10.8) proved for $h = r+1$, each job in position $h$, $r+2 \le h \le n$, starts in the schedule associated with permutation $\pi$ no earlier than in the schedule associated with $\pi'$. This means that (10.8) holds for each $h, r+2 \le h \le n$. The theorem is fully proved. □

Theorem 10.1 immediately implies that for the combined effect (10.3) defined by a convex function $f$ and a non-increasing array (10.9), any regular objective function $\Phi(\pi)$ that depends only on the completion times is minimized if the jobs are arranged in a sequence in which a job with a larger normal processing time is not followed by a job with a smaller normal processing time. Examples of such a function include, but not limited to $C_{\max}$, $\sum C_j^z$, where $z$ is a given positive number, and their linear combination $\xi C_{\max} + \eta \sum C_j^z$.

**Theorem 10.2** *For problem $1\left|p_j(r) = p_j f(P_r)g(r)\right|\Phi$ with $\Phi \in \left\{C_{\max}, \sum C_j^z,\right.$ $\left.\xi C_{\max} + \eta \sum C_j^z\right\}$ under an effect (10.3) that combines a cumulative effect with a positional learning effect, an optimal permutation can be found in $O(n \log n)$ time by sorting the jobs in accordance with the SPT rule, provided that $f$ is convex on $[0, +\infty)$.*

The latter theorem should be seen as an extension of Theorem 2.4.

Reformulating Theorem 10.2 in terms of 1-priorities, we conclude that problem $1\left|p_j(r) = p_j f(P_r) g(r)\right|\Phi$, where $\Phi \in \left\{C_{\max}, \sum C_j^z, \xi C_{\max} + \eta \sum C_j^z\right\}$, with an effect (10.3) admits the 1-priority $\omega(j) = 1/p_j$, provided that functions $f$ and $g$ satisfy the conditions of Theorem 10.1.

Now, we pass to analyzing scheduling problems with a combined effect (10.3), provided function $f$ is concave and the array $g(r)$, $1 \le r \le n$, is non-decreasing, i.e., represents a positional deterioration effect.

It appears that the analysis of the effect (10.3) defined by a function convex $f$ and positional factors defined by a non-decreasing array $g(r)$, $1 \le r \le n$, is not fully symmetric to that presented in Theorem 10.1. A certain symmetry is observed only for the problem of minimizing makespan: If function $f$ is concave and the array $g(r)$, $1 \le r \le n$, is non-decreasing, then an optimal permutation can be found by the LPT rule. On the other hand, for minimizing the total completion time, an LPT permutation need not be optimal for such an effect.

**Theorem 10.3** *Let $\pi$ be a permutation, in which two jobs $u$ and $v$ such that*

$$p_u < p_v,$$

*occupy two consecutive positions $r$ and $r+1$, i.e., $u = \pi(r)$ and $v = \pi(r+1)$. Let permutation $\pi'$ be obtained from $\pi$ by swapping the jobs $u$ and $v$. Let permutation $\pi'$ be obtained from $\pi$ by swapping the jobs $u$ and $v$. Then for a single machine problem with a cumulative effect (10.3) the inequality $C_{\max}(\pi') \le C_{\max}(\pi)$ holds, provided that*

$$1 = g(1) \le g(2) \le \cdots \le g(n), \tag{10.15}$$

*and $f$ is concave on* $[0, +\infty)$.

*Proof* As in the proof of Theorem 10.1, we represent permutations $\pi$ and $\pi'$ as $\pi = (\pi_1, u, v, \pi_2)$ and as $\pi' = (\pi_1, v, u, \pi_2)$, where $\pi_1$ and $\pi_2$ are subsequences of jobs that precede job $u$ and follow job $v$ in permutation $\pi$, respectively.

We present the proof assuming that both sequences $\pi_1$ and $\pi_2$ are non-empty; otherwise, the corresponding part of the proof can be skipped.

The actual processing times and the completion times of all jobs in sequence $\pi_1$ are not affected by the swap of jobs $u$ and $v$. Denote $P_r = \sum_{h=1}^{r-1} p_{\pi(h)}$ and define $Y$ as the completion time of the job in the $(r-1)$th position in sequence $\pi$ (or, equivalently, in $\pi'$), i.e., $Y = C_{\pi(r-1)} = C_{\pi'(r-1)}$. It is clear that $C_{\pi(r)} = Y + p_u f(P_r) g(r)$ and $C_{\pi'(r)} = Y + p_v f(P_r) g(r)$. Notice that $p_u < p_v$, which implies that $C_{\pi(r)} < C_{\pi'(r)}$. This explains why under the conditions of Theorem 10.3 it is not possible to prove a more general statement, similar to Theorem 10.1.

We now prove that $C_{\pi'(r+1)} \leq C_{\pi(r+1)}$. As in the proof of Theorem 10.1, define $\Delta$ by (10.11). Defining $\lambda$ by (10.10) and $\mu$ by (10.13), rewrite $\Delta$ as (10.14).

Unlike in the proof of Theorem 10.1, here $\lambda < 1$ and $\mu \geq 1$, so that condition (b) of Lemma 10.1 is valid. Thus, $\varphi(p_v) \leq 0$ and $\Delta \leq 0$, as required.

The actual processing times of all jobs in the sequence $\pi_2$ are not affected by the swap of jobs $u$ and $v$. Besides, since $C_{\pi'(r+1)} \leq C_{\pi(r+1)}$, it follows that each job in position $h$, $r+2 \leq h \leq n$, starts in the schedule associated with permutation $\pi$ no earlier than in the schedule associated with $\pi'$. This means that $C_{\max}\left(\pi'\right) \leq C_{\max}(\pi)$ holds, which proves the theorem. $\square$

Theorem 10.3 immediately leads to the following statement regarding single machine scheduling problems to minimize the makespan.

**Theorem 10.4** *For problem* $1\left|p_j(r) = p_j f(P_r) g(r)\right|C_{\max}$ *under an effect (10.3) that combines a cumulative effect with a positional deterioration effect, an optimal permutation can be found in* $O(n \log n)$ *time by sorting the jobs in accordance with the LPT rule, provided that function* $f$ *is concave on* $[0, +\infty)$.

Reformulating Theorem 10.4, we conclude that problem $1\left|p_j(r) = p_j f(P_r)\right.$ $g(r)|C_{\max}$ with an effect (10.3) admits the 1-priority $\omega(j) = p_j$, provided that functions $f$ and $g$ satisfy the conditions of Theorem 10.3.

Notice that the proof of Theorem 10.3 in fact demonstrates that the inequality (10.8) holds for each $h$ other than $r$, while $C_{\pi(r)} < C_{\pi'(r)}$. This fact does not allow us to derive any conclusions regarding the status of the problem of minimizing total completion time under an effect (10.3), provided that function $f$ is concave on $[0, +\infty)$ and (10.15) holds. The status of this problem remains open even if $g(r) = 1$, $1 \leq r \leq n$.

Below, we now present a counterexample that demonstrates that problem $1\left|p_j(r) = p_j f(P_r)\right| \sum C_j$ can be solved neither by the SPT nor by the LPT rule.

*Example 10.1* Consider an instance of a single machine problem to minimize the sum of completion times under the cumulative effect (10.1) with the function $f(P_r) = (1 + P_r)^{\frac{1}{2}}$. It is clear that $f$ is concave on $[0, +\infty)$. There are 3 jobs with the normal processing times listed below

$$p_1 = 6, \; p_2 = 7, \; p_3 = 9.$$

For this instance, the total completion time $\sum C_j$ is minimized neither by the SPT sequence, nor by the LPT sequence. The corresponding computations, accurate up to three decimal places, are shown in Table 10.1. We see that permutation $(2, 1, 3)$ delivers a smaller value of the total completion time than that produced by the SPT and the LPT sequences.

**Table 10.1** Computations for Example 10.1

| | $\pi = (1, 2, 3)$ [SPT] | $\pi = (3, 2, 1)$ [LPT] | $\pi = (2, 1, 3)$ |
|---|---|---|---|
| $C_{\pi(1)}$ | 6.000 | 9.000 | 7.000 |
| $C_{\pi(2)}$ | 24.520 | 31.136 | 23.971 |
| $C_{\pi(3)} = C_{\max}(\pi)$ | 58.195 | 55.875 | 57.645 |
| $\sum C_{\pi(j)}$ | 88.715 | 96.011 | 88.616 |

### *10.1.2 Combining Polynomial Cumulative Effects with Positional Effects*

Below, we consider the implications of the general statements proved above for effects with a specific function $f$.

Let us first consider the combined effect (10.3) in which $f(P_r) = (1 + bP_r)^A$, so that the resulting effect is given by

$$p_j(r) = p_j(1 + bP_r)^A g(r). \tag{10.16}$$

Recall that $b$ is a suitably defined constant which can be either positive or negative. For the problem of minimizing the makespan, Theorems 10.2 and 10.4 imply the following statement.

**Theorem 10.5** *For problem* $1\left|p_j(r) = p_j(1 + bP_r)^A g(r)\right|C_{\max}$ *under an effect (10.16) applied with either* $A < 0$ *or with* $A \geq 1$*, and* $g(r)$ *defined according to (10.9), an optimal permutation can be found in* $O(n \log n)$ *time by sorting the jobs in accordance with the SPT rule, while if the effect is applied with* $0 < A \leq 1$*, and* $g(r)$ *is defined according to (10.15), then an optimal permutation can be found in* $O(n \log n)$ *time by sorting the jobs in accordance with the LPT rule.*

*Proof* To see that the theorem holds, apply Theorems 10.2 and 10.4 to the function $f(P_r) = (1 + bP_r)^A$. We see that the second-order derivative

$$\frac{d^2 f}{dP_r^2} = A(A-1)b^2(1 + bP_r)^{A-2}$$

is non-negative, i.e., $f$ is convex, provided that either $A < 0$ or $A \geq 1$. On the other hand, $\frac{d^2 f}{dP_r^2}$ is non-positive (and $f$ is concave) if $0 < A \leq 1$. $\square$

Assuming that $b > 0$, Theorem 10.5 implies that problem $1\left|p_j(r) = p_j(1 + bP_r)^A g(r)\right|C_{\max}$ for $A < 0$ (a learning cumulative effect) or $A \geq 1$ (a fast deterioration cumulative effect) along with a positional learning effect is solvable in $O(n \log n)$ time, and an optimal permutation is found by sequencing jobs in the SPT order, i.e., $\omega(j) = 1/p_j$ is the 1-priority. On the other hand, for problem

**Table 10.2** Results for scheduling on a single machine with a combined effect (10.3)

| Condition on $f$ | Condition on $g$ | Objective | Rule | Statement |
|---|---|---|---|---|
| $f$ convex | $g \searrow$ | $C_{\max}$ | SPT | Theorem 10.2 |
| $f$ convex | $g \searrow$ | $\sum C_j^z$ | SPT | Theorem 10.2 |
| $f$ convex | $g \searrow$ | $\xi C_{\max} + \eta \sum C_j^z$ | SPT | Theorem 10.2 |
| $f$ concave | $g \nearrow$ | $C_{\max}$ | LPT | Theorem 10.4 |
| $f$ concave | $g = 1$ | $\sum C_j$ | open | Example 10.1 |
| $f = (1 + bP_r)^A$, $A < 0$ or $A \geq 1$ | $g \searrow$ | $C_{\max}$ | SPT | Theorem 10.5 |
| $f = (1 + bP_r)^A$, $A < 0$ or $A \geq 1$ | $g \searrow$ | $\sum C_j^z$ | SPT | Theorem 10.5 |
| $f = (1 + bP_r)^A$, $A < 0$ or $A \geq 1$ | $g \searrow$ | $\xi C_{\max} + \eta \sum C_j^z$ | SPT | Theorem 10.5 |
| $f = (1 + bP_r)^A$, $0 < A \leq 1$ | $g \nearrow$ | $C_{\max}$ | LPT | Theorem 10.6 |
| $f = (1 + bP_r)^A$, $0 < A < 1$ | $g = 1$ | $\sum C_j$ | open | Example 10.1 |

$1\left|p_j(r) = p_j(1 + bP_r)^A g(r)\right|C_{\max}$ with $0 < A \leq 1$ (a slow deterioration cumulative effect) and a positional deterioration effect, an optimal solution can be found in $O(n \log n)$ time by the LPT rule, i.e., $\omega(j) = p_j$ is the 1-priority.

Reformulating Theorem 10.5, we conclude that for problem $1\left|p_j(r) = p_j(1 + bP_r)^A g(r)\right|C_{\max}$ under an effect (10.16), the 1-priority is either $\omega(j) = 1/p_j$ (if $A < 0$ or $A \geq 1$, and $g(r)$ is defined by (10.9)) or $\omega(j) = p_j$ (if $0 < A \leq 1$, and $g(r)$ is defined by (10.15)).

For other single machine problems, Theorem 10.2 leads to the following statement.

**Theorem 10.6** *Let $z$ be a positive number. For problem $1\left|p_j(r) = p_j(1 + bP_r)^A \ g(r)\right|\Phi$ under an effect (10.16) applied with either $A < 0$ or with $A \geq 1$, and $g(r)$ defined according to (10.9), an optimal permutation can be found for all $\Phi \in \left\{\sum C_j^z, \xi C_{\max} + \eta \sum C_j^z\right\}$ in $O(n \log n)$ time by sorting the jobs in accordance with the SPT rule.*

The proof of this theorem is the same as that of Theorem 10.5.

Thus, problem $1\left|p_j(r) = p_j(1 + bP_r)^A g(r)\right|\sum C_j^z$ for $A < 0$ (a learning effect) or $A \geq 1$ (a fast deterioration effect) along with a positional learning effect is solvable in $O(n \log n)$ time, and an optimal permutation is found by sequencing jobs in the SPT order. Reformulating Theorem 10.6, we conclude that for problem $1\left|p_j(r) = p_j(1 + bP_r)^A g(r)\right|\sum C_j^z$ under an effect (10.16), the 1-priority is either $\omega(j) = 1/p_j$ (if $A < 0$ or $A \geq 1$, and $g(r)$ is defined by (10.9)) or $\omega(j) = p_j$ (if $0 < A \leq 1$, and $g(r)$ is defined by (10.15)).

The status of problem $1\left|p_j(r) = p_j(1 + bP_r)^A g(r)\right|\sum C_j$ for $0 < A < 1$ (a slow deterioration effect in the case of $b > 0$) is left undecided. As demonstrated in Example 10.1, for this problem even if $g(r) = 1$, $1 \le r \le n$, there exists an instance for which neither the SPT sequence nor the LPT sequence is optimal.

The results presented in Sect. 10.1 for combined effects are summarized in Table 10.2. Here, in the second column, we use symbols $\nearrow$ and $\searrow$ to indicate whether the sequence $g(r)$, $1 \le r \le n$, is non-decreasing or non-increasing, respectively. Additionally, we write $g = 1$ if $g(r) = 1$, $1 \le r \le n$.

## 10.2 Pure Cumulative Effects

Many results for a pure cumulative effect can be easily obtained from the statements in the previous section by setting $g(r) = 1$, $1 \le r \le n$; see, e.g., Table 10.3. This is why in this section we focus of the results which cannot be derived from considering an effect of the form (10.3).

### *10.2.1 Job-Dependent Linear Generalized Cumulative Effect*

Consider an effect that arises when a job $j \in N$ is associated not only with normal processing time $p_j$ but also with two additional parameters, $b_j$ and $q_j$. The actual processing time of job $j$ scheduled in the $r$th position of permutation $\pi$ is defined by

$$p_j(r) = p_j\left(1 + b_j\sum_{h=1}^{r-1} q_{\pi(h)}\right), \tag{10.17}$$

where $b_j > 0$ under a deterioration effect and $b_j < 0$ under a learning effect. No explicit dependence on the normal time of previously scheduled jobs is assumed, and the values of $b_j$ can be understood as job-dependent rates that reflect how sensitive a particular job is to the previously scheduled jobs.

As pointed out in Sect. 6.3, this effect more realistically reflects situations of practical interest.

In this section, we demonstrate that under effect (10.17), the problem of minimizing the makespan can be solved by a variant of the WSPT rule, which delivers an optimal solution to a single machine problem $1||\sum w_j C_j$ of minimizing the weighted sum of the completion times. Assume that in problem $1||\sum w_j C_j$, the processing time of job $j \in N$ is denoted by $q_j$. Then, the value of the objective function for a schedule associated with a permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ is given by

$$\sum_{h=1}^{n} w_{\pi(h)} C_{\pi(h)} = \sum_{k=1}^{n} w_{\pi(h)} \sum_{r=1}^{h} q_{\pi(r)},$$

and an optimal permutation can be found in $O(n \log n)$ time by sorting the jobs in accordance with the WSPT rule (also known as Smith's rule), i.e., in non-decreasing order of the ratios $q_j/w_j$; see Theorem 2.6.

**Theorem 10.7** *For a single machine problem of minimizing the makespan under the effect (10.17), an optimal permutation can be found in $O(n \log n)$ time by sorting the jobs in non-decreasing order of $q_j/(p_j b_j)$.*

*Proof* To prove the theorem, we reduce the problem under consideration to problem $1||\sum w_j C_j$, with the processing times equal to $q_j$ and the weights defined by

$$w_j := b_j p_j, \ \ j \in N. \tag{10.18}$$

For the original problem, we have

$$\begin{aligned} C_{\max}(\pi) &= p_{\pi(1)} + \sum_{r=2}^{n} p_{\pi(r)} \left( 1 + b_{\pi(r)} \sum_{h=1}^{r-1} q_{\pi(h)} \right) \\ &= \sum_{r=1}^{n} p_{\pi(r)} + \sum_{r=2}^{n} b_{\pi(r)} p_{\pi(r)} \sum_{h=1}^{r-1} q_{\pi(h)} \\ &= \sum_{r=1}^{n} p_{\pi(r)} + \sum_{r=1}^{n} b_{\pi(r)} p_{\pi(r)} \sum_{h=1}^{r-1} q_{\pi(h)}, \end{aligned}$$

where the last equality is due to $\sum_{h=1}^{0} q_{\pi(h)} = 0$.

Using (10.18), we further rewrite

$$\begin{aligned} C_{\max}(\pi) &= \sum_{r=1}^{n} p_{\pi(r)} + \sum_{r=1}^{n} w_{\pi(r)} \sum_{h=1}^{r-1} q_{\pi(h)} \\ &= \sum_{r=1}^{n} p_{\pi(r)} + \sum_{r=1}^{n} w_{\pi(r)} \sum_{h=1}^{r} q_{\pi(h)} - \sum_{r=1}^{n} w_{\pi(r)} q_{\pi(r)} \\ &= \sum_{r=1}^{n} w_{\pi(r)} \sum_{h=1}^{r} q_{\pi(h)} + \sum_{j=1}^{n} \left( p_j - w_j q_j \right). \end{aligned}$$

Thus, $C_{\max}(\pi)$ is minimized if the minimum of $\sum_{r=1}^{n} w_{\pi(r)} \sum_{h=1}^{r} q_{\pi(h)}$ is attained. The latter expression is the objective function in problem $1||\sum w_j C_j$, so that the optimal permutation can be found by the WSPT rule. In terms of the original problem, an optimal permutation is obtained by sorting the jobs in non-decreasing order of the ratios $q_j/(b_j p_j)$. $\square$

Reformulating Theorem 10.7, we conclude that for the problem of minimizing the makespan under an effect (10.17), the 1-priority is $\omega(j) = b_j p_j / q_j$.

**Corollary 10.1** *If effect (10.17) is applied with $q_j = p_j$, then the resulting problem $1\left|p_j(\pi; r) = p_j\left(1 + b_j P_r\right)\right|C_{\max}$ is solvable in $O(n \log n)$ time by sequencing jobs in non-increasing order of $b_j$.*

## 10.2.2 Job-Independent Linear Cumulative Effect

In the remainder of this section, we consider a pure cumulative effect of the form (10.2) applied with $A = 1$, i.e., a linear effect given by

$$p_j(r) = p_j\left(1 + b\sum_{h=1}^{r-1} p_{\pi(h)}\right). \tag{10.19}$$

According to Theorem 10.5, for an effect (10.16) if $A = 1$ and $g(r)$ is defined according to (10.9), an optimal permutation for minimizing the makespan can be found by sorting the jobs in accordance with the SPT rule, and if $A = 1$ and $g(r)$ is defined according to (10.15), an optimal permutation for minimizing the makespan can be found by sorting the jobs in accordance with the LPT rule. Notice that effect (10.19) can be seen as a special case of effect (10.16) with $A = 1$ and $g(r) = 1, 1 \le r \le n$, so that both (10.9) and (10.15) are satisfied. Thus, Theorem 10.5 implies that both SPT and LPT rules will result in an optimal solution for problem $1\left|p_j(r) = p_j(1 + bP_r)\right|C_{\max}$. In fact, we can prove an even more general statement.

**Lemma 10.2** *For problem $1\left|p_j(r) = p_j(1 + bP_r)\right|C_{\max}$ with a cumulative effect of the form (10.19), the value of the objective function does not depend on the sequencing of jobs.*

*Proof* A possible proof of the lemma demonstrates that for an arbitrary permutation $\pi = (\pi(1), \dots, \pi(n))$, the expression for $C_{\max}(\pi)$ can be written in a sequence-independent way. Due to (10.19), we have that

$$\begin{aligned} C_{\max}(\pi) &= p_{\pi(1)} + \sum_{r=2}^{n} p_{\pi(r)}\left(1 + b\sum_{h=1}^{r-1} p_{\pi(h)}\right) \\ &= \sum_{r=1}^{n} p_{\pi(r)} + b\sum_{r=2}^{n} p_{\pi(r)} \sum_{h=1}^{r-1} p_{\pi(h)}. \end{aligned}$$

**Table 10.3** Results for scheduling independent jobs on a single machine with a pure cumulative effect

| $p_j(r)$ | Condition | Objective | 1-Priority | Statement |
|---|---|---|---|---|
| $p_j f(P_r)$ | $f$ convex | $C_{\max}$ | $1/p_j$ | Theorem 10.2 |
| $p_j f(P_r)$ | $f$ convex | $\sum C_j^z$ | $1/p_j$ | Theorem 10.2 |
| $p_j f(P_r)$ | $f$ convex | $\xi C_{\max} + \eta \sum C_j^z$ | $1/p_j$ | Theorem 10.2 |
| $p_j f(P_r)$ | $f$ concave | $C_{\max}$ | $p_j$ | Theorem 10.4 |
| $p_j(1+bP_r)^A$ | $A=1$ | $C_{\max}$ | $\forall$ | Lemma 10.2 |
| $p_j(1+bP_r)^A$ | $A<0$ or $A\geq 1$ | $C_{\max}$ | $1/p_j$ | Theorem 10.5 |
| $p_j(1+bP_r)^A$ | $0<A\leq 1$ | $C_{\max}$ | $p_j$ | Theorem 10.5 |
| $p_j(1+bP_r)^A$ | $A<0$ or $A\geq 1$ | $\sum C_j^z$ | $1/p_j$ | Theorem 10.6 |
| $p_j(1+bP_r)^A$ | $A<0$ or $A\geq 1$ | $\xi C_{\max} + \eta \sum C_j^z$ | $1/p_j$ | Theorem 10.6 |
| $p_j(1+bP_r)^A$ | $0<A<1$ | $\sum C_j$ | open | Example 10.1 |
| (10.17) | | $C_{\max}$ | $q_j/(b_j p_j)$ | Theorem 10.7 |
| $p_j\left(1+b_j P_r\right)$ | | $C_{\max}$ | $b_j$ | Corollary 10.1 |

Changing the order of summation, we obtain

$$C_{\max}(\pi) = \sum_{r=1}^{n} p_{\pi(r)} + b\sum_{r=1}^{n-1} p_{\pi(r)} \sum_{h=r+1}^{n} p_{\pi(h)} = \sum_{r=1}^{n} p_{\pi(r)} + b \sum_{1\leq r<h\leq n} p_{\pi(r)} p_{\pi(h)}.$$

Since

$$\left(\sum_{r=1}^{n} p_{\pi(r)}\right)^2 = \sum_{r=1}^{n} p_{\pi(r)}^2 + 2\sum_{1\leq r<h\leq n} p_{\pi(r)} p_{\pi(h)},$$

we deduce

$$\sum_{1\leq r<h\leq n} p_{\pi(r)} p_{\pi(h)} = \frac{1}{2}\left(\sum_{r=1}^{n} p_{\pi(r)}\right)^2 - \frac{1}{2}\sum_{r=1}^{n} p_{\pi(r)}^2,$$

so that

$$C_{\max}(\pi) = \sum_{r=1}^{n} p_{\pi(r)} + \frac{b}{2}\left(\sum_{r=1}^{n} p_{\pi(r)}\right)^2 - \frac{b}{2}\sum_{r=1}^{n} p_{\pi(r)}^2.$$

The right-hand side of the above expression is in fact sequence-independent, so that it can be written as

$$C_{\max}(\pi) = \sum_{j=1}^{n} p_j + \frac{b}{2}\left(\sum_{j=1}^{n} p_j\right)^2 - \frac{b}{2}\sum_{j=1}^{n} p_j^2,$$

which implies that for problem $1\left|p_j(r) = p_j(1 + bP_r)\right|C_{\max}$ with a cumulative effect (10.19), the makespan is a constant that does not depend on the order of jobs.

Another evidence that Lemma 10.2 holds comes from Theorem 10.7. Indeed, since for problem $1\left|p_j(r) = p_j(1 + bP_r)\right|C_{\max}$, Theorem 10.7 applies with $q_j = p_j, b_j = b, j \in N$, and we see that any permutation is optimal. $\square$

The results for pure cumulative effects that follow from Sects. 10.1 and 10.2 are summarized in Table 10.3. Here, we use symbol $\forall$ to indicate that an arbitrary permutation is optimal.

## 10.3 Scheduling Under Precedence Constraints

In this section, we consider single machine problems to minimize an objective function $\Phi \in \left\{C_{\max}, \sum C_j\right\}$ with a deterioration cumulative effect (10.2) applied with $A \geq 1$. Unlike in Sects. 10.1 and 10.2, here, we assume that the jobs of set $N$ are not independent and a precedence relation given by a reduction graph $G = (N, U)$ is imposed over the set $N$ of jobs. We denote the corresponding problems by $1\left|p_j(r) = p_j(1 + bP_r)^A, prec\right|\Phi$ (for arbitrary precedence constraints) and by $1\left|p_j(r) = p_j(1 + bP_r)^A, SP - prec\right|\Phi$ (for series-parallel precedence constraints). See Chap. 3 for definitions and main results on scheduling under precedence constraints.

Let $\pi$ be a (partial) permutation of jobs contained as a subsequence in some schedule. The length of a permutation $\pi$, i.e., the number of elements in $\pi$, is denoted by $|\pi|$.

### *10.3.1 Minimizing Makespan*

We start with considering the problem with the makespan objective function.

First, observe that Lemma 10.2 immediately implies that for problem $1\left|p_j(r) = p_j(1 + bP_r), prec\right|C_{\max}$ with a cumulative effect (10.2) applied with $A = 1$, any permutation that respects the given precedence constraints is optimal. Such a permutation can be found in $O(n)$ time.

Now, we pass to considering the problem of minimizing the makespan with a cumulative effect (10.2) applied with $A = 2$ and show that the objective function is priority-generating. Recall that $C_{\max}$ admits a 1-priority for $A > 1$, which is a necessary requirement for the objective function to be priority-generating. In accordance with Sect. 3.2, we apply Recipe 3.1.

In the proofs below, the following notation is used. For a partial permutation $\pi$ that is contained as a subsequence in some schedule, assume that (i) the first job in $\pi$ starts at time $\tau$ and (ii) the sum of the normal processing times $p_j$ of the jobs that precede the first job in $\pi$, i.e., those completed by time $\tau$, is equal to $\zeta$. Under these assumptions, let $C_{\max}(\pi; \tau; \zeta)$ denote the maximum completion time of the jobs in $\pi$.

**Theorem 10.8** *For the single machine problem to minimize the makespan under the deterioration effect (10.2) applied with $A = 2$ and $b > 0$, the objective function is priority-generating and*

$$\omega(\pi) = \frac{\sum_{j=1}^{|\pi|} p_{\pi(j)}}{\sum_{j=1}^{|\pi|} p_{\pi(j)}^2} \tag{10.20}$$

*is its priority function. In the case $A = 2$ and $b < 0$, the priority function is $-\omega(\pi)$. Problem $1\left|p_j(r) = p_j(1 + bP_r)^A, SP - prec\right|C_{\max}$ with $A = 2$ is solvable in $O(n \log n)$ time.*

*Proof* We present the proof for the case $b > 0$. For problem $1\left|p_j(r) = p_j (1 + bP_r)^A\right|C_{\max}$, by definition, we have that

$$C_{\max}(\pi; \tau; \zeta) = \tau + C_{\max}(\pi; 0; \zeta) = \tau + \sum_{j=1}^{|\pi|} p_{\pi(j)}\left(1 + b\left(\zeta + \sum_{i=1}^{j-1} p_{\pi(i)}\right)\right)^2. \tag{10.21}$$

Let $\pi^{\alpha\beta} = (\pi_1\alpha\beta\pi_2)$ and $\pi^{\beta\alpha} = (\pi_1\beta\alpha\pi_2)$ be two permutations of all jobs that only differ in the order of the subsequences $\alpha$ (containing $u$ jobs) and $\beta$ (containing $v$ jobs).
Define

$$\Delta := C_{\max}(\pi^{\alpha\beta}) - C_{\max}(\pi^{\beta\alpha}) = C_{\max}(\pi^{\alpha\beta}; 0; 0) - C_{\max}(\pi^{\beta\alpha}; 0; 0).$$

Let $\zeta'$ denote the total sum of the normal processing times of the jobs in $\pi_1$. It follows from (10.21) that

$$\begin{aligned} C_{\max}(\pi_1\alpha\beta\pi_2; 0; 0) &= C_{\max}(\pi_1; 0; 0) + C_{\max}(\alpha\beta\pi_2; 0; \zeta'),\\ C_{\max}(\pi_1\alpha\beta\pi_2, 0, 0) &= C_{\max}(\pi_1; 0; 0) + C_{\max}(\beta\alpha\pi_2; 0; \zeta'); \end{aligned}$$

so that $\Delta C = C_{\max}(\alpha\beta\pi_2; 0; \zeta') - C_{\max}(\beta\alpha\pi_2; 0; \zeta')$. Furthermore,

$$\begin{aligned} C_{\max}(\alpha\beta\pi_2; 0; \zeta') &= C_{\max}(\alpha\beta; 0; \zeta')\\ &\quad + \sum_{k=1}^{|\pi_2|} p_{\pi_2(k)}\left(1 + b\left(\zeta' + \sum_{i=1}^{u} p_{\alpha(i)} + \sum_{j=1}^{v} p_{\beta(j)} + \sum_{i=1}^{k-1} p_{\pi_2(i)}\right)\right)^2,\\ C_{\max}(\beta\alpha\pi_2; 0; \zeta') &= C_{\max}(\beta\alpha; 0; \zeta')\\ &\quad + \sum_{k=1}^{|\pi_2|} p_{\pi_2(k)}\left(1 + b\left(\zeta' + \sum_{j=1}^{v} p_{\beta(j)} + \sum_{i=1}^{u} p_{\alpha(i)} + \sum_{i=1}^{k-1} p_{\pi_2(i)}\right)\right)^2, \end{aligned}$$

so that $\Delta C = C_{\max}(\alpha\beta; 0; \zeta') - C_{\max}(\beta\alpha; 0; \zeta')$.

Next, we deduce

$$\begin{aligned}
C_{\max}(\alpha\beta; 0; \zeta') &= C_{\max}(\alpha; 0; \zeta') + \sum_{k=1}^{v} p_{\beta(k)} \left(1 + b\left(\zeta' + \sum_{i=1}^{u} p_{\alpha(i)} + \sum_{j=1}^{k-1} p_{\beta(j)}\right)\right)^2 \\
&= \sum_{k=1}^{u} p_{\alpha(k)} \left(1 + b\left(\zeta' + \sum_{i=1}^{k-1} p_{\alpha(i)}\right)\right)^2 \\
&\quad + \sum_{k=1}^{v} p_{\beta(k)} \left(1 + b\left(\zeta' + \sum_{i=1}^{u} p_{\alpha(i)} + \sum_{j=1}^{k-1} p_{\beta(j)}\right)\right)^2, \\
C_{\max}(\beta\alpha; 0; \zeta') &= C_{\max}(\beta; 0; \zeta') + \sum_{k=1}^{u} p_{\alpha(k)} \left(1 + b\left(\zeta' + \sum_{j=1}^{v} p_{\beta(j)} + \sum_{i=1}^{k-1} p_{\alpha(i)}\right)\right)^2 \\
&= \sum_{k=1}^{v} p_{\beta(k)} \left(1 + b\left(\zeta' + \sum_{j=1}^{k-1} p_{\beta(j)}\right)\right)^2 \\
&\quad + \sum_{k=1}^{u} p_{\alpha(k)} \left(1 + b\left(\zeta' + \sum_{j=1}^{v} p_{\beta(j)} + \sum_{i=1}^{k-1} p_{\alpha(i)}\right)\right)^2,
\end{aligned}$$

so that for $\Delta$, we derive

$$\begin{aligned}
\Delta = \sum_{k=1}^{u} p_{\alpha(k)} &\left(\left(1 + b\left(\zeta' + \sum_{i=1}^{k-1} p_{\alpha(i)}\right)\right)^2\right. \\
&\left. - \left(1 + b\left(\zeta' + \sum_{j=1}^{v} p_{\beta(j)} + \sum_{i=1}^{k-1} p_{\alpha(i)}\right)\right)^2\right) \\
+ \sum_{k=1}^{v} p_{\beta(k)} &\left(\left(1 + b\left(\zeta' + \sum_{i=1}^{u} p_{\alpha(i)} + \sum_{j=1}^{k-1} p_{\beta(j)}\right)\right)^2\right. \\
&\left. - \left(1 + b\left(\zeta' + \sum_{j=1}^{k-1} p_{\beta(j)}\right)\right)^2\right).
\end{aligned}$$

Proceeding further, we obtain

$$\begin{aligned}
\Delta = &- \sum_{k=1}^{u} p_{\alpha(k)} \left(2 \sum_{j=1}^{v} p_{\beta(j)} \left(1 + b\left(\zeta' + \sum_{i=1}^{k-1} p_{\alpha(i)}\right)\right) + b\left(\sum_{i=1}^{v} p_{\beta(i)}\right)^2\right) \\
&+ \sum_{k=1}^{v} p_{\beta(k)} \left(2 \sum_{i=1}^{u} p_{\alpha(i)} \left(1 + b\left(\zeta' + \sum_{j=1}^{k-1} p_{\beta(j)}\right)\right) + b\left(\sum_{i=1}^{u} p_{\alpha(i)}\right)^2\right)
\end{aligned}$$

$$= 2b \sum_{i=1}^{u} p_{\alpha(i)} \left( \sum_{1 \le j < k \le v} p_{\beta(j)} p_{\beta(k)} \right) + b \sum_{k=1}^{v} p_{\beta(k)} \left( \sum_{i=1}^{u} p_{\alpha(i)} \right)^2$$
$$- 2b \sum_{i=1}^{u} p_{\beta(i)} \left( \sum_{1 \le j < k \le u} p_{\alpha(j)} p_{\alpha(k)} \right) - b \sum_{k=1}^{u} p_{\alpha(k)} \left( \sum_{i=1}^{v} p_{\beta(i)} \right)^2.$$

Adding and subtracting $b \sum_{k=1}^{u} p_{\alpha(k)} \sum_{j=1}^{v} p_{\beta(j)}^2$ and $b \sum_{k=1}^{v} p_{\beta(k)} \sum_{i=1}^{u} p_{\alpha(i)}^2$, we obtain

$$\Delta = b \sum_{k=1}^{u} p_{\alpha(k)} \left( \sum_{i=1}^{v} p_{\beta(i)} \right)^2 + b \sum_{k=1}^{v} p_{\beta(k)} \left( \sum_{i=1}^{u} p_{\alpha(i)} \right)^2 - b \sum_{k=1}^{u} p_{\alpha(k)} \sum_{i=1}^{v} p_{\beta(i)}^2$$
$$- b \sum_{k=1}^{v} p_{\beta(k)} \left( \sum_{i=1}^{u} p_{\alpha(i)} \right)^2 - b \sum_{k=1}^{u} p_{\alpha(k)} \left( \sum_{i=1}^{v} p_{\beta(i)} \right)^2 + b \sum_{k=1}^{v} p_{\beta(k)} \sum_{i=1}^{u} p_{\alpha(i)}^2,$$

which reduces to

$$\Delta = b \sum_{k=1}^{v} p_{\beta(k)} \sum_{i=1}^{u} p_{\alpha(i)}^2 - b \sum_{k=1}^{u} p_{\alpha(k)} \sum_{i=1}^{v} p_{\beta(i)}^2. \tag{10.22}$$

Dividing both sides of (10.22) by $b \sum_{k=1}^{u} p_{\alpha(k)}^2 \sum_{i=1}^{v} p_{\beta(i)}^2$, we deduce from $b > 0$ that $\Delta \le 0$, provided that

$$\frac{\sum_{i=1}^{u} p_{\alpha(i)}}{\sum_{i=1}^{u} p_{\alpha(i)}^2} \ge \frac{\sum_{i=1}^{v} p_{\beta(i)}}{\sum_{i=1}^{v} p_{\beta(i)}^2}.$$

For an arbitrary (partial) permutation $\pi$, define the function $\omega(\pi)$ by (10.20). It is easily verified that for $b > 0$, the inequality $\omega(\alpha) > \omega(\beta)$ implies $C_{\max}(\pi^{\alpha\beta}) \le C_{\max}(\pi^{\beta\alpha})$, while the equality $\omega(\alpha) = \omega(\beta)$ implies $C_{\max}(\pi^{\alpha\beta}) = C_{\max}(\pi^{\beta\alpha})$, as required by the definition of the priority function.

For $b < 0$, the inequality $\Delta \le 0$ holds, provided that

$$\frac{\sum_{i=1}^{u} p_{\alpha(i)}}{\sum_{i=1}^{u} p_{\alpha(i)}^2} \le \frac{\sum_{i=1}^{v} p_{\beta(i)}}{\sum_{i=1}^{v} p_{\beta(i)}^2},$$

which leads to a priority function that is the negation of $\omega(\pi)$ defined by (10.20). $\square$

Observe that if (10.20) is applied to a single job $j$, i.e., to a permutation of length one, then the priority function becomes a 1-priority function $\omega(j) = 1/p_j$, which is consistent with the SPT being an optimal priority rule for the corresponding problem with independent jobs; see Theorem 10.5.

It is unlikely that a priority function exists for $A$ greater than 2, as the lemma below demonstrates for $A = 3$.

**Lemma 10.3** *For the single machine problem under the deterioration model (10.2) with $A = 3$ the makespan is not a priority-generating objective function.*

*Proof* In order to disprove that an objective function $\Phi$ is priority-generating, we rely on Recipe 3.2. outlined in Sect. 3.2. An instance of the problem should be exhibited such that $\Phi(\pi^{\alpha\beta}) < \Phi(\pi^{\beta\alpha})$ for some permutations $\pi^{\alpha\beta} = (\pi_1\alpha\beta\pi_2)$ and $\pi^{\beta\alpha} = (\pi_1\beta\alpha\pi_2)$, while $\Phi(\varphi^{\alpha\beta}) > \Phi(\varphi^{\beta\alpha})$ for some other permutations $\varphi^{\alpha\beta} = (\varphi'\alpha\beta\varphi'')$ and $\varphi^{\beta\alpha} = (\varphi'\beta\alpha\varphi'')$.

Consider the following instance of the problem in question. There are four jobs with

$$p_1 = 3,\ \ p_2 = 8,\ \ p_3 = 7,\ \ p_4 = 6.$$

Let $\alpha = (1, 2)$, $\beta = (3)$, and $\pi_2 = (4)$ with permutation $\pi_1$ being empty, so that $\pi^{\alpha\beta} = (\alpha\beta, 4)$ and $\pi^{\beta\alpha} = (\beta\alpha, 4)$. We have that

$$\begin{aligned}
C_{\max}(\alpha\beta, 4) &= 3 \times 1^3 + 8 \times (1+3)^3 + 7 \times (1+3+8)^3 + 6 \times (1+3+8+7)^3 \\
&= 53765; \\
C_{\max}(\beta\alpha, 4) &= 7 \times 1^3 + 3 \times (1+7)^3 + 8 \times (1+7+3)^3 + 6 \times (1+7+3+8)^3 \\
&= 53345,
\end{aligned}$$

so that $C_{\max}(\alpha\beta, 4) > C_{\max}(\beta\alpha, 4)$.

On the other hand, taking $\varphi' = (4)$ and empty permutation $\varphi'''$ we obtain

$$C_{\max}(4, \alpha\beta) = 49859 < C_{\max}(4, \beta\alpha) = 49943,$$

which proves the lemma. □

We now pass to consider a pure job-dependent generalized linear cumulative effect (10.17).

**Theorem 10.9** *For the single machine problem to minimize the makespan under the cumulative effect (10.17), the objective function is priority-generating and*

$$\omega(\pi) = \frac{\sum_{j=1}^{|\pi|} p_{\pi(j)} b_{\pi(j)}}{\sum_{j=1}^{|\pi|} q_{\pi(j)}} \tag{10.23}$$

*is its priority function. The problem is solvable in $O(n \log n)$ time.*

*Proof* We present the proof for the case $b > 0$. For the problem under consideration,

$$C_{\max}(\pi; \tau; \zeta) = \tau + C_{\max}(\pi; 0; \zeta) = \tau + \sum_{j=1}^{|\pi|} p_{\pi(j)}\left(1 + b_{\pi(j)}\left(\zeta + \sum_{i=1}^{j-1} q_{\pi(i)}\right)\right).$$

As in the proof of Theorem 10.8, let $\pi^{\alpha\beta} = (\pi_1\alpha\beta\pi_2)$ and $\pi^{\beta\alpha} = (\pi_1\beta\alpha\pi_2)$ be two permutations of all jobs that only differ in the order of the subsequences $\alpha$ (containing $u$ jobs) and $\beta$ (containing $v$ jobs). Define $\Delta := C_{\max}(\pi^{\alpha\beta}) - C_{\max}(\pi^{\beta\alpha})$, and let $\zeta'$ denote the total sum of the normal processing times of the jobs in $\pi_1$. Then, $\Delta = C_{\max}(\alpha\beta\pi_2; 0; \zeta') - C_{\max}(\beta\alpha\pi_2; 0; \zeta')$. Furthermore,

$$\begin{aligned}C_{\max}(\alpha\beta\pi_2; 0; \zeta') &= C_{\max}(\alpha\beta; 0; \zeta')\\ &\quad + \sum_{k=1}^{|\pi_2|} p_{\pi_2(k)}\left(1 + b_{\pi_2(k)}\left(\zeta' + \sum_{i=1}^{u} q_{\alpha(i)} + \sum_{j=1}^{v} q_{\beta(j)} + \sum_{i=1}^{k-1} q_{\pi_2(i)}\right)\right),\\ C_{\max}(\beta\alpha\pi_2; 0; \zeta') &= C_{\max}(\beta\alpha; 0;\\ &\quad \zeta') + \sum_{k=1}^{|\pi_2|} p_{\pi_2(k)}\left(1 + b_{\pi_2(k)}\left(\zeta' + \sum_{j=1}^{v} q_{\beta(j)} + \sum_{i=1}^{u} q_{\alpha(i)} + \sum_{i=1}^{k-1} q_{\pi_2(i)}\right)\right),\end{aligned}$$

so that $\Delta C = C_{\max}(\alpha\beta; 0; \zeta') - C_{\max}(\beta\alpha; 0; \zeta')$.

Next, we deduce

$$\begin{aligned}C_{\max}(\alpha\beta; 0; \zeta') &= C_{\max}(\alpha; 0; \zeta') + \sum_{k=1}^{v} p_{\beta(k)}\left(1 + b_{\beta(k)}\left(\zeta' + \sum_{i=1}^{u} q_{\alpha(i)} + \sum_{j=1}^{k-1} q_{\beta(j)}\right)\right)\\ &= \sum_{k=1}^{u} p_{\alpha(k)}\left(1 + b_{\alpha(k)}\left(\zeta' + \sum_{i=1}^{k-1} q_{\alpha(i)}\right)\right)\\ &\quad + \sum_{k=1}^{v} p_{\beta(k)}\left(1 + b_{\beta(k)}\left(\zeta' + \sum_{i=1}^{u} q_{\alpha(i)} + \sum_{j=1}^{k-1} q_{\beta(j)}\right)\right),\\ C_{\max}(\beta\alpha; 0; \zeta') &= C_{\max}(\beta; 0; \zeta') + \sum_{k=1}^{u} p_{\alpha(k)}\left(1 + b_{\alpha(k)}\left(\zeta' + \sum_{j=1}^{v} q_{\beta(j)} + \sum_{i=1}^{k-1} q_{\alpha(i)}\right)\right)\\ &= \sum_{k=1}^{v} p_{\beta(k)}\left(1 + b_{\beta(k)}\left(\zeta' + \sum_{j=1}^{k-1} q_{\beta(j)}\right)\right)\\ &\quad + \sum_{k=1}^{u} p_{\alpha(k)}\left(1 + b_{\alpha(k)}\left(\zeta' + \sum_{j=1}^{v} q_{\beta(j)} + \sum_{i=1}^{k-1} q_{\alpha(i)}\right)\right),\end{aligned}$$

so that for $\Delta$, we derive

$$\Delta = \sum_{k=1}^{u} p_{\alpha(k)} \left( \left( 1 + b_{\alpha(k)} \left( \zeta' + \sum_{i=1}^{k-1} q_{\alpha(i)} \right) \right) \right.$$
$$\left. - \left( 1 + b_{\alpha(k)} \left( \zeta' + \sum_{j=1}^{v} q_{\beta(j)} + \sum_{i=1}^{k-1} q_{\alpha(i)} \right) \right) \right)$$
$$+ \sum_{k=1}^{v} p_{\beta(k)} \left( \left( 1 + b_{\beta(k)} \left( \zeta' + \sum_{i=1}^{u} q_{\alpha(i)} + \sum_{j=1}^{k-1} q_{\beta(j)} \right) \right) \right.$$
$$\left. - \left( 1 + b_{\beta(k)} \left( \zeta' + \sum_{j=1}^{k-1} q_{\beta(j)} \right) \right) \right).$$

Performing cancellations, we obtain

$$\Delta = -\sum_{k=1}^{u} p_{\alpha(k)} b_{\alpha(k)} \sum_{j=1}^{v} q_{\beta(j)} + \sum_{k=1}^{v} p_{\beta(k)} b_{\beta(k)} \sum_{i=1}^{u} q_{\alpha(i)}.$$

Dividing by $\sum_{k=1}^{u} q_{\alpha(k)} \sum_{i=1}^{v} q_{\beta(i)}$, we deduce that $\Delta \le 0$, provided that

$$\frac{\sum_{k=1}^{u} p_{\alpha(k)} b_{\alpha(k)}}{\sum_{k=1}^{u} q_{\alpha(k)}} \ge \frac{\sum_{i=1}^{v} p_{\beta(i)} b_{\beta(k)}}{\sum_{i=1}^{v} q_{\beta(i)}}.$$

For an arbitrary (partial) permutation $\pi$, define the function $\omega(\pi)$ by (10.23). It is easily verified that $\omega(\alpha) > \omega(\beta)$ implies $C_{\max}(\pi^{\alpha\beta}) \le C_{\max}(\pi^{\beta\alpha})$, while $\omega(\alpha) = \omega(\beta)$ implies $C_{\max}(\pi^{\alpha\beta}) = C_{\max}(\pi^{\beta\alpha})$, as required by the definition of the priority function. □

Observe that if (10.23) is applied to a single job $j$, i.e., to a permutation of length one, then the priority function becomes a 1-priority function $\omega(j) = b_j p_j / q_j$, which is consistent with Theorem 10.7.

## 10.3.2 Minimizing Total Completion Time

Consider now the problem of minimizing the sum of the completion times $F(\pi) = \sum C_j$ under the deterioration model (10.2).

Suppose that some subset of jobs is processed starting at time zero in accordance with a permutation $\pi$. The sum of their completion times can be written as

$$F(\pi) = \sum_{k=1}^{|\pi|} (|\pi| - k + 1) p_{\pi(k)} \left(1 + \sum_{j=1}^{k-1} p_{\pi(j)}\right)^A, \tag{10.24}$$

which can be deduced by replacing constant processing times in (2.9) by their actual processing times found by (10.2).

Below, we demonstrate that no priority function exists for the problem of minimizing $F(\pi) = \sum C_j$ for small integer-valued $A$.

First, recall that Theorem 3.4 of Chap. 3 states that for a single machine problem, the priority function for the total completion time objective remains the priority function for the makespan. It is straightforward to verify that the proof of Theorem 3.4 carries over if the jobs are subject to a deterioration effect of the form (10.2). In fact, in that proof all new jobs that should be added to the initial instance should be given normal time $p_j = \varepsilon$. Further, as the contrapositive statement to Theorem 3.4, we derive that if for the problem of minimizing the makespan the objective is not priority-generating, then neither is the objective of minimizing the total completion time. Combining this with Lemma 10.3, we deduce the following statement.

**Lemma 10.4** *For the single machine problem under the deterioration model (10.2) with $A = 3$ the sum of the completion times $\sum C_j$ is not a priority-generating objective function.*

Thus, in the remainder of this section, we focus on the problem with a cumulative deterioration effect applied with $A \in \{1, 2\}$.

**Lemma 10.5** *For the single machine problem under the deterioration model (10.2) with $A \in \{1, 2\}$ the sum of the completion times $F(\pi) = \sum C_{\pi(j)}$ is not a priority-generating objective function.*

*Proof* The proof is similar to that of Lemma 10.3 and is based on Recipe 3.2. Consider the instance of the problem with four jobs with normal processing times

$$p_1 = 1,\ p_2 = 6,\ p_3 = 4,\ p_4 = 7,$$

and let $\alpha = (1, 2)$ and $\beta = (3)$.

Applying (10.24) for $A = 1$, we compare

$$F(\alpha\beta, 4) = 188 > F(\beta\alpha, 4) = 187;$$
$$F(4, \alpha\beta) = 220 < F(4, \beta\alpha) = 226.$$

Similarly, for $A = 2$, we compare

$$F(\alpha\beta, 4) = 1596 > F(\beta\alpha, 4) = 1531;$$
$$F(4, \alpha\beta) = 2092 < F(4, \beta\alpha) = 2098.$$

This demonstrates that a priority function exists neither for $A = 1$ nor for $A = 2$. □

## 10.4 Bibliographic Notes

The cumulative effects have been introduced by Kuo and Yang in a series of papers. They focus on the learning effect (10.2) with $b = 1$ and $A < 0$. For $A < 0$, the optimality of the SPT rule for problems $1\left|p_j(r) = p_j(1 + P_r)^A\right|C_{\max}$ and $1\left|p_j(r) = p_j(1 + P_r)^A\right|\sum C_j$ is proved in Kuo and Yang (2006a) and in Kuo and Yang (2006b), respectively. See Janiak et al. (2011) for a review of scheduling with pure cumulative effects.

The results similar to Theorem 10.1 are proved by Yin et al. (2009), who, however, make an unnecessary assumption that function $f$ is non-increasing.

A special case of effect (10.2) is introduced by Koulamas and Kyparisis (2007). The authors define the actual processing time of job $j$ scheduled in $r$th position of permutation $\pi$ as

$$p_j(r) = p_j\left(1 - \frac{\sum_{h=1}^{r-1} p_{\pi(h)}}{p(N)}\right)^A, \tag{10.25}$$

where $p(N) = \sum_{j=1}^{n} p_j$. It is easy to see that in the case of learning $A \geq 1$, function $f(P_r) = (1 - P_r/p(N))^A$ has a non-decreasing derivatives, i.e., due to Theorems 10.5 and 10.6 for a single machine scheduling problem to minimize function $\Phi \in \left\{C_{\max}, \sum C_j^z, \xi C_{\max} + \eta \sum C_j^z\right\}$ under the cumulative learning effect (10.25), an optimal schedule can be found by the SPT rule. Koulamas and Kyparisis (2007) prove this for $\Phi = C_{\max}$ and $\Phi = \sum C_j$ from the first principles. Notice that (10.25) can be rewritten as

$$p_j(r) = p_j\left(\frac{\sum_{h=r}^{n} p_{\pi(h)}}{\sum_{h=1}^{n} p_{\pi(h)}}\right)^A,$$

which means that the learning effect depends not on the earlier processed jobs, but on the jobs which follow job $\pi(r)$. We share an opinion of Biskup (2008), who finds such a dependence strange. Indeed, as argued in Biskup (2008), if $p_j = 1$, $j \in N$, and $A = 1$, then the actual processing time of the job in the second position is 0.8, 0.9, and 0.95 for $n = 5$, $n = 10$, and $n = 20$, respectively; however, the learning experience gained prior processing and the second job is exactly the same in all three instances.

Cheng et al. (2008) consider a version of a combined cumulative effect such that

$$p_j(r) = p_j\left(\frac{\sum_{h=r}^{n} p_{\pi(h)}}{\sum_{h=1}^{n} p_{\pi(h)}}\right)^A r^a,$$

where $A \geq 1$, $a < 0$. The authors prove that the SPT rule is optimal for minimizing the makespan, as well as for minimizing the sum of the completion times.

A special case of problem $1\left|p_j(r) = p_j(1 + bP_r)^A g(r)\right|C_{\max}$ with $b = 1/p(N)$, $g(r) = r^a$, is studied by Lu et al. (2015) for $0 < A < 1$, $a > 1$, and by Wu and Lee (2008) for $A < 0$, $a < 0$. The result proved by Lu et al. (2015) is a corollary of Theorem 10.5. For the problem of minimizing the sum of the completion times under the same effect, Lu et al. (2015) prove that an optimal permutation is $V$-shaped with respect to the normal processing times. On the other hand, Wu and Lee (2008) prove that the SPT rule is optimal for minimizing the makespan, as well as for minimizing the sum of the completion times; both these results follow from Theorem 10.5.

Huang and Wang (2015) consider a version of a combined cumulative effect such that

$$p_j(r) = p_j \left( 1 + \sum_{h=1}^{r-1} b(h) p_{\pi(h)} \right)^A,$$

where $A \geq 1$ and array $b(r)$, $1 \leq r \leq n$, is non-increasing, i.e., represents a positional learning effect. It is proved that for each problem of minimizing a function $\Phi \in \left\{ C_{\max}, \sum C_j^z, \xi C_{\max} + \eta \sum C_j^z \right\}$ on a single machine, an optimal permutation can be found by the SPT rule.

Job-dependent linear cumulative generalized effect has not been studied prior to Rustogi and Strusevich (2016), even for the variant with $q_j = p_j$; see Theorem 10.7.

The material of Sect. 10.3 is based on Gordon et al. (2008) and Rustogi and Strusevich (2016).

## References

Biskup D (2008) A state-of-the-art review on scheduling with learning effects. Eur J Oper Res 188:315–329

Cheng TCE, Wu C-C, Lee W-C (2008) Some scheduling problems with sum-of-processing-times-based and job-position-based learning effects. Inf Sci 178:2476–2487

Gordon VS, Potts CN, Strusevich VA, Whitehead JD (2008) Single machine scheduling models with deterioration and learning: Handling precedence constraints via priority generation. J Sched 11:357–370

Huang X, Wang J-J (2015) Machine scheduling problems with a position-dependent deterioration. Appl Math Model 39:2897–2908

Janiak A, Krysiak T, Trela R (2011) Scheduling problems with learning and ageing effects: a survey. Decis Mak Manuf Serv 5:19–36

Koulams C, Kyparisis G (2007) Single-machine and two-machine flowshop scheduling with general learning functions. Eur J Oper Res 178:402–407

Kuo W-H, Yang D-L (2006a) Minimizing the makespan in a single machine scheduling problem with a time-based learning effect. Inf Proc Lett 97:64–67

Kuo W-H, Yang D-L (2006b) Minimizing the total completion time in a single-machine scheduling problem with a time-dependent learning effect. Eur J Oper Res 174:1184–1190

Lu Y-Y, Wang J-J, Huang X (2015) Scheduling jobs with position and sum-of-processing-time based processing times. Appl Math Model 39:4013–4021

Rustogi K, Strusevich VA (2016) Single machine scheduling with a generalized job-dependent cumulative effect. J Sched (In press), doi:10.1007/s10951-016-0497-6

Wu C-C, Lee W-C (2008) Single-machine scheduling problems with a learning effect. Appl Math Model 32:1191–1197

Yin Y, Xu D, Sun K, Li H (2009) Some scheduling problems with general position-dependent and time-dependent learning effects. Inf Sci 179:2416–2425

# Chapter 11
# Scheduling on Parallel Machines with Various Effects

In this chapter, we consider scheduling problems on parallel machines, provided that actual processing times of the jobs are subject to various effects.

In a basic model, the jobs of set $N = \{1, 2, \dots, n\}$ are to be processed on $m$ parallel machines $M_1, M_2, \dots, M_m$. Without loss of generality, we assume that $n \geq m$.

Given $m$ parallel machines, a schedule $S$ is associated with a partition of set $N$ into $m$ subsets $N_1, N_2, \dots, N_m$, some of which may be empty. The jobs of set $N_i$, and only those, are assigned to machine $M_i$ and are processed on that machine in accordance with a sequence $\pi^{[i]} = \left(\pi^{[i]}(1), \pi^{[i]}(2), \dots, \pi^{[i]}\left(n^{[i]}\right)\right)$, where $n^{[i]} = N_i$, $1 \leq i \leq m$. Notice that we can see the jobs being split into several machines as being split into several groups, and starting from this chapter, the group-dependent parameters are accompanied by a superscript of the form $[x]$, where $x$ is the group number. This is why in the above text, we write $\pi^{[i]}$ and $n^{[i]}$.

If a job $j \in N$ is scheduled on machine $M_i$, $1 \leq i \leq m$, it is associated with a normal processing time $p_{ij}$. The actual processing time of a job may depend on its position on the machine it is assigned to, or on its start time, or on a combination of both.

Unless stated otherwise, in all problems considered in this chapter, the objective is the total completion time, i.e., the sum of the completion times. Recall that in the case of the makespan objective, problem $P2| |C_{\max}$ on two identical parallel machines is NP-hard in the ordinary sense, even if the processing times are constant; see Sect. 1.3.3.

The following statement is frequently used to solve the problems considered in this chapter.

**Lemma 11.1** *Suppose $n$ jobs are to be scheduled on $m$ machines, such that machine $M_i$, $i \in \{1, 2, \dots, m\}$ has $n^{[i]}$ jobs assigned to it, sequenced in accordance with a permutation $\pi^{[i]} = \left(\pi^{[i]}(1), \dots, \pi^{[i]}(n^{[i]})\right)$, where $\sum_{i=1}^{m} n^{[i]} = n$. Further, sequencing a job $j$ in a position $\pi^{[i]}(r)$ generates a positional weight $W^{[i]}(r)$, so that the overall contribution of job $j$ to the objective function is equal to $W^{[i]}(r)p_j = W^{[i]}(r)p_{\pi^{[i]}(r)}$. A generic objective function can be written as*

$$\Phi(\pi) = \sum_{i=1}^{m} \sum_{r=1}^{n^{[i]}} W^{[i]}(r) p_{\pi^{[i]}(r)}, \tag{11.1}$$

*which can be minimized by using Algorithm Match, if the obtained positional weights are job-independent.*

The proof for the above statement follows from Theorem 2.1.

## 11.1 Combined Effects

In this section, we address parallel machine scheduling problems under combined effects considered in Sect. 8.1.3. The jobs are subject to an additive linear job-independent start-time-dependent effect of the form similar to (8.19). The actual processing time $p_{ij}(\tau; r)$ of job $j$ that is sequenced in position $r$, $1 \le r \le n$, and starts at time $\tau > 0$, on machine $M_i$, is given by

$$p_{ij}(\tau; r) = \left(p_{ij} \pm a^{[i]}\tau\right) g^{[i]}(r),\ 1 \le r \le n,\ 1 \le i \le m,\ j \in N, \tag{11.2}$$

where

- the factor $g^{[i]}(r)$ is used to incorporate a job-independent positional effect, and the superscript "$[i]$" is used to stress that the effect is machine-dependent, i.e., for the same job, a different positional effect may be applied depending on its assignment to particular machine;
- $a^{[i]} > 0$ is a given machine-dependent rate which is common for all jobs; in the case of $p_{ij} + a^{[i]}\tau$, we have a deterioration effect, while $p_{ij} - a^{[i]}\tau$ defines a learning effect.

Notice that if a machine $M_i$, $1 \le i \le m$, is under a start-time-dependent learning effect, i.e., a negative sign is used in (11.2), we must also adopt an additional assumption of the form (8.21), which guarantees that the actual processing times do not assume negative values. Other than this additional assumption, the treatment for both deterioration and learning versions of the effect (11.2) is the same and does not depend on the sign used in front of $a^{[i]}\tau$.

For each machine $M_i$, the function $g^{[i]}$ is given as an array of $n$ numbers, which in general need not be monotone. If array $g^{[i]}(r)$, $1 \le r \le n$, $1 \le i \le m$, is monotone non-decreasing, so that an analogue of (7.10) holds (or non-decreasing, so that an analogue of (7.11) holds), then we have a situation of positional deterioration (or of positional learning, respectively).

### 11.1.1 Identical and Uniform Machines

First, let us consider the problem of minimizing the total completion time on uniform machines under a combined effect (11.2). The actual processing time of a job $j \in N$, if processed on machine $M_i$, $1 \le i \le m$, starting at time $\tau \ge 0$, at a position $r \ge 1$ of permutation $\pi^{[i]}$ is given by

$$p_{ij}(\tau; r) = \frac{\left(p_j \pm a^{[i]}\tau\right) g^{[i]}(r)}{s_i}, \ 1 \le r \le n, \ 1 \le i \le m, \tag{11.3}$$

where $p_j$ is the normal processing time of job $j$ and $s_i$ is the speed of machine $M_i$. The factors $g^{[i]}(r)$ incorporate an arbitrary positional effect, so that the sequence $g^{[i]}(r)$, $1 \le r \le n$, $1 \le i \le m$, is not necessarily monotone. We denote this problem by $Qm\left|p_{ij}(\tau; r) = \left(p_j \pm a^{[i]}\tau\right) g^{[i]}(r)/s_i\right| \sum C_j$.

Recall from Sect. 8.1.3 that for problem $1\left|p_j(\tau; r) = \left(p_j + a\tau\right) g(r)\right| \sum C_j$ on a single machine, the total completion time is given by (8.26). Adopting (8.26) for a schedule $S$ defined by permutations $\pi^{[i]} = \left(\pi^{[i]}(1), \pi^{[i]}(2), \ldots, \pi^{[i]}\left(n^{[i]}\right)\right)$, where $\sum_{i=1}^{m} n^{[i]} = n$, the total completion time of all $n^{[i]}$ jobs on a machine $M_i$ under the effect (11.3) can be given by

$$\sum_{r=1}^{n^{[i]}} C_{\pi^{[i]}(r)} = \sum_{r=1}^{n^{[i]}} \frac{p_{\pi^{[i]}(r)} g^{[i]}(r)}{s_i} \left( \sum_{k=r}^{n^{[i]}} \prod_{q=r+1}^{k} \left(1 \pm a^{[i]} g^{[i]}(q)\right) \right).$$

Thus, for schedule $S$, the total completion time for all jobs on all machines can be written as

$$\sum_{j=1}^{n} C_j(S) = \sum_{i=1}^{m} \sum_{r=1}^{n^{[i]}} \frac{p_{\pi^{[i]}(r)} g^{[i]}(r)}{s_i} \left( \sum_{k=r}^{n^{[i]}} \prod_{q=r+1}^{k} \left(1 \pm a^{[i]} g^{[i]}(q)\right) \right), \tag{11.4}$$

which can be rewritten as the generic objective function (11.1) with the positional weights

$$W^{[i]}(r) = \frac{g^{[i]}(r)}{s_i} \left( \sum_{k=r}^{n^{[i]}} \prod_{q=r+1}^{k} \left(1 \pm a^{[i]} g^{[i]}(q)\right) \right), \ 1 \le r \le n^{[i]}, \ 1 \le i \le m. \tag{11.5}$$

Notice that the positional weights given by (11.5) are job-independent; thus, due to Lemma 11.1, an optimal solution may be delivered by Algorithm Match. However, none of the positional weights $W^{[i]}(r)$ can be computed without exact knowledge of the number $n^{[i]}$ of jobs assigned to machine $M_i$. If the number of jobs to be scheduled on each machine is known in advance, so that $\sum_{i=1}^{m} n^{[i]} = n$, then we will have a total of $n$ positional weights. In order to solve problem $Qm\left|p_{ij}(\tau; r) = \left(p_j \pm a^{[i]}\tau\right) g^{[i]}(r)/s_i\right| \sum C_j$, the obtained $n$ positional weights must be matched with $n$ jobs, so that the generic objective function (11.1) is minimized, and this can be achieved in $O(n \log n)$ time by Algorithm Match.

Unfortunately, for problem $Qm\left|p_{ij}(\tau; r) = \left(p_j \pm a^{[i]}\tau\right)g^{[i]}(r)/s_i\right|\sum C_j$ in its general form, the number of jobs $n^{[i]}$, $1 \le i \le m$, is not known in advance. Moreover, there is no easy way to determine the optimal values of $n^{[i]}$, $1 \le i \le m$. Thus, to solve problem $Qm\left|p_{ij}(\tau; r) = \left(p_j \pm a^{[i]}\tau\right)g^{[i]}(r)/s_i\right|\sum C_j$, we must generate all possible values for $n^{[i]}$, $1 \le i \le m$, so that $\sum_{i=1}^{m} n^{[i]} = n$, and for each instance, solve the resulting problem $Qm\left|\sum_{i=1}^{m} n^{[i]} = n, p_{ij}(\tau; r) = \left(p_j \pm a^{[i]}\tau\right)g^{[i]}(r)/s_i\right|\sum C_j$ by Algorithm Match. Finally, the instance that results in the smallest value of the objective function is chosen as the optimal solution.

To generate all possible values for $n^{[i]}$, $1 \le i \le m$, we need to generate all compositions of $n$ into at most $m$ summands. See Sect. 5.3 for a discussion on combinatorial counting. This generates $O(n^{m-1})$ instances of problem $Qm\left|\sum_{i=1}^{m} n^{[i]} = n, p_{ij}(\tau; r) = \left(p_j \pm a^{[i]}\tau\right)g^{[i]}(r)/s_i\right|\sum C_j$, with known values of $n^{[i]}$, $1 \le i \le m$. The SPT sequence of jobs can be found in advance, and according to Algorithm Match, to match the jobs to the $n$ weights $W^{[i]}(r)$, $1 \le r \le n^{[i]}$, $1 \le i \le m$, these weights have to be sorted in non-decreasing order. For each instance of problem $Qm\left|\sum_{i=1}^{m} n^{[i]} = n, p_{ij}(\tau; r) = \left(p_j \pm a^{[i]}\tau\right)g^{[i]}(r)/s_i\right|\sum C_j$, this sorting requires $O(n \log n)$ time for an arbitrary positional effect. Thus, problem $Qm\left|p_{ij}(\tau; r) = \left(p_j \pm a^{[i]}\tau\right)g^{[i]}(r)/s_i\right|\sum C_j$ can be solved in $O(n^m \log n)$ time.

A similar solution approach can be used to solve the problem on identical parallel machines. For that model, we assume that $g^{[i]}(r) = g(r)$, $1 \le r \le n$, and $a^{[i]} = a$, $s_i = 1$, for all $1 \le i \le m$. Let us denote the resulting problem with identical parallel machines by $Pm\left|p_{ij}(\tau; r) = \left(p_j \pm a\tau\right)g(r)\right|\sum C_j$. There are two important points of differences between problems $Pm\left|p_{ij}(\tau; r) = \left(p_j \pm a\tau\right)g(r)\right|\sum C_j$ and $Qm\left|p_{ij}(\tau; r) = \left(p_j \pm a^{[i]}\tau\right)g^{[i]}(r)/s_i\right|\sum C_j$. First, in the case of uniform machines, even for the classical problem $Qm\left|\,\right|\sum C_j$, an optimal schedule need not use all machines, leaving slower machines empty, whereas in the case of identical machines, an optimal schedule belongs to the class of those schedules in which all machines are used, provided that $n \ge m$. Second, in the case of uniform machines, the machines are not identical with respect to their speeds and positional factors. Thus, for problem $Qm\left|p_{ij}(\tau; r) = \left(p_j \pm a^{[i]}\tau\right)g^{[i]}(r)/s_i\right|\sum C_j$, we need to generate all compositions of $n$ into at most $m$ summands, whereas for problem $Pm\left|p_{ij}(\tau; r) = \left(p_j \pm a\tau\right)g(r)\right|\sum C_j$, we need to generate all partitions of $n$ into exactly $m$ summands. This still generates $O(n^{m-1})$ instances of problem $Pm\left|\sum_{i=1}^{m} n^{[i]} = n, p_{ij}(\tau; r) = \left(p_j \pm a\tau\right)g(r)\right|\sum C_j$, with known values of $n^{[i]}$, $1 \le i \le m$. Thus, problem $Pm\left|p_{ij}(\tau; r) = \left(p_j \pm a\tau\right)g(r)\right|\sum C_j$ can be solved in at most $O(n^m \log n)$ time, i.e., it is not simpler computationally than its counterpart on uniform machines.

The following algorithm formally describes the steps required to solve the problem $Qm\left|p_{ij}(\tau; r) = \left(p_j \pm a^{[i]}\tau\right)g^{[i]}(r)/s_i\right|\sum C_j$.

**Algorithm QSumCombi1**

INPUT: An instance of problem $Qm\left|p_{ij}(\tau;r) = \left(p_j \pm a^{[i]}\tau\right)g^{[i]}(r)/s_i\right|\sum C_j$

OUTPUT: An optimal schedule $S$ defined by the processing sequences $\pi^{[i]}$, $1 \le i \le m$

```
Step 1. Form a list $L$ of jobs by sorting the values $p_j$, $j \in N$, in the SPT order.
Step 2. For each choice of values $n^{[i]}$, $1 \le i \le m$, do
  (a) For each $i$, $1 \le i \le m$, compute the values $W^{[i]}(r)$ by (11.5) and store them
      in a list $G_i$.
  (b) Merge the lists $G_i$ $1 \le i \le m$, into a single list $G$, sorted in non-increasing
      order of its elements.
  (c) Compute the contribution of each job to the objective function, by multiplying
      the $j$th element of list $L$ by the $j$th element of the list $G$, $1 \le j \le n$. If an
      element of $G$ that is matched to job $j$ is $W^{[i]}(r)$, then job $j$ is placed into the
      $r$th position of sequence $\pi^{[i]}$. Compute the objective function as the sum of
      all found products.
Step 3. Output a schedule defined by the processing sequences $\pi^{[i]}$, $1 \le i \le m$,
  that correspond to the choice of $n^{[i]}$, $1 \le i \le m$, that leads to the smallest value
  of the objective function.
```

In Algorithm QSumCombi1 applied to the problem on uniform parallel machines, the values $n^{[i]}$ are generated as possible compositions of $n$ into at most $m$ summands. To solve the problem on identical machines, the same algorithm can be used for $g^{[i]}(r) = g(r)$ and $a^{[i]} = a$, $s_i = 1$, $1 \le i \le m$, provided that the values $n^{[i]}$ are generated as possible partitions of $n$ into exactly $m$ summands.

The following statement holds.

**Theorem 11.1** *Problems* $Qm\left|p_{ij}(\tau;r) = \left(p_j \pm a^{[i]}\tau\right)g^{[i]}(r)/s_i\right|\sum C_j$ *and* $Pm\left|p_{ij}(\tau;r) = \left(p_j \pm a\tau\right)g(r)\right|\sum C_j$ *can be solved by Algorithm QSumCombi1 in* $O(n^m \log n)$ *time, even if the sequence* $g^{[i]}(r), 1 \le r \le n, 1 \le i \le m$, *is not necessarily monotone.*

Notice that if a pure positional effect is considered, the corresponding problems $Qm\left|p_{ij}(r) = p_j g^{[i]}(r)/s_i\right|\sum C_j$ and $Pm\left|p_{ij}(r) = p_j g(r)\right|\sum C_j$ can still be solved by Algorithm QSumCombi1 applied with $a^{[i]} = 0$, $1 \le i \le m$. This still requires $O(n^m \log n)$ time, i.e., the problems with a pure positional effect are not computationally simpler than their counterparts with a combined effect. However, if a pure start-time-dependent effect is considered, the running time can be considerably improved, as shown in Sect. 11.3.

### *11.1.2 Unrelated Machines*

We are given $m$ unrelated parallel machines. The combined effect is applied, so that the actual processing time of job $j$ assigned to the $r$th position on machine $M_i$ and starting at time $\tau \geq 0$ is given by (11.2), where the factors $g^{[i]}(r)$ incorporate an arbitrary job-independent positional effect, so that the sequence $g^{[i]}(r)$, $1 \leq r \leq n$, $1 \leq i \leq m$, is not necessarily monotone. We denote the problem of minimizing the total completion time by $Rm\left|p_{ij}(\tau; r) = \left(p_{ij} \pm a^{[i]}\tau\right)g^{[i]}(r)\right|\sum C_j$.

Under the effect (11.2), for a schedule $S$ on unrelated machines defined by permutations $\pi^{[i]} = \left(\pi^{[i]}(1), \pi^{[i]}(2), \ldots, \pi^{[i]}\left(n^{[i]}\right)\right)$, $1 \leq i \leq m$, the total completion time of all jobs can be adopted from (11.4) and be written as

$$\sum_{j=1}^{n} C_j(S) = \sum_{i=1}^{m}\sum_{r=1}^{n^{[i]}} p_{(i,\pi^{[i]}(r))} g^{[i]}(r) \left(\sum_{k=r}^{n^{[i]}} \prod_{q=r+1}^{k} \left(1 \pm a^{[i]} g^{[i]}(q)\right)\right), \tag{11.6}$$

where $p_{(i,\pi^{[i]}(r))}$ is the normal processing time of a job $j = \pi^{[i]}(r)$ scheduled in position $r$ of permutation $\pi^{[i]}$ on machine $M_i$.

To minimize the objective $\sum C_j(S)$, let us define the cost

$$c_{j,(i,r)} = p_{ij} g^{[i]}(r) \left(\sum_{k=r}^{n^{[i]}} \prod_{q=r+1}^{k} \left(1 \pm a^{[i]} g^{[i]}(q)\right)\right), \tag{11.7}$$

which represents the contribution of a job $j = \pi^{[i]}(r)$, to the objective function. Notice that none of the costs $c_{j,(i,r)}$, $j \in N$, $1 \leq r \leq n^{[i]}$, $1 \leq i \leq m$, can be computed without exact knowledge of the number $n^{[i]}$ of jobs assigned to machine $M_i$. Thus, similar to the solution to problem $Qm\left|p_{ij}(\tau; r) = \left(p_j \pm a^{[i]}\tau\right)g^{[i]}(r)/s_i\right|$ $\sum C_j$ presented in Sect. 11.1.1, we must generate all possible values for $n^{[i]}$, $1 \leq i \leq m$, so that $\sum_{i=1}^{m} n^{[i]} = n$, and for each instance, solve the resulting problem $Rm\left|\sum_{i=1}^{m} n^{[i]} = n, p_{ij}(\tau; r) = \left(p_{ij} \pm a^{[i]}\tau\right)g^{[i]}(r)\right|\sum C_j$. To obtain all possible values for $n^{[i]}$, $1 \leq i \leq m$, we need to generate all compositions of $n$ into at most $m$ summands, which according to (5.12) can be done in $O(n^{m-1})$ ways.

Notice that the objective function $\sum C_j(S)$ given by (11.6) cannot be expressed as (11.1) because the value of the normal processing time is dependent on the machine it is assigned to. Therefore, Lemma 11.1 cannot be applied.

To solve problem $Rm\left|\sum_{i=1}^{m} n^{[i]} = n, p_{ij}(\tau; r) = \left(p_{ij} \pm a^{[i]}\tau\right)g^{[i]}(r)\right|\sum C_j$, define a linear assignment problem with an $n \times n$ cost matrix $\mathbf{C} = \left(c_{j,(i,r)}\right)$ having $n$ rows, each corresponding to a job $j \in N$, and $n$ columns, each corresponding to an available position; see Sect. 4.1 for information on the linear assignment problem (LAP). Number the columns by a string of the form $(i, r)$, where $i$, $1 \leq i \leq m$, refers to a machine index, and $r$, $1 \leq r \leq n^{[i]}$, indicates a position in permutation $\pi^{[i]}$ of jobs assigned to machine $M_i$. More precisely, the value of element $c_{j,(i,r)}$ at the intersection of the $j$th row and the $v$th column of matrix $\mathbf{C}$ is defined by the relation (11.7); here, $v = n^{[1]} + n^{[2]} + \cdots + n^{[i-1]} + r$, where $1 \leq i \leq m$ and $1 \leq r \leq n^{[i]}$.

As a result, the problem of minimizing the objective function (11.6) reduces to LAP written out below

$$\begin{aligned}
\text{minimize} & \sum_{j=1}^{n}\sum_{i=1}^{m}\sum_{r=1}^{n^{[i]}} c_{j,(i,r)}x_{j,(i,r)} & & \\
\text{subject to} & \sum_{i=1}^{m}\sum_{r=1}^{n^{[i]}} x_{j,(i,r)} = 1, & & j = 1,\dots,n; \\
& \sum_{j=1}^{n} x_{j,(i,r)} = 1, & & i = 1,\dots,m,\ r = 1,\dots,n^{[i]}; \\
& x_{j,(i,r)} \in \{0,1\}, & & j = 1,\dots,n,\ i = 1,\dots,m, \\
& & & r = 1,\dots,n^{[i]}.
\end{aligned} \tag{11.8}$$

The algorithm to solve an assignment problem of the form (11.8) has been outlined in Sect. 4.1. The running time of this algorithm is $O(n^3)$. Suppose that for some $i$, $1 \le i \le m$, the solution of the assignment problem (11.8) related to problem $Rm\left|\sum_{i=1}^{m} n^{[i]} = n,\, p_{ij}(\tau; r) = \left(p_{ij} \pm a^{[i]}\tau\right)g^{[i]}(r)\right|\sum C_j$ is found, then $x_{j,(i,r)} = 1$ implies that job $j$ is assigned to the $r$th, $1 \le r \le n^{[i]}$, position of machine $M_i$, $1 \le i \le m$. This process is repeated $O\left(n^{m-1}\right)$ times, for each generated instance of problem $Rm\left|\sum_{i=1}^{m} n^{[i]} = n,\, p_{ij}(\tau; r) = \left(p_{ij} \pm a^{[i]}\tau\right)g^{[i]}(r)\right|\sum C_j$, and the instance that delivers the smallest value of the objective function is chosen to produce an optimal solution to problem $Rm\left|p_{ij}(\tau; r) = \left(p_{ij} \pm a^{[i]}\tau\right)g^{[i]}(r)\right|\sum C_j$. The following statement holds.

**Theorem 11.2** *Problem $Rm\left|p_{ij}(\tau; r) = \left(p_{ij} \pm a^{[i]}\tau\right)g^{[i]}(r)\right|\sum C_j$, in which unrelated machines are under a combined job-independent effect (11.2), can be solved in $O\left(n^{m+2}\right)$ time, by reducing the problem to solving a series of linear assignment problems, even if the sequence $g^{[i]}(r)$, $1 \le r \le n$, $1 \le i \le m$, $j \in N$, is not necessarily monotone.*

Notice that if a pure positional effect is considered, i.e., $a^{[i]} = 0$, $1 \le i \le m$, the resulting problem $Rm\left|p_{ij}(r) = p_{ij}g^{[i]}(r)\right|\sum C_j$ cannot be solved any faster than in $O\left(n^{m+2}\right)$ time. However, for the same running time, it may be possible to consider a more general positional effect instead.

Consider problem $Rm\left|p_{ij}(r) = p_{ij}g_j^{[i]}(r)\right|\sum C_j$, with job-dependent positional effects, so that the actual processing time $p_{ij}(r)$ of job $j$ that is sequenced in position $r$ on machine $M_i$ is given by

$$p_{ij}(r) = p_{ij}g_j^{[i]}(r),\ 1 \le r \le n,\ 1 \le i \le m,\ j \in N. \tag{11.9}$$

For each machine $M_i$ and job $j \in N$, the function $g_j^{[i]}$ is given as an array of $n$ numbers, which in general need not be monotone. If monotone, the array represents either a positional deterioration effect

$$1 = g_j^{[i]}(1) \le g_j^{[i]}(2) \le \cdots \le g_j^{[i]}(n),\ 1 \le i \le m,\ j \in N, \tag{11.10}$$

or a positional learning effect

$$1 = g_j^{[i]}(1) \geq g_j^{[i]}(2) \geq \cdots \geq g_j^{[i]}(n),\ 1 \leq i \leq m,\ j \in N. \tag{11.11}$$

Under the effect (11.9) with non-monotone positional factors, for a schedule $S$ on unrelated machines defined by permutations $\pi^{[i]} = \left(\pi^{[i]}(1), \pi^{[i]}(2), \ldots, \pi^{[i]}\left(n^{[i]}\right)\right)$, the total completion time of all jobs can be adopted from (11.6) and be written as

$$\sum_{j=1}^{n} C_j(S) = \sum_{i=1}^{m} \sum_{r=1}^{n^{[i]}} p_{\left(i,\pi^{[i]}(r)\right)} \left(n^{[i]} - r + 1\right) g_{\pi^{[i]}(r)}^{[i]}(r).$$

To minimize the objective $\sum C_j(S)$, define the cost

$$c_{j,(i,r)} = p_{ij}\left(n^{[i]} - r + 1\right) g_j^{[i]}(r),$$

for known values of $n^{[i]}$, $1 \leq i \leq m$, and solve an LAP of the form (11.8). Similar to problem $Rm\left|p_{ij}(\tau; r) = \left(p_{ij} \pm a^{[i]}\tau\right) g^{[i]}(r)\right| \sum C_j$, all possible values for $n^{[i]}$, $1 \leq i \leq m$, can be generated in $O(n^{m-1})$ ways, and for each composition, an LAP can be solved in $O\left(n^3\right)$ time, so that the following statement holds.

**Theorem 11.3** *Problem* $Rm\left|p_{ij}(r) = p_{ij} g_j^{[i]}(r)\right| \sum C_j$, *in which unrelated machines are under a job-dependent positional effect (11.9), can be solved in* $O\left(n^{m+2}\right)$ *time, by reducing the problem to solving a series of linear assignment problems, even if the sequence* $g_j^{[i]}(r)$, $1 \leq r \leq n$, $1 \leq i \leq m$, $j \in N$, *is not necessarily monotone.*

Note that if a pure time-dependent effect is considered, the running time can be considerably improved, as shown in Sect. 11.3.

## 11.2 Start-Time-Dependent Job-Dependent Linear Effects

In this section, we address a problem of scheduling jobs on $m$ identical parallel machines under an additive job-dependent linear deterioration effect. Each job $j \in N$ is associated with a normal processing time $p_j$ and a non-negative rate $a_j$. We study a deterioration effect under which the actual processing time $p_j(\tau)$ of job $j$ that starts at time $\tau$ is given by

$$p_j(\tau) = p_j + a_j\tau. \tag{11.12}$$

For the problems on a single machine, this effect is considered in Sect. 8.13.

For the purpose of establishing the complexity status of relevant scheduling problems, we consider a simplified effect $p_j(\tau) = a_j\tau$. Such an effect can be seen as a special case of the linear additive effect (11.12), provided that $p_j = 0$, $j \in N$. Besides, it is also a special case of a multiplicative effect $p_j(\tau) = p_j a_j \tau$, provided that $p_j = 1$, $j \in N$.

Unlike other start-time-dependent effects considered in this book, under the effect $p_j(\tau) = a_j\tau$, there is no guarantee that for the job that is scheduled first the actual start time is equal to its normal time. Moreover, if the first job starts at time zero, then the problem degenerates, with all actual processing times and all completion times to become zero. To avoid such a degeneracy phenomenon, it is common to assume that the first job on each machine starts at time $\tau_0 > 0$.

Consider a schedule $S$ associated with permutations $\pi^{[i]} = \left(\pi^{[i]}(1), \pi^{[i]}(2), \ldots, \pi^{[i]}\left(n^{[i]}\right)\right)$, $1 \le i \le m$. We deduce

$$\begin{aligned} C_{\pi^{[i]}(1)} &= \tau_0 + a_{\pi^{[i]}(1)}\tau_0 = \tau_0\left(1 + a_{\pi^{[i]}(1)}\right); \\ C_{\pi^{[i]}(2)} &= C_{\pi^{[i]}(1)} + a_{\pi^{[i]}(2)}C_{\pi^{[i]}(1)} = \left(1 + a_{\pi^{[i]}(2)}\right)C_{\pi^{[i]}(1)} = \tau_0\left(1 + a_{\pi^{[i]}(1)}\right)\left(1 + a_{\pi^{[i]}(2)}\right). \end{aligned}$$

Extending this argument, it is easy to verify that

$$C_{\pi^{[i]}(r)} = \tau_0 \prod_{k=1}^{r}\left(1 + a_{\pi^{[i]}(k)}\right),\ 1 \le r \le n^{[i]}.$$

For schedule $S$ on $m$ parallel machines, denote the completion time of the last job scheduled on machine $M_i$ by $C^{(i)}$, $1 \le i \le m$.

### *11.2.1 Minimizing Makespan: Complexity and Approximation Scheme*

We show that problem $P2\left|p_j(\tau) = a_j\tau\right|C_{\max}$ of minimizing the makespan on two identical parallel machines is NP-hard in the strong sense, but still even a more general problem $Pm\left|p_j(\tau) = p_j + a_j\tau\right|C_{\max}$ admits an FPTAS. Recall that problem $P2|\ |C_{\max}$ is NP-hard in the ordinary sense.

In order to establish the complexity status of problem $P2\left|p_j(\tau) = a_j\tau\right|C_{\max}$, we provide polynomial reduction of Product Partition to a decision version of problem $P2\left|p_j(\tau) = a_j\tau\right|C_{\max}$. For completeness, we formulate Product Partition below.

Product Partition: Given the index set $R = \{1, \ldots, r\}$ and positive integers $e_1, \ldots, e_r$ such that $\prod_{i \in R} e_i = E^2$, is it possible to partition set $R$ into disjoint subsets $R_1$ and $R_2$ such that $\prod_{i \in R_1} e_i = \prod_{i \in R_2} e_i = E$?

Recall that Product Partition is NP-complete in the strong sense, see Sect. 1.3.2 for discussion. Notice that without loss of generality, we may assume that $e_j \ge 2$.

**Theorem 11.4** *Problem $P2\left|p_j(\tau) = a_j\tau\right|C_{\max}$ is NP-hard in the strong sense.*

*Proof* Given an arbitrary instance of Product Partition, define the following instance of the decision version of problem $P2\left|p_j(\tau) = a_j\tau\right|C_{\max}$.

There are $n = r + 2$ jobs, such that

$$a_j = e_j - 1, \ 1 \le j \le r;$$
$$a_{r+1} = a_{r+2} = 2E - 1.$$

We show that PRODUCT PARTITION has a solution if and only if for the constructed instance of problem $P2\left|p_j(\tau) = a_j\tau\right|C_{\max}$ there exists a schedule $S_0$ for which the value of the objective function is at most $Y = 2\tau_0 E^2$ for an arbitrary chosen start time $\tau_0 > 0$.

First, assume that PRODUCT PARTITION has a solution, and $R_1$ and $R_2$ are the found subset of $R$, i.e., $\prod_{i \in R_1} e_i = \prod_{i \in R_2} e_i = E$. Then, a required schedule $S_0$ exists and can be found as follows:

- machine $M_1$ processes jobs of set $R_1$ in the order of their numbering, followed by job $r + 1$;
- machine $M_2$ processes jobs of set $R_2$ in the order of their numbering, followed by job $r + 2$.

We see that

$$C_{r+1} = \tau_0 \left( \prod_{j \in R_1} (1 + a_j) \right) (1 + a_{r+1}) = \tau_0 E(2E) = 2\tau_0 E^2;$$

$$C_{r+2} = \tau_0 \left( \prod_{j \in R_2} (1 + a_j) \right) (1 + a_{r+2}) = \tau_0 E(2E) = 2\tau_0 E^2,$$

so that $C_{\max}(S_0) = 2\tau_0 E^2 = Y$.

Now assume that there exists a required schedule $S_0$, in which the makespan does not exceed $Y$.

If in schedule $S_0$ jobs $r + 1$ and $r + 2$ are processed on the same machine, then the completion time of the later scheduled job is at least $\tau_0(1 + a_{r+1})(1 + a_{r+2}) = \tau_0(2E)(2E) = 4\tau_0 E^2 > Y$. Thus, jobs $r + 1$ and $r + 2$ are assigned to different machines. Without loss of generality, we may assume that in $S_0$ job $r + 1$ is assigned to machine $M_1$, while job $r + 2$ is assigned to machine $M_2$.

Let $T \subseteq R$ denote the set of jobs that in schedule $S_0$ are processed on machine $M_1$. Compute

$$C^{(1)} = \tau_0 \left( \prod_{j \in T} (1 + a_j) \right) (1 + a_{r+1}) = 2\tau_0 \left( \prod_{j \in T} (1 + a_j) \right) E;$$

$$C^{(2)} = \tau \left( \prod_{j \in R \setminus T} (1 + a_j) \right) (1 + a_{r+2}) = 2\tau_0 \left( \prod_{j \in R \setminus T} (1 + a_j) \right) E.$$

Since $C_{\max}(S_0) = \max\{C^{(1)}, C^{(2)}\} \le Y$, we deduce that

$$\max\left\{\prod_{j \in T}(1 + a_j), \prod_{j \in R \setminus T}(1 + a_j)\right\} \le E.$$

This is only possible if

$$\prod_{j \in T}(1 + a_j) = \prod_{j \in R \setminus T}(1 + a_j) = E,$$

i.e., if sets $T$ and $R \setminus T$ form a solution to Product Partition.

Since Product Partition is NP-complete in the strong sense, it follows that problem $P2|p_j(\tau) = a_j\tau|C_{\max}$ is NP-hard in the strong sense. □

Unlike majority problems that are NP-hard in the strong sense, problem $P2|p_j(\tau) = a_j\tau|C_{\max}$ and, in fact its generalization, problem $Pm|p_j(\tau) = p_j + a_j\tau|C_{\max}$ admit a fully polynomial-time approximation scheme (FPTAS); see Sect. 1.3.4 for relevant definitions and discussions.

Any schedule for problem $Pm|p_j(\tau) = p_j + a_j\tau|C_{\max}$ is defined by

**(a)** a split of the set of jobs $N$ into $m$ subsets $N_1, N_2, \dots, N_m$ such that the jobs of set $N_i$ and only those are processed on machine $M_i$, $1 \le i \le m$;

**(b)** a sequence in accordance with which the jobs of set $N_i$ are processed on machine $M_i$, $1 \le i \le m$.

It follows from Theorem 8.6 that given a partition $N_1, N_2, \dots, N_m$ into $m$ subsets, the smallest makespan is attained if the jobs of set $N_i$ are sequenced in non-decreasing order of the ratios $p_j/a_j$. Thus, throughout this section, we assume that the jobs are numbered in such a way that

$$\frac{p_1}{a_1} \le \frac{p_2}{a_2} \le \cdots \le \frac{p_n}{a_n}. \tag{11.13}$$

This allows us to develop a dynamic programming (DP) algorithm for solving problem $Pm|p_j(\tau) = p_j + a_j\tau|C_{\max}$. Informally, a DP algorithm scans the jobs in the order of their numbering and tries to assign the next job to become the last on a machine. Formally, let $\mathbf{x} = (x_1, x_2, \dots, x_n)$ be an $n$ dimensional assignment vector with integer components from the set $\{0, 1, \dots, m\}$. Each vector $\mathbf{x}$ defines a (partial) schedule for problem $Pm|p_j(\tau) = p_j + a_j\tau|C_{\max}$ in the following way: If $x_j = i \ge 1$, then job $j$, $1 \le j \le n$, is processed on machine $M_i$, $1 \le i \le m$, while $x_j = 0$, then job $j$ is not yet assigned for processing.

Given a vector $\mathbf{x} = (x_1, x_2, \dots, x_n)$ and a $k$, $1 \le k \le n$, let $\mathbf{x}_{[k]} = (x_1, x_2, \dots, x_k, 0, \dots, 0)$ denote a "truncated" vector obtained from $\mathbf{x}$ by replacing $n-k$ last components by zero. Each vector $\mathbf{x}_{[k]}$ is a partial assignment vector and is associated with $m$ values $F^i_{[k]}(\mathbf{x}_{[k]})$, $1 \le i \le m$, so that under the assignment $\mathbf{x}_{[k]}$ the last job assigned to machine $M_i$ completes at time $F^i_{[k]}(\mathbf{x}_{[k]})$.

**Algorithm ParMachLinDeterDP**

**Step 1**. If required, renumber the jobs in accordance with (11.13). Define $\mathbf{x}_{[0]} := (0, 0, \ldots, 0)$, and $F^i_{[0]}(\mathbf{x}_{[0]}) := 0$, $1 \leq i \leq m$.
**Step 2**. For $k$ from 1 to $n$ do

- **(a)** For each vector $\mathbf{x}_{[k-1]} \in Y_{k-1}$, form $m$ vectors $\mathbf{x}_{[k]}$ obtained from $\mathbf{x}_{[k-1]}$ by keeping all its components unchanged, except the $k$ -th component $x^{(k)}_{[k]}$ which is made equal to $i$, $1 \leq i \leq m$.
- **(b)** For each vector $\mathbf{x}_{[k]}$ obtained from a vector $\mathbf{x}_{[k-1]}$ compute

$$F^i_{[k]}(\mathbf{x}_{[k]}) := \begin{cases} F^i_{[k-1]}(\mathbf{x}_{[k-1]}) + p_k + a_k F^i_{[k-1]}(\mathbf{x}_{[k-1]}), & \text{if } \mathbf{x}^{(k)}_k = i \\ F^i_{[k-1]}(\mathbf{x}_{[k-1]}), & \text{otherwise.} \end{cases} \tag{11.14}$$

**Step 3**. The optimal makespan is equal to $\min \max\{F^i_{[n]}(\mathbf{x}_{[n]}) | 1 \leq i \leq m\}$, where the minimum is taken over all vectors $\mathbf{x}_{[n]}$. An optimal assignment can be found by backtracking.

Notice that the running time of Algorithm ParMachLinDeterDP is not pseudopolynomial with respect to the length of the input of the problem. Still, the algorithm can be converted into an FPTAS.

Before we present an FPTAS for problem $Pm\left|p_j(\tau) = p_j + a_j\tau\right|C_{\max}$, we describe the following auxiliary partitioning procedure. Let $A$ be a subset of the assignment vectors and $H(\mathbf{x})$ be a function which puts a positive number into correspondence to a vector $\mathbf{x} \in A$. For a given positive $v$, we want to partition set $A$ into $r_H$ disjoint subsets $A^H_1, \ldots A^H_{r_H}$ such that the inequality

$$\left|H(\mathbf{x}') - H(\mathbf{x}'')\right| \leq v \min\{H(\mathbf{x}'), H(\mathbf{x}'')\} \tag{11.15}$$

holds for any pairs $\mathbf{x}'$ and $\mathbf{x}''$ of vectors of each set $A_\ell$, $1 \leq \ell \leq r_H$.

**Procedure Parti(*A*, *H*, *v*)**

**Step 1**. If required, renumber the vectors of set $A$ in accordance with

$$0 \leq H(\mathbf{x}_1) \leq \cdots \leq H(\mathbf{x}_{|A|}). \tag{11.16}$$

Define $h_1 := 0$. Set $\ell := 1$.
**Step 2**. Starting from vector $\mathbf{x}_{h_{\ell-1}+1}$, try to determine the index $h_\ell$ such that

$$H(\mathbf{x}_{h_\ell}) \leq (1+v)H(\mathbf{x}_{h_{\ell-1}+1}), \quad H(\mathbf{x}_{h_\ell+1}) > (1+v)H(\mathbf{x}_{h_{\ell-1}+1}).$$

If $h_\ell$ exists and $h_\ell < |A|$, form set $A^H_\ell := \{\mathbf{x}_{h_{\ell-1}+1}, \ldots, \mathbf{x}_{h_\ell}\}$, update $\ell := \ell + 1$ and repeat Step 2; otherwise, define $r_H := \ell - 1$, form set $A^H_{r_H} := \{\mathbf{x}_{h_{\ell-1}+1}, \ldots, \mathbf{x}_{|A|}\}$ and Stop.

If the values $H(\mathbf{x})$ are known for all $\mathbf{x} \in A$, then the procedure requires $O(|A| \log|A|)$ time. Lemmas below establish several properties that hold for a partition found by Procedure Parti($A$, $H$, $v$).

**Lemma 11.2** *Let subsets $A_1^H, \ldots A_{r_H}^H$ form a partition of set $A$ found by Procedure Parti($A$, $H$, $v$). Then (11.15) holds for any pair $\mathbf{x}'$ and $\mathbf{x}''$ of vectors of each set $A_\ell^H$, $1 \le \ell \le r_H$.*

*Proof* Take an arbitrary set $A_\ell^H$ which contains at least two vectors $\mathbf{x}'$ and $\mathbf{x}''$. Due to the numbering (11.16), we have that

$$\min\{H(\mathbf{x}'), H(\mathbf{x}'')\} \ge H(\mathbf{x}_{h_{\ell-1}+1}), \ \ \max\{H(\mathbf{x}'), H(\mathbf{x}'')\} \le (1+v)H(\mathbf{x}_{h_{\ell-1}+1}).$$

Since $\max\{H(\mathbf{x}'), H(\mathbf{x}'')\} - \min\{H(\mathbf{x}'), H(\mathbf{x}'')\} = |H(\mathbf{x}') - H(\mathbf{x}'')|$, it follows that

$$|H(\mathbf{x}') - H(\mathbf{x}'')| \le vH(\mathbf{x}_{h_{\ell-1}+1}) \le v \min\{H(\mathbf{x}'), H(\mathbf{x}'')\},$$

as required. □

**Lemma 11.3** *Let the vectors of set $A$ be numbered in accordance with (11.16). If $H(\mathbf{x}_1) \ge 1$ and $v \le 1$, then Procedure Parti($A$, $H$, $v$) finds a partition of $A$ in $r_H$ subsets such that*

$$r_H \le \frac{\log H(\mathbf{x}_{|A|})}{v} + 2. \tag{11.17}$$

*Proof* Since $\log H(\mathbf{x}_{|A|}) > 0$ for $H(\mathbf{x}_{|A|}) \ge 1$, we see that (11.17) holds, provided that $r_H \le 2$.

For $r_H > 2$, we see that

$$H(\mathbf{x}_{h_1+1}) > (1+v)H(\mathbf{x}_{h_0+1}) = (1+v)H(\mathbf{x}_1)$$

and

$$H(\mathbf{x}_{h_\ell+1}) > (1+v)H(\mathbf{x}_{h_{\ell-1}+1}) \ge (1+v)^\ell H(\mathbf{x}_1) \ 2 \le \ell \le r_H - 1,$$

so that

$$H(\mathbf{x}_{|A|}) \ge H(\mathbf{x}_{h_{r_H-1}+1}) > (1+v)^{r_H-1} H(\mathbf{x}_1),$$

from which (11.17) can be derived. □

Now, we are ready to state and analyze an FPTAS for problem $Pm|p_j(\tau) = p_j + a_j\tau|C_{\max}$.

**Algorithm ParMachLinDeterFPTAS**

**Step 1**. If required, renumber the jobs in accordance with (11.13). Define $\mathbf{x}_{[0]} := (0, 0, \ldots, 0)$, and $F^i_{[0]}\big(\mathbf{x}_{[0]}\big) := 0$, $1 \le i \le m$. Form set $Y_0 := \big\{\mathbf{x}_{[0]}\big\}$. For a given $\varepsilon$, compute $v := \frac{\varepsilon}{en}$, where $e = \lim_{n\to\infty}\big(1+\frac{1}{n}\big)^n = 2.71828\ldots$ Set $k := 1$.

**Step 2**. For the current $k$, perform the following

**(a)** For each vector $\mathbf{x}_{[k-1]} \in Y_{k-1}$ form $m$ vectors $\mathbf{x}_{[k]}$ obtained from $\mathbf{x}_{[k-1]}$ by keeping all its components unchanged, except the $k$th component $x^{(k)}_{[k]}$ which is made equal to $i$, $1 \le i \le m$. Call the resulting set of vectors $Y'_k$.

**(b)** For each vector $\mathbf{x}_{[k]} \in Y'_k$ obtained from a vector $\mathbf{x}_{[k-1]} \in Y_{k-1}$, compute $F^i_{[k]}\big(\mathbf{x}_{[k]}\big)$ by (11.14).

**(c)** If $k = n$, go to Step 3.

**(d)** If $k < n$, then for each $i$, $1 \le i \le m$, call Procedure Parti($Y'_k$, $F^i_{[k]}$, $v$) to obtain a partition of $Y'_k$ into disjoint sets $Y_1^{F^i}, \ldots, Y_{r_{F_i}}^{F^i}$. For each choice of integers $u_i \in \big\{1, \ldots, r_{F_i}\big\}$, $1 \le i \le m$, introduce string $\big(u_1, u_{2,} \cdots, u_m\big)$. Using string $\big(u_1, u_{2,} \cdots, u_m\big)$ as an identifier, define a set

$$Y_{(u_1,u_2,\cdots,u_m)} := \bigcap_{i=1}^{m} Y_{u_i}^{F_i},$$

and for each non-empty set $Y_{(u_1,u_2,\cdots,u_m)}$ find the assignment vector $\mathbf{x}_{(u_1,u_2,\cdots,u_m)}$ which delivers the smallest makespan for all associated partial schedules, i.e.,

$$\begin{aligned}&\max\big\{F^i_{[k]}\big(\mathbf{x}_{(u_1,u_2,\cdots,u_m)}\big)|1 \le i \le m\big\}\\ &= \min\big\{\max\big\{F^i_{[k]}\big(\mathbf{x}_{[k]}\big)|1 \le i \le m\big\}|\mathbf{x}_{[k]} \in Y_{(u_1,u_2,\cdots,u_m)}\big\}.\end{aligned}$$

Form the set

$$Y_k := \big\{\mathbf{x}_{(u_1,u_2,\cdots,u_m)} \in Y_{(u_1,u_2,\cdots,u_m)}|u_i \in \big\{1, \ldots, r_{F_i}\big\}, 1 \le i \le m\big\}.$$

Update $k := k + 1$ and repeat Step 2.

**Step 3**. Output vector $\mathbf{x}_\varepsilon \in Y_n$ such that

$$\max\big\{F^i_{[n]}(\mathbf{x}_\varepsilon)|1 \le i \le m\big\} = \min\big\{\max\big\{F^i_{[n]}\big(\mathbf{x}_{[n]}\big)|1 \le i \le m\big\}|\mathbf{x}_{[n]} \in Y_n\big\}.$$

A schedule that correspond to this solution can be found by determining an assignment jobs to machines backtracking. The makespan of the corresponding schedule is equal to $\max\big\{F^i_{[n]}(\mathbf{x}_\varepsilon)|1 \le i \le m\big\}$.

Define

$$v_1 := v;\ \ v_k := v + (1 + v)v_{k-1}, 2 \le k \le n. \tag{11.18}$$

For problem $Pm\left|p_j(\tau) = p_j + a_j\tau\right|C_{\max}$, let $\mathbf{x}^* = \left(x_1^*, x_2^*, \dots, x_n^*\right)$ be an assignment vector that defines an optimal schedule $S^*$. For $k$, $1 \le k \le n$, let $\mathbf{x}_{[k]}^* = \left(x_1^*, \dots, x_k^*, 0, \dots 0\right)$ denote the truncated optimal assignment vector that differs from $\mathbf{x}^*$ by the fact that its last $n-k$ components are zero.

In order to evaluate the accuracy of Algorithm ParMachLinDeterFPTAS, we first prove the following auxiliary statement.

**Lemma 11.4** *For each $k$, $1 \le k \le n$, Algorithm ParMachLinDeterFPTAS finds a vector $\mathbf{x}'_{[k]} \in Y'_k$ with the last $n-k$ zero components such that for the truncated optimal assignment vector $\mathbf{x}^*_{[k]}$ the inequality*

$$\left|F^i_{[k]}\left(\mathbf{x}^*_{[k]}\right) - F^i_{[k]}\left(\mathbf{x}'_{[k]}\right)\right| \le \upsilon_k F^i_{k[k]}\left(\mathbf{x}^*_{[k]}\right) \tag{11.19}$$

*holds for each $i$, $1 \le i \le m$.*

*Proof* The proof is by induction. For $k = 1$, suppose that a string $\left(u_1, u_{2,} \cdots, u_m\right)$ is such that $\mathbf{x}^*_{[1]} = \left(x_1^*, 0, \dots, 0\right) \in Y_{(u_1,u_2,\cdots,u_m)} \subseteq Y'_1$. Let $\mathbf{x}_{(u_1,u_2,\cdots,u_m)}$ be an assignment vector found in set $Y_{(u_1,u_2,\cdots,u_m)}$ in Step 2(d) of the first iteration of the scheme. It is possible that $\mathbf{x}_{(u_1,u_2,\cdots,u_m)}$ is different from $\mathbf{x}^*_{[1]}$; however, it follows from Lemma 11.2 applied to $A = Y'_1$ and $H = F^i_{[1]}$ that inequality

$$\left|F^i_{[1]}\left(\mathbf{x}^*_{[1]}\right) - F^i_{[1]}\left(\mathbf{x}_{(u_1,u_2,\cdots,u_m)}\right)\right| \le \upsilon F^i_{[1]}\left(\mathbf{x}^*_{[1]}\right) = \upsilon_1 F^i_{[1]}\left(\mathbf{x}^*_{[1]}\right)$$

holds for each $i$, $1 \le i \le m$. Thus, we may set $\mathbf{x}'_{[1]} := \mathbf{x}_{(u_1,u_2,\cdots,u_m)}$ to obtain a required vector.

Assume now that for each $k$, $1 \le k \le q < n$, the inequality (11.19) holds for each $i$, $1 \le i \le m$.

Suppose that for some a string $(u_1, u_2, \cdots, u_m)$, $\mathbf{x}'_{[q]} = \mathbf{x}_{(u_1,u_2,\cdots,u_m)}$, i.e.,

$$\left|F^i_{[q]}\left(\mathbf{x}^*_{[q]}\right) - F^i_{[q]}\left(\mathbf{x}_{(u_1,u_2,\cdots,u_m)}\right)\right| \le \upsilon_q F^i_{[q]}\left(\mathbf{x}^*_{[q]}\right)$$

holds for each $i$, $1 \le i \le m$. Take vector $\mathbf{x}^*_{[q+1]} = \left(x_1^*, x_2^*, \dots, x_q^*, x_{q+1}^*, 0, \dots, 0\right)$ and define vector $\tilde{\mathbf{x}}_{(u_1,u_2,\cdots,u_m)}$ obtained from vector $\mathbf{x}_{(u_1,u_2,\cdots,u_m)}$ by making its $(q+1)$th component equal to $x^*_{q+1}$. Recall that the value $x^*_{q+1}$ corresponds to the number of machine to which job $q+1$ is assigned in an optimal schedule. For $i = x^*_{q+1}$, compute

$$\begin{aligned}
&\left|F^i_{[q+1]}\left(\mathbf{x}^*_{[q+1]}\right) - F^i_{[q+1]}\left(\tilde{\mathbf{x}}_{(u_1,u_2,\cdots,u_m)}\right)\right| \\
&= \Big|F^i_{[q]}\left(\mathbf{x}^*_{[q]}\right) + p_{q+1} + a_{q+1}F^i_{[q]}\left(\mathbf{x}^*_{[q]}\right) \\
&\quad - F^i_{[q]}\left(\mathbf{x}_{(u_1,u_2,\cdots,u_m)}\right) - p_{q+1} - a_{q+1}F^i_{[q]}\left(\mathbf{x}_{(u_1,u_2,\cdots,u_m)}\right)\Big| \\
&= \left|\left(1 + a_{q+1}\right)\left(F^i_{[q]}\left(\mathbf{x}^*_{[q]}\right) - F^i_{[q]}\left(\mathbf{x}_{(u_1,u_2,\cdots,u_m)}\right)\right)\right| \le \left(1 + a_{q+1}\right)\upsilon_q F^i_{[q]}\left(\mathbf{x}^*_{[q]}\right) \\
&\le \upsilon_q\left(F^i_{[q]}\left(\mathbf{x}^*_{[q]}\right) + p_{q+1} + a_{q+1}F^i_{[q]}\left(\mathbf{x}^*_{[q]}\right)\right) = \upsilon_q F^i_{[q+1]}\left(\mathbf{x}^*_{[q+1]}\right).
\end{aligned}$$

For $i \neq x^*_{k+1}$, we deduce

$$\left|F^i_{[q+1]}\left(\mathbf{x}^*_{[q+1]}\right) - F^i_{[q+1]}\left(\tilde{\mathbf{x}}_{(u_1,u_2,\cdots,u_m)}\right)\right| = \left|F^i_{[q]}\left(\mathbf{x}^*_{[q]}\right) - F^i_{[q]}\left(\mathbf{x}_{(u_1,u_2,\cdots,u_m)}\right)\right|.$$

Thus, the inequality

$$\left|F^i_{[q+1]}\left(\mathbf{x}^*_{[q+1]}\right) - F^i_{[q+1]}\left(\tilde{\mathbf{x}}_{(u_1,u_2,\cdots,u_m)}\right)\right| \leq v_q F^i_{[q+1]}\left(\mathbf{x}^*_{[q+1]}\right) \tag{11.20}$$

holds for each $i$, $1 \leq i \leq m$. In particular, (11.20) implies that

$$F^i_{[q+1]}\left(\tilde{\mathbf{x}}_{(u_1,u_2,\cdots,u_m)}\right) \leq \left(1 + v_q\right) F^i_{[q+1]}\left(\mathbf{x}^*_{[q+1]}\right) \tag{11.21}$$

holds for each $i$, $1 \leq i \leq m$.

Now, let a string $(v_1, v_2, \cdots, v_m)$ be such that $\tilde{\mathbf{x}}_{(u_1,u_2,\cdots,u_m)} \in Y_{(v_1,v_2,\cdots,v_m)} \subseteq Y'_{q+1}$. Let $\mathbf{x}_{(v_1,v_2,\cdots,v_m)}$ be an assignment vector found in set $Y_{(v_1,v_2,\cdots,v_m)}$ in Step 2(d) of the $(q+1)$th iteration of the scheme. It is possible that $\mathbf{x}_{(v_1,v_2,\cdots,v_m)}$ is different from $\tilde{\mathbf{x}}_{(u_1,u_2,\cdots,u_m)}$; however, it follows from Lemma 11.2 applied to $A = Y'_{q+1}$ and $H = F^i_{[q+1]}$ that inequality

$$\begin{aligned}
&\left|F^i_{[q+1]}\left(\tilde{\mathbf{x}}_{(u_1,u_2,\cdots,u_m)}\right) - F^i_{[q+1]}\left(\mathbf{x}_{(v_1,v_2,\cdots,v_m)}\right)\right| \\
&\leq v F^i_{[q+1]}\left(\tilde{\mathbf{x}}_{(u_1,u_2,\cdots,u_m)}\right) \leq v\left(1 + v_q\right) F^i_{[q+1]}\left(\mathbf{x}^*_{[q+1]}\right)
\end{aligned} \tag{11.22}$$

holds for each $i$, $1 \leq i \leq m$, where the last inequality of (11.22) is due to (11.21).

Using (11.20) and (11.22), we deduce

$$\begin{aligned}
&\left|F^i_{[q+1]}\left(\mathbf{x}^*_{[q+1]}\right) - F^i_{[q+1]}\left(\tilde{\mathbf{x}}_{(u_1,u_2,\cdots,u_m)}\right)\right| \\
&= \Big|F^i_{[q+1]}\left(\mathbf{x}^*_{[q+1]}\right) - F^i_{[q+1]}\left(\tilde{\mathbf{x}}_{(u_1,u_2,\cdots,u_m)}\right) \\
&\quad + \left(F^i_{[q+1]}\left(\tilde{\mathbf{x}}_{(u_1,u_2,\cdots,u_m)}\right) - F^i_{[q+1]}\left(\tilde{\mathbf{x}}_{(u_1,u_2,\cdots,u_m)}\right)\right)\Big| \\
&\leq \left|F^i_{[q+1]}\left(\mathbf{x}^*_{[q+1]}\right) - F^i_{[q+1]}\left(\tilde{\mathbf{x}}_{(u_1,u_2,\cdots,u_m)}\right)\right| \\
&\quad + \left|F^i_{[q+1]}\left(\tilde{\mathbf{x}}_{(u_1,u_2,\cdots,u_m)}\right) - F^i_{[q+1]}\left(\tilde{\mathbf{x}}_{(u_1,u_2,\cdots,u_m)}\right)\right| \\
&\leq v_q F^i_{[q+1]}\left(\mathbf{x}^*_{[q+1]}\right) + v\left(1 + v_q\right) F^i_{[q+1]}\left(\mathbf{x}^*_{[q+1]}\right) = v_{q+1} F^i_{[q+1]}\left(\mathbf{x}^*_{[q+1]}\right),
\end{aligned}$$

as required. □

Lemma 11.4 is the basis of the main statement of this section.

**Theorem 11.5** *For problem $Pm\left|p_j(\tau) = p_j + a_j\tau\right|C_{\max}$, Algorithm ParMachLin-DeterFPTAS behaves as an FPTAS.*

*Proof* Let $S^*$ be an optimal schedule, associated with an optimal assignment vector $\mathbf{x}^*$, so that

$$C_{\max}\left(S^*\right) = \max\left\{F^i_{[n]}\left(\mathbf{x}^*\right) | 1 \le i \le m\right\}.$$

Also, let $S_\varepsilon$ be a schedule associated with the vector $\mathbf{x}_\varepsilon$ found by Algorithm ParMachLinDeterFPTAS, so that

$$C_{\max}(S_\varepsilon) = \max\left\{F^i_{[n]}(\mathbf{x}_\varepsilon) | 1 \le i \le m\right\}.$$

It follows from Lemma 11.4 that

$$C_{\max}(S_\varepsilon) \le (1 + v_n) C_{\max}\left(S^*\right).$$

It is easy to verify that (11.18) implies that

$$v_n = (1 + v)^n - 1.$$

Recall that

$$e = \sum_{k=0}^{\infty} \frac{1}{k!},$$

so that

$$e - 1 = \sum_{k=1}^{\infty} \frac{1}{k!}.$$

We deduce that

$$\begin{aligned}(1 + v)^n - 1 &= \sum_{k=0}^{n} \binom{n}{k} v^k - 1 = \sum_{k=1}^{n} \frac{n!}{(n-k)!k!n^k} \left(\frac{\varepsilon}{e-1}\right)^k \\ &\le \sum_{k=1}^{n} \frac{1}{k!} \left(\frac{\varepsilon}{e-1}\right)^k \le \left(\frac{\varepsilon}{e-1}\right) \sum_{k=1}^{n} \frac{1}{k!} \le \varepsilon,\end{aligned}$$

so that

$$C_{\max}(S_\varepsilon) \le (1 + \varepsilon) C_{\max}\left(S^*\right),$$

as required.

To complete the proof of the theorem, we need to estimate the running time of Algorithm ParMachLinDeterFPTAS.

Define

$$p_{\max} = \max\{p_j | 1 \le j \le n\};$$
$$a_{\max} = \max\{a_j | 1 \le j \le n\};$$
$$L = \log \max\{n, 1/\varepsilon, p_{\max}, 1 + a_{\max}\}.$$

For each $k$, $1 \le k \le n$, the $k$th iteration of Step 2 of Algorithm ParMachLinDeterFPTAS requires $O(|Y'_k| \log |Y'_k|)$ time. We see from Step 2(a) and Step 2(d) that $|Y'_k| \le m|Y_{k-1}|$, where

$$|Y_{k-1}| \le \prod_{i=1}^{m} r_{F_i}.$$

We deduce from Lemma 11.3 that

$$r_{F_i} \le \left(\log(np_{\max}(1 + a_{\max})^n)\right)/v + 2 \le en(n+2)L/\varepsilon + 2, \ 1 \le i \le m,$$

so that

$$|Y'_k| = O(mn^{2m} L^m / \varepsilon^m),$$

and

$$O(|Y'_k| \log |Y'_k|) = O(m^2 n^{2m} L^{m+1} / \varepsilon^m).$$

Thus, we conclude that the running time of Algorithm ParMachLinDeterFPTAS is $O(m^2 n^{2m+1} L^{m+1} / \varepsilon^m)$, which is polynomial with respect to the length of the problem's input. □

### 11.2.2 Minimizing Total Flow Time: Complexity

In this subsection, we still consider the start-time-dependent effect $p_j(\tau) = a_j \tau$, $j \in N$, and prove that problem $P2 | p_j(\tau) = a_j \tau | \sum C_j$ to minimize the sum of the completion times on two identical parallel machines is NP-hard in the ordinary sense. The following problem is used for reduction.

SUBSET PRODUCT: Given the index set $R = \{1, \ldots, r\}$, positive integers $e_1, \ldots, e_r$, and an integer $V$, does there exist a subset $R' \subseteq R$ such that $\prod_{j \in R'} e_j = V$?

Recall that SUBSET PRODUCT is NP-hard in the ordinary sense; see Sect. 1.3.2 for a discussion. Notice that without loss of generality, we may assume that $e_j \ge 2$, since if $e_j = 1$ then item $j$ can be always included into set $R'$.

Denote

$$U := \prod_{j \in R} e_j; \ W := U/V. \tag{11.23}$$

It can be assumed that $W$ is an integer, since otherwise SUBSET PRODUCT does not have a solution.

We will need the following auxiliary statement.

**Lemma 11.5** *Given an instance of* SUBSET PRODUCT, *for any set $T \subseteq R$ the inequality*

$$W \prod_{j \in T} e_j + V \prod_{j \in R \setminus T} e_j \geq 2U, \tag{11.24}$$

*holds. Moreover, (11.24) holds as equality if and only if $\prod_{j \in T} e_j = V$ (and hence $\prod_{j \in R \setminus T} e_j = W$).*

*Proof* Indeed, denote

$$w := W \prod_{j \in T} e_j, \ v := V \prod_{j \in R \setminus T} e_j,$$

so that $U = \sqrt{wv}$, since $U = VW$ and $U = \prod_{j \in T} e_j \prod_{j \in R \setminus T} e_j$. Then, by the classical inequality on the arithmetic mean and the geometric mean, we have that $w + v \geq 2\sqrt{wv}$, and the equality holds if and only if $w = v$. □

Now, we are ready to prove that SUBSET PRODUCT polynomially reduces to a decision version of problem $P2\left|p_j(\tau) = a_j\tau\right|\sum C_j$.

**Theorem 11.6** *Problem $P2\left|p_j(\tau) = a_j\tau\right|\sum C_j$ is NP-hard in the ordinary sense.*

*Proof* Given an arbitrary instance of SUBSET PRODUCT, define the following instance of the decision version of problem $P2\left|p_j(\tau) = a_j\tau\right|\sum C_j$.

There are $n = r + 4$ jobs, such that

$$\begin{aligned} a_j &= e_j - 1, \ 1 \leq j \leq r, \\ a_{r+1} &= UV - 1, \ a_{r+2} = UW - 1, \\ a_{r+3} &= a_{r+4} = U^3 - 1. \end{aligned}$$

We show that SUBSET PRODUCT has a solution if and only if for the constructed instance of problem $P2\left|p_j(\tau) = a_j\tau\right|\sum C_j$ there exists a schedule $S_0$ for which the value of the objective function is at most $Y = \tau_0\left(2U^5 + U^4\right)$, where $\tau_0$ is an arbitrarily chosen start time of the first job on each machine.

First, assume that SUBSET PRODUCT has a solution, and $R'$ is the found subset of $R$, i.e., $\prod_{j\in R'} e_j = V$ and $\prod_{j\in R\setminus R'} e_j = W$. Then, a required schedule $S_0$ exists and can be found as follows:

- machine $M_1$ processes jobs of set $R'$ in the order of their numbering, followed by the sequence $(r+2, r+3)$ of jobs;
- machine $M_2$ processes jobs of set $R\setminus R'$ in the order of their numbering, followed by the sequence $(r+1, r+4)$ of jobs.

We see that

$$\begin{aligned}
C_{r+1} &= \tau_0\left(\prod_{j\in R\setminus R'}\left(1+a_j\right)\right)(1+a_{r+1}) = \tau_0 W(UV) = \tau_0 U^2;\\
C_{r+2} &= \tau_0\left(\prod_{j\in R'}\left(1+a_j\right)\right)(1+a_{r+2}) = \tau_0 V(UW) = \tau_0 U^2;\\
C_{r+3} &= C_{r+2}(1+a_{r+3}) = \tau_0 U^5;\\
C_{r+4} &= C_{r+1}(1+a_{r+4}) = \tau_0 U^5.
\end{aligned}$$

Notice that $C_j < C_{r+2} = C_{r+1}$ for $j \in R'$ and $C_j < C_{r+1}$ for $j \in R\setminus R'$, so that

$$\sum_{j\in R} C_j < rC_{r+1}.$$

Thus, for schedule $S_0$, we compute

$$\begin{aligned}
\sum_{j=1}^{r+4} C_j(S_0) &= \sum_{j\in R} C_j + C_{r+1} + C_{r+2} + C_{r+3} + C_{r+4}\\
&< (r+2)C_{r+1} + 2C_{r+3} = (r+2)\tau_0 U^2 + 2\tau_0 U^5\\
&< \tau_0\left(2U^5 + U^4\right),
\end{aligned}$$

where the last inequality is due to $U^2 > U \geq 2^r > r+2$.

Now assume that there exists a required schedule $S_0$, in which the sum of the completion times does not exceed $Y$.

If in schedule $S_0$ jobs $r+3$ and $r+4$ are processed on the same machine, then the completion time of the later scheduled job is at least $\tau_0(1+a_{r+3})(1+a_{r+4}) = \tau_0 U^6 > Y$. Thus, jobs $r+3$ and $r+4$ are assigned to different machines.

If in schedule $S_0$ jobs $r+1$ and $r+2$ are processed on the same machine, then either job $r+3$ or job $r+4$ is also assigned to that machine, and the completion time of that of these three jobs that is scheduled last is at least $\tau_0(1+a_{r+1})(1+a_{r+2})(1+a_{r+3}) = \tau_0(UV)(UW)U^3 = \tau_0 U^6 > Y$. Thus, jobs $r+1$ and $r+2$ are assigned to different machines.

Without loss of generality, we may assume that in $S_0$ jobs $r+2$ and $r+3$ are assigned to machine $M_1$, while jobs $r+1$ and $r+4$ are assigned to machine $M_2$. Let $T \subseteq R$ denote the set of jobs that are processed on machine $M_1$.

Compute

$$\begin{aligned}
C^{(1)} &= \tau_0\left(\prod_{j\in T}\left(1+a_j\right)\right)(1+a_{r+2})(1+a_{r+3}) \\
&= \tau_0\left(\prod_{j\in T}\left(1+a_j\right)\right)(UV)U^3 = \tau_0\left(W\prod_{j\in T}\left(1+a_j\right)\right)U^4; \\
C^{(2)} &= \tau_0\left(\prod_{j\in R\setminus T}\left(1+a_j\right)\right)(1+a_{r+1})(1+a_{r+4}) \\
&= \tau_0\left(\prod_{j\in R\setminus T}\left(1+a_j\right)\right)(UW)U^3 = \tau_0\left(V\prod_{j\in R\setminus T}\left(1+a_j\right)\right)U^4.
\end{aligned}$$

We have that

$$\sum_{j=1}^{r+4} C_j(S_0) \geq C^{(1)} + C^{(2)} = \tau_0 U^4\left(W\prod_{j\in T}\left(1+a_j\right) + V\prod_{j\in R\setminus T}\left(1+a_j\right)\right).$$

If Subset Product does not have a solution, i.e., $\prod_{j\in T}\left(1+a_j\right) \neq V$ and $\prod_{j\in R\setminus T}\left(1+a_j\right) \neq W$, we derive from Lemma 11.5 that

$$W\prod_{j\in T}\left(1+a_j\right) + V\prod_{j\in R\setminus T}\left(1+a_j\right) > 2U,$$

so that

$$\sum_{j=1}^{r+4} C_j(S_0) > \tau_0 U^4(2U+1) = Y.$$

Thus, if schedule $S_0$ with $\sum_{j=1}^{r+4} C_j(S_0) \leq Y$ exists, Subset Product must have a solution.

The presented reduction requires time that is polynomial in $r$ and $W$. Since Subset Product is NP-complete in the ordinary sense, it follows that problem $P2\left|p_j(\tau) = a_j\tau\right|\sum C_j$ has the same complexity status. □

## 11.3 Start-Time-Dependent Job-Independent Linear Effects

In is section, we address parallel machine scheduling problems to minimize the sum of the completion times under linear additive start-time-dependent job-independent effects. The jobs of set $N = \{1, 2, \ldots, n\}$ are to be processed on $m$ parallel machines. If a job $j \in N$ scheduled on machine $M_i$, $1 \le i \le m$, is associated with a normal processing time $p_{ij}$, then the actual processing time $p_{ij}(\tau)$ of job $j$ that starts at time $\tau$ is given by

$$p_{ij}(\tau) = p_{ij} \pm a^{[i]}\tau, \tag{11.25}$$

where $a^{[i]} > 0$ is a given machine-dependent rate which is common for all jobs; in the case of $p_{ij} + a^{[i]}\tau$, we have a deterioration effect, while $p_{ij} - a^{[i]}\tau$ defines a learning effect. Notice that if a machine is under a start-time-dependent learning effect, i.e., the negative sign is used in (11.25), we must also adopt the additional assumption $a^{[i]} < 1$, $1 \le i \le m$, which follows from (8.15) and guarantees that the actual processing times do not assume negative values. Other than this additional assumption, the treatment for both deterioration and learning versions of the effect (11.25) is the same and does not depend on the sign used in front of $a^{[i]}\tau$.

### *11.3.1 Identical and Uniform Machines*

First, let us consider problem $Qm\left|p_{ij}(\tau) = p_j/s_i \pm a^{[i]}\tau\right|\sum C_j$ on uniform machines under an additive linear job-independent effect (11.25), so that we have $p_{ij} = p_j/s_i$, $1 \le i \le m$, $j \in N$. As in Sect. 11.1.1, to solve problem $Pm\left|p_{ij}(\tau) = p_j \pm a\tau\right|\sum C_j$ on identical parallel machines under the effect (11.25), we can use the same algorithm as for problem $Qm\left|p_{ij}(\tau) = p_j/s_i \pm a^{[i]}\tau\right|$ $\sum C_j$, applied with $s_i = 1$ and $a^{[i]} = a$, $1 \le i \le m$.

Under the effect (11.25), for a schedule $S$ on uniform machines defined by permutations $\pi^{[i]} = \left(\pi^{[i]}(1), \pi^{[i]}(2), \ldots, \pi^{[i]}\left(n^{[i]}\right)\right)$, $1 \le i \le m$, the total completion time can be easily obtained by substituting $g^{[i]}(r) = 1$, $1 \le r \le n$, $1 \le i \le m$, in (11.4), which is an expression for the total completion time for the more general problem $Qm\left|p_{ij}(r) = \left(p_j \pm a^{[i]}\tau\right)g^{[i]}(r)/s_i\right|\sum C_j$. We deduce that for problem $Qm\left|p_{ij}(\tau) = p_j/s_i \pm a^{[i]}\tau\right|\sum C_j$, the sum of the completion times for all jobs on all machines can be written as

$$\begin{aligned}\sum_{j=1}^{n} C_j(S) &= \sum_{i=1}^{m}\sum_{r=1}^{n^{[i]}} \frac{p_{\pi^{[i]}(r)}}{s_i}\left(\sum_{k=r}^{n^{[i]}}\prod_{q=r+1}^{k}\left(1 \pm a^{[i]}\right)\right)\\ &= \sum_{i=1}^{m}\sum_{r=1}^{n^{[i]}} \frac{p_{\pi^{[i]}(r)}}{s_i}\sum_{k=r}^{n^{[i]}}\left(1 \pm a^{[i]}\right)^{k-r} \qquad (11.26)\\ &= \sum_{i=1}^{m}\sum_{r=1}^{n^{[i]}} \frac{p_{\pi^{[i]}(r)}}{s_i}\left(\pm\frac{\left(1 \pm a^{[i]}\right)^{n^{[i]}-r+1}-1}{a^{[i]}}\right),\end{aligned}$$

which can be rewritten as the generic objective function (11.1) with

$$W^{[i]}(r) = \frac{\left(1 + a^{[i]}\right)^{n^{[i]}-r+1}-1}{s_i a^{[i]}},\ 1 \le r \le n^{[i]},\ 1 \le i \le m, \qquad (11.27)$$

for the case of deterioration and

$$W^{[i]}(r) = \frac{1-\left(1 - a^{[i]}\right)^{n^{[i]}-r+1}}{s_i a^{[i]}},\ 1 \le r \le n^{[i]},\ 1 \le i \le m, \qquad (11.28)$$

for the case of learning. Notice that the positional weights given by (11.27) and (11.28) are job-independent; thus, a solution of the corresponding problem can be found by Algorithm Match due to Lemma 11.1. Below, we present a procedure for solving problem $Qm\left|p_{ij}(\tau) = p_j/s_i + a^{[i]}\tau\right|\sum C_j$ with a deterioration effect. Solution to problem $Qm\left|p_{ij}(\tau) = p_j/s_i - a^{[i]}\tau\right|\sum C_j$ with a learning effect can be found in a similar way.

Denote

$$\chi^{[i]}(r) = \left(1 + a^{[i]}\right)^r - 1,\ 1 \le r \le n,\ 1 \le i \le m,$$

and set the value $n^{[i]} = n$, $1 \le i \le m$. Compute all possible positional weights $W^{[i]}(r)$, $1 \le r \le n$, $1 \le i \le m$, by (11.27). These weights can be organized in an $n \times m$ matrix such that

$$\begin{pmatrix} \chi^{[1]}(n)/\left(s_1a^{[1]}\right) & \chi^{[2]}(n)/\left(s_2a^{[2]}\right) & \cdots & \chi^{[m]}(n)/\left(s_ma^{[m]}\right)\\ \chi^{[1]}(n-1)/\left(s_1a^{[1]}\right) & \chi^{[2]}(n-1)/\left(s_2a^{[2]}\right) & \cdots & \chi^{[m]}(n-1)/\left(s_ma^{[m]}\right)\\ \vdots & \vdots & \vdots & \vdots\\ \chi^{[1]}(2)/\left(s_1a^{[1]}\right) & \chi^{[2]}(2)/\left(s_2a^{[2]}\right) & \cdots & \chi^{[m]}(2)/\chi^{[2]}(n)\\ \chi^{[1]}(1)/\left(s_1a^{[1]}\right) = 1/s_1 & \chi^{[2]}(1)/\left(s_2a^{[2]}\right) = 1/s_2 & \cdots & \chi^{[m]}(1)/\left(s_ma^{[m]}\right) = 1/s_m \end{pmatrix}, \qquad (11.29)$$

where each column of the matrix represents all possible positional weights that can be associated with a particular machine, the first element of column $i$ representing a weight associated with the first position of machine $M_i$, while the last element of column $i$, representing a weight associated with the last, i.e., the $n$th position on machine $M_i$, $1 \le i \le m$. Notice that the elements of each column form a non-increasing sequence of the weights, i.e.,

$$W^{[i]}(1) \ge W^{[i]}(2) \ge \cdots \ge W^{[i]}(n).$$

The above inequality will also hold for problem $Qm\left|p_{ij}(\tau) = p_j/s_i - a^{[i]}\tau\right| \sum C_j$ for a learning effect. Also notice that the elements of matrix (11.29) can be found recursively for each column by using

$$W^{[i]}(r) := \frac{\left(\left(W^{[i]}(r+1)s_v a^{[i]} + 1\right)\left(1 + a^{[i]}\right) - 1\right)}{s_i a^{[i]}}, \ 1 \le r \le n-1, 1 \le i \le m, \tag{11.30}$$

for a deterioration effect and

$$W^{[i]}(r) := \frac{\left(\left(W^{[i]}(r+1)s_v a^{[i]} - 1\right)\left(1 - a^{[i]}\right) + 1\right)}{s_i a^{[i]}}, \ 1 \le r \le n-1, 1 \le i \le m, \tag{11.31}$$

for a learning effect.

The following statement explains how problem $Qm\left|p_{ij}(\tau) = p_j/s_i + a^{[i]}\tau\right| \sum C_j$ can be solved, without prior knowledge of the values $n^{[i]}$, $1 \le i \le m$.

**Theorem 11.7** *Given problem $Qm\left|p_{ij}(\tau) = p_j/s_i + a^{[i]}\tau\right| \sum C_j$, where $1 \le i \le m$, compute the matrix (11.29) of all possible positional weights $W^{[i]}(r)$ and choose the $n$ smallest among them. If in each column the chosen elements occupy consecutive positions starting from the last row, then assigning the jobs with the largest normal processing times to the positions associated with the smallest positional weights will ensure that the objective function (11.1) is minimized.*

*Proof* In a schedule for problem $Qm\left|p_{ij}(\tau) = p_j/s_i + a^{[i]}\tau\right| \sum C_j$, at most $n$ positions can be used on each machine. Matrix (11.29) provides the weights for $nm$ positions, in which $n$ jobs can be potentially scheduled on $m$ machines. Recall that the contribution of a job $j = \pi^{[i]}(r)$ to the objective function is given by $W^{[i]}(r)p_j$. Thus, in order to ensure the smallest value of the objective function, we must choose $n$ positions that correspond to the smallest positional weights. The smallest positional weight for a machine is associated with the last position on that machine, irrespective of the number of the assigned jobs, i.e., for machine $M_i$, in accordance with (11.27) the smallest weight is $W^{[i]}\left(n^{[i]}\right) = 1/s_i$. The next smallest positional weight on machine $M_i$ is located immediately above in the same column, and so on. Thus, the $n$ smallest positional weights are found in the consecutive positions of the columns at the bottom of the matrix (11.29).

For each machine $M_i$, either the found positions form a block of $n^{[i]}$ consecutive positions that completes at the last row of the $i$th column of matrix (11.29) or no positions are taken from the $i$th column. In the former case, we have a list of $n^{[i]}$ positional weights that will be associated with the $n^{[i]}$ jobs assigned to $M_i$. In the latter case, $n^{[i]} = 0$, so that no jobs are assigned to machine $M_i$.

An optimal schedule can be found by matching the $n$ found smallest positional weights to the jobs with the largest processing times, which is performed by Algorithm Match. □

Notice that Theorem 11.7 is only applicable to solving those scheduling problems with changing processing times, for which all possible positional weights can be computed in advance, which is not the case, e.g., for problem $Pm\left|p_{ij}(r) = \left(p_j + a\tau\right) g(r)\right| \sum C_j$ considered in Sect. 11.1.

The problem of finding the $n$ smallest positional weights and matching them to the appropriate jobs is structurally similar to that of problem $Qm||\sum C_j$. The latter problem can be solved by a method described in Sect. 2.3.1.

Adapting this approach to our problem, consider the jobs in the LPT order. To assign the first job, compare the $m$ values $1/s_i$, $1 \le i \le m$ and assign the job to the last position of the machine associated with the smallest value of $1/s_i$, $1 \le i \le m$. The next positional weight that can be taken from this machine is computed and replaces the previously used one. The process continues, and for the current job, the smallest of the $m$ available positional weights determines the machine and the position within the machine, where the job should be assigned. This approach does not require any advance knowledge of the number of jobs $n^{[i]}$ in each machine, or in fact, even an advance knowledge of the full matrix (11.29) since the recursive formulae (11.30) or (11.31) may be used.

A formal description of the algorithm is given below.

**Algorithm NSmallQM**

INPUT: An instance of problem $Qm\left|p_{ij}(\tau) = p_j/s_i + a^{[i]}\tau\right| \sum C_j$

OUTPUT: An optimal schedule $S$ defined by the processing sequences $\pi^{[i]}$, $1 \le i \le m$

**Step 0**. Renumber the jobs in the LPT order.

**Step 1**. For each machine $M_i$, $1 \le i \le m$, define an empty processing sequence $\pi^{[i]} := (\varnothing)$ and the weight $W^{[i]} := 1/s_i$. Create a non-decreasing list $\Omega$ of the values $W^{[i]}$, $1 \le i \le m$.

**Step 2**. For each job $j$ from 1 to $n$ do

**(a)** Take the first element $W^{[v]}$ in list $\Omega$, the smallest available positional weight.

**(b)** Assign job $j$ to machine $M_v$ and place it in front of the current permutation $\pi^{[v]}$, i.e., update $\pi^{[v]} := (j, \pi^{[v]})$ and associate job $j$ with the positional weight $W^{[v]}$. Remove $W^{[v]}$ from the list $\Omega$. Update $W^{[v]} := \left(\left(W^{[v]} s_v a^{[v]} + 1\right)\left(1 + a^{[v]}\right) - 1\right)/\left(s_v a^{[v]}\right)$ in accordance with (11.30) and insert the updated value $W^{[v]}$ into $\Omega$, while maintaining the list $\Omega$ non-decreasing.

**Step 3**. With the found permutations $\pi^{[i]}$, $1 \le i \le m$, compute the optimal value of the objective function $\sum C_j(S)$ by substituting appropriate values in (11.1).

The same algorithm may be used to solve problem $Qm\left|p_{ij}(\tau) = p_j/s_i - a^{[i]}\tau\right| \sum C_j$ by updating the recursive formula in Step 2(b) in accordance with (11.31).

Step 0 of Algorithm NSmallQM requires $O(n \log n)$ time. In Step 1, list $\Omega$ can be created in $O(m \log m)$ time. Each iteration of the loop in Step 2 requires $O(\log m)$

time, since the insertion of the updated weight into a sorted list can be done by binary search. Since $n \geq m$, the following statement holds.

**Theorem 11.8** *Algorithm NSmallQM solves problem* $Qm\left|p_{ij}(\tau) = p_j/s_i \pm a^{[i]}\tau\right|\sum C_j$ *on uniform machines under an additive linear job-independent effect (11.25) in* $O(n \log n)$ *time.*

Let us now consider problem $Pm\left|p_{ij}(\tau) = p_j \pm a\tau\right|\sum C_j$ on identical parallel machines under an additive linear job-independent effect (11.25), with $p_{ij} = p_j$ and $a^{[i]} = a$, $1 \leq i \leq m$. To solve problem $Pm\left|p_{ij}(\tau) = p_j \pm a\tau\right|\sum C_j$, we simply need to run Algorithm NSmallQM with values $s_i = 1$, $a^{[i]} = a$, $1 \leq i \leq m$. Due to this simplification, Steps 1 and 2 can be completed faster, but Step 0 still requires $O(n \log n)$ time. The following statement holds.

**Corollary 11.1** *Algorithm NSmallQM solves problem* $Pm\left|p_{ij}(\tau) = p_j \pm a\tau\right|\sum C_j$*, in which identical machines are under an additive linear job-independent effect (11.25), in* $O(n \log n)$ *time. The optimal schedule has a balanced load on all machines, i.e., the difference between the number of jobs assigned to any two machines is no more than 1.*

*Proof* We show that for problem $Pm\left|p_{ij}(\tau) = p_j \pm a\tau\right|\sum C_j$, Algorithm NSmallQM can be simplified; however, it will still require $O(n \log n)$ time.

Let us recompute the matrix (11.29) with values $s_i = 1$, $a^{[i]} = a$, $1 \leq i \leq m$,

$$\begin{pmatrix} \pm((1 \pm a)^n - 1)/a & \pm((1 \pm a)^n - 1)/a & \cdots & \pm((1 \pm a)^n - 1)/a \\ \pm\big((1 \pm a)^{n-1} - 1\big)/a & \pm\big((1 \pm a)^{n-1} - 1\big)/a & \cdots & \pm\big((1 \pm a)^{n-1} - 1\big)/a \\ \vdots & \vdots & \vdots & \vdots \\ \pm\big((1 \pm a)^2 - 1\big)/a & \pm\big((1 \pm a)^2 - 1\big)/a & \cdots & \pm\big((1 \pm a)^2 - 1\big)/a \\ 1 & 1 & \cdots & 1 \end{pmatrix}.$$

Notice that the positional weights for each machine are identical for a given position $r$, $1 \leq r \leq n$. Obviously, the $n$ smallest weights are found in consecutive positions at the bottom of the matrix. The smallest $m$ positional weights are associated with the last position on each of the $m$ machines. The next smallest $m$ positional weights are associated with the second last positions of each of the $m$ machines, and so on. Assuming that $n = \lambda m + \mu$, where $\lambda$ and $\mu$ are non-negative integers, $\mu \leq m - 1$, the optimal number of jobs in each group can be given by

$$n^{[i]} = \begin{cases} \left\lceil \frac{n}{m} \right\rceil = \lambda + 1, & 1 \leq i \leq \mu \\ \left\lfloor \frac{n}{m} \right\rfloor = \lambda, & \mu + 1 \leq i \leq m. \end{cases}$$

With known values of $n^{[i]}$ and $W^{[i]}(r)$, $1 \le i \le m, 1 \le r \le n^{[i]}$, Steps 1 and 2 of algorithm can be completed in constant time. Step 0 still requires $O(n \log n)$ time; thus, an optimal solution to problem $Pm\left|p_{ij}(\tau) = p_j \pm a\tau\right| \sum C_j$ is found in $O(n \log n)$ time. □

### 11.3.2 Unrelated Machines

Now, we study problem $Rm\left|p_{ij}(\tau) = p_{ij} \pm a^{[i]}\tau\right| \sum C_j$ on unrelated parallel machines under an additive linear job-independent effect (11.25).

Consider a schedule $S$ on unrelated machines defined by permutations $\pi^{[i]} = \left(\pi^{[i]}(1), \pi^{[i]}(2), \dots, \pi^{[i]}\left(n^{[i]}\right)\right)$, $1 \le i \le m$. Under the effect (11.25), for schedule $S$, the total completion time of all jobs can be adopted from (11.26) and be written as

$$\sum_{j=1}^{n} C_j(S) = \sum_{i=1}^{m} \sum_{r=1}^{n^{[i]}} p_{i,\pi^{[i]}(r)} \left( \pm \frac{\left(1 \pm a^{[i]}\right)^{n^{[i]}-r+1} - 1}{a^{[i]}} \right), \tag{11.32}$$

where $p_{(i,\pi^{[i]}(r))}$ is the normal processing time of a job $j = \pi^{[i]}(r)$ scheduled in position $r$ of permutation $\pi^{[i]}$ on machine $M_i$. Notice that the above objective function cannot be expressed as (11.1) because the value of the normal processing time is dependent on the machine it is assigned to. Therefore, Lemma 11.1 cannot be applied.

To minimize the objective $\sum_{j=1}^{n} C_j(S)$, let us define the cost

$$c_{j,(i,r)} = p_{ij} \left( \pm \frac{\left(1 \pm a^{[i]}\right)^{n^{[i]}-r+1} - 1}{a^{[i]}} \right), \tag{11.33}$$

which represents the contribution of a job $j = \pi^{[i]}(r)$, to the objective function. Next, set the value $n^{[i]} = n$, $1 \le i \le m$, and compute all possible costs $c_{j,(i,r)}$, $1 \le r \le n$, $1 \le i \le m$, by (11.33) for every job $j \in N$.

Define a rectangular assignment problem with an $n \times k$ cost matrix $\mathbf{C} = \left(c_{j,(i,r)}\right)$ having $n$ rows, each corresponding to a job $j \in N$, and $k = nm$ columns. Number the columns by a string of the form $(i, r)$, where $i$, $1 \le i \le m$, refers to a machine index, and $r$, $1 \le r \le n$, indicates a position within the machine. More precisely, the value of element $c_{j,(i,r)}$ at the intersection of the $j$th row and the $v$th column of matrix $C$ for $v$, $1 \le v \le k$, such that $v = n(i-1) + r$, where $1 \le i \le m$ and $1 \le r \le n$, is defined by the relation (11.33). Notice that the matrix $\mathbf{C}$ represents a set of all possible values of $c_{j,(i,r)}$ and can be computed in $O\left(n^2 m\right)$ time.

As a result, problem of minimizing the objective function (11.32) reduces to a rectangular assignment problem written out below

$$\begin{array}{lll}\min & \sum_{j=1}^{n}\sum_{i=1}^{m}\sum_{r=1}^{n} c_{j,(i,r)} y_{j,(i,r)} & \\ \text{subject to} & \sum_{i=1}^{m}\sum_{r=1}^{n} y_{j,(i,r)} = 1, & j = 1, \dots, n; \\ & \sum_{j=1}^{n} y_{j,(i,r)} \leq 1, & i = 1, \dots, m,\ r = 1, \dots, n; \\ & y_{j,(i,r)} \in \{0, 1\}, & j = 1, \dots, n, i = 1, \dots, m, \\ & & r = 1, \dots, n. \end{array} \tag{11.34}$$

The algorithm to solve a rectangular assignment problem of the form (11.34) is outlined in Sect. 4.1.1. The running time of Algorithm LAPD is $O(n^3 + kn)$, $k = nm \geq n$, for an $n \times k$ cost matrix. Thus, an optimal solution for problem (11.34) can be found in $O(n^3)$ time due to $n \geq m$.

Suppose that for some $i$, $1 \leq i \leq m$, the solution to the assignment problem (11.34) related to problem $Rm\left|p_{ij}(\tau) = p_{ij} \pm a^{[i]}\tau\right| \sum C_j$ is found. Then, $y_{j,(i,r)} = 1$ implies that job $j$ is assigned to the $r$th position of machine $M_i$. The conditions of (11.34) mean that each job will be assigned to a position and no position will be used more than once. Moreover, the fact that the cost function $c_{j,(i,r)}$ defined by (11.33) forms a monotone non-increasing sequence with respect to $r$, $1 \leq r \leq n$, i.e., $c_{j,(i,1)} \geq c_{j,(i,2)} \geq \cdots \geq c_{j,(i,n)}$, $1 \leq i \leq m$, $j \in N$, guarantees that the found assignment admits a meaningful scheduling interpretation, because for each of the $m$ machines either several consecutive positions starting from the first are filled or the machine is not used at all.

Notice that the sign in front of the rate $a^{[i]}$ in (11.25) has no influence on the solution procedure outlined above, i.e., the method works for both deterioration and learning effects. The following statement holds.

**Theorem 11.9** *Problem $Rm\left|p_{ij}(\tau) = p_{ij} \pm a^{[i]}\tau\right| \sum C_j$, in which unrelated machines are under an additive linear job-independent effect (11.25), can be solved in $O\left(n^3\right)$ time, by reducing the problem to a rectangular assignment problem.*

## 11.4 Bibliographic Notes

We are not aware of any publications that study combined effects on parallel machines. Below, we review several important results, related to pure positional or time-dependent effects on parallel machines.

Mosheiov (2001) studies problem $Pm\left|p_{ij}(r) = p_j r^b\right| \sum C_j$ with a polynomial positional learning effect on identical machines. Mosheiov and Sidney (2003) study a more general problem $Qm\left|p_{ij}(r) = p_{ij} r^b\right| \sum C_j$ with a job-dependent polynomial positional learning effect on uniform machines. Both papers claim to solve the problem of minimizing the total completion time in $O\left(n^{m+3}\right)$ time (as reported in Biskup

2008), whereas according to Theorem 11.3, a more general problem can be solved in $O\left(n^{m+2}\right)$ time. The higher running time obtained by Mosheiov (2001) and Mosheiov and Sidney (2003) is due to the fact that they overestimate the number of LAPs to solve as $O(n^m)$, instead of $O(n^{m-1})$. Rustogi and Strusevich (2012) study problem $Qm\left|p_{ij}(r) = p_j g^{[i]}(r)/s_i\right|\sum C_j$, with a job-independent, possibly non-monotone positional effect on uniform machines and show that the problem can be solved in $O(n^m \log n)$ time by reducing the problem to solving a series of linear assignment problems with a product matrix. Theorem 11.1 presents a generalization of this result. Gara-Ali et al. (2016) study problem $Rm\left|p_{ij}(r) = p_{ij}g_j^{[i]}(r)\right|\sum C_j$ with a job-dependent, possibly non-monotone positional effect on unrelated machines. Similarly to Theorem 11.3, they show that the problem of minimizing the total completion time can be solved in $O\left(n^{m+2}\right)$ time.

Theorem 11.4 on the complexity status of problem $P2\left|p_j(\tau) = a_j\tau\right|C_{\max}$ is due to Kononov (1997). The fact that Product Partition is NP-hard in the strong sense is established in Ng et al. (2010).

Our presentation of the FPTAS for problem $Pm\left|p_j(\tau) = p_j + a_j\tau\right|C_{\max}$ follows the paper Kang and Ng (2007), which in turn is heavily based on the work by Kovalyov and Kubiak (1998) that develops an FPTAS for a single machine problem with a special deterioration effect. In our description, we have fixed multiple technical and presentational flaws contained in Kang and Ng (2007). Notice that problem $Pm\left|p_j(\tau) = p_j + a_j\tau\right|C_{\max}$ belongs to a rare type of combinatorial optimization problems for which a pseudopolynomial-time dynamic programming algorithm is not possible unless $\mathcal{P}=\mathcal{NP}$, but an FPTAS exists. See Kang and Ng (2007) and Kovalyov and Kubiak (2012) for discussion.

The proof of Theorem 11.6 on the complexity status of problem $P2\left|p_j(\tau) = p_j + a_j\tau\right|\sum C_j$ is due to Chen (1996); an alternative proof is given by Kononov (1997). Notice that in the original paper by Chen (1996), it is stated that problem $P2\left|p_j(\tau) = a_j\tau\right|\sum C_j$ is NP-hard in strong sense, since the author assumed the unary NP-hardness of Subset Product, as claimed in Garey and Johnson (1979). However, in fact, both Subset Product and problem $P2\left|p_j(\tau) = a_j\tau\right|\sum C_j$ are NP-hard in the ordinary sense; the corresponding corrections are done in Johnson (1981) and Chen (1997), respectively. See Kang and Ng (2007) and Sect. 1.3.2 for discussion.

The result of Corollary 11.1 follows from Kuo and Yang (2008), who study problem $Pm\left|p_{ij}(\tau) = p_j \pm a\tau\right|\sum C_j$, in which identical machines are under an additive linear job-independent effect (11.25).

## References

Biskup D (2008) A state-of-the-art review on scheduling with learning effects. Eur J Oper Res 188:315–329

Chen Z-L (1996) Parallel machine scheduling with time dependent processing times. Discret Appl Math 70:81–93

Chen Z-L (1997) Erratum: Parallel machine scheduling with time dependent processing times. Discret Appl Math 75:103

Gara-Ali A, Finke G, Espinouse ML (2016) Parallel-machine scheduling with maintenance: praising the assignment problem. Eur J Oper Res 252:90–97

Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. Freeman, San Francisco

Johnson DS (1981) The NP-completeness column: an ongoing guide. J Algorithm 2:393–405

Kang L, Ng CT (2007) A note on a fully polynomial-time approximation scheme for parallel-machine scheduling with deteriorating jobs. Int J Prod Econ 109:180–184

Kononov A (1997) Scheduling problems with linear increasing processing times. In: Zimmermann U et al (eds) Operations Research Proceedings. Springer, Berlin, pp 199–206

Kovalyov MY, Kubiak W (1998) A fully polynomial approximation scheme for minimizing makespan of deteriorating jobs. J Heur 3:287–297

Kovalyov MY, Kubiak W (2012) A generic FPTAS for partition type optimization problems. Int J Plan Sched 1:209–233

Kuo W-H, Yang D-L (2008) Parallel-machine scheduling with time dependent processing times. Theor Comput Sci 393:204–210

Mosheiov G (2001) Parallel machine scheduling with a learning effect. J Oper Res Soc 52:1165–1169

Mosheiov G, Sidney JB (2003) Scheduling with general job-dependent learning curves. Eur J Oper Res 147:665–670

Ng CT, Barketau MS, Cheng TCE, Kovalyov MY (2010) "Product Partition" and related problems of scheduling and systems reliability: computational complexity and approximation. Eur J Oper Res 207:601–604

Rustogi K, Strusevich VA (2012) Simple matching vs linear assignment in scheduling models with positional effects: A critical review. Eur J Oper Res 222:393–407

# Part III
# Scheduling with Rate Modifying Activities

# Chapter 12
# General Framework for Studying Models with Rate-Modifying Activities

The importance of the planning of machine maintenance for production enterprises and service organizations is widely recognized by both practitioners and management scientists; see for example, popular books on maintenance Palmer (2012) and Nyman and Levitt (2010), as well as various Internet emporiums such as www.plant-maintenance.com, www.maintenanceworld.com, www.maintenanceresources.com.

We will distinguish between several types of maintenance and rate-modifying activities that take place in a certain time period. Essentially, during such a period, the machine is not available for processing jobs, and activities done during the period may alter the processing conditions and therefore affect the processing times of jobs scheduled after the period.

In this introductory chapter, we use the single machine environment to introduce all concepts that are needed further in this part of the book.

In Sect. 12.1, we introduce problems with compulsory maintenance, i.e., a maintenance activity should take place in a given interval. Section 12.2 addresses a more flexible situation, where a maintenance activity must start before a given deadline. Maintenance activities which modify processing rates of the jobs that follow that activity are presented in Sect. 12.3. Finally, Sect. 12.4 considers scheduling problems for which the jobs are subject to various effects studied in Part II of this book and rate-modifying activities. For each model, we introduce decisions that should be taken in order to solve the corresponding problems. For the most general model given in Sect. 12.4, we present a generic procedure aimed at solving the relevant problems on a single machine. This procedure and its extension to the parallel machine environment presented in Chap. 20 are widely used throughout the remainder of the book.

V.A. Strusevich and K. Rustogi, *Scheduling with Times-Changing Effects and Rate-Modifying Activities*, International Series in Operations Research & Management Science 243, DOI 10.1007/978-3-319-39574-6_12

## 12.1 Compulsory Maintenance

Suppose that the jobs of set $N = \{1, 2, \ldots, n\}$ are processed on a single machine. Consider the simplest situation that the machine is not available during the time interval $[s, t]$, which is known is advance. This interval can be understood as the period of planned compulsory maintenance, and we will refer to it as the compulsory maintenance period (CMP). The decision-maker may alter neither the start of the CMP nor its duration. Typically, a CMP may correspond to a compulsory break/rest period, or the end of a work period. The processing time of each job is not affected by a position of the job, before or after the CMP, i.e., it remains constant throughout the whole planning horizon.

For a given sequence of jobs, a job $\ell \in N$ that starts before a CMP but cannot be fully completed before it begins is called a *crossover* job. Depending on how a crossover job is handled, we distinguish between two *scenarios*:

- *resumable*: A crossover job is interrupted when a CMP begins and resumes after the CMP from the point of interruption;
- *non-resumable*: A crossover job restarts from scratch after the CMP.

*Example 12.1* To illustrate the model with a single CMP and the two scenarios of handling the crossover job, consider a lecturer who cannot finish proving a theorem by the end of the lecturing hour. If she has another hour of lecturing after a short five or ten minute break, she may proceed with the proof from where it was stopped before the break, expecting the students to remember the details of the proved part. However, if that was the last lecturing hour for today, and the next lecture takes place only in a week's time, then it will be wise to start the proof from the very beginning, since the gap is too big for most of the students to be fully engaged. This example represents the resumable and the non-resumable scenarios.

It is not within the scope of this book to give a comprehensive exposition of scheduling problems with CMPs. The main reason is that in these models the decision-maker simply faces the CMPs as intervals of machine non-availability and they do not affect the processing times of jobs. This does not correspond to the spirit of this book, which focuses on changing processing times and those maintenance periods that may alter processing conditions. The reader is advised to consult the survey papers Lee (1996, 2004) and Ma et al. (2010) for a comprehensive exposition of scheduling problems with CMPs.

In this book, we only discuss the single machine problems with a single CMP to minimize the makespan $C_{\max}$, the total completion time $\sum C_j$, and its weighted counterpart $\sum w_j C_j$; see Chap. 13. In order to find a schedule that minimizes a certain function that depends on the completion times, the decision-maker should take two decisions:

**CMP Decision 1**. *Splitting in groups*: To split the jobs into groups, one before the first CMP and others after each CMP.

**CMP Decision 2**. *Sequencing within groups*: To sequence the jobs of each group so that the resulting schedule delivers an optimal value of the chosen objective function.

The reason why in Chap. 13 we mainly discuss the models with a single CMP can be explained by the fact that the relevant problems with more than one CMPs are not only hard to solve, but also hard to approximate. A noticeable exception is the situation that the CMPs should be introduced into a schedule on a periodic basis, so that exactly $T$ time units should elapse between two consecutive CMPs. A model with periodic compulsory maintenance is also among those studied in Chap. 13.

## 12.2 Flexible Maintenance

An extension of the CMP model described above gives more freedom to the decision-maker. In reality, a maintenance period (MP) does not always start exactly at a prescribed time. In most practical situations, it is required that an MP either starts before a given deadline $D_{MP}$ or should take place within a given window $[s, t]$. Such situations occur when manufacturers insist that the users should perform equipment maintenance after so many working hours/days, e.g., car servicing. Alternative examples include mobile phone recharging or vehicle refilling.

Compared to the CMP model, the above model with a single *flexible* MP provides more control to the decision-maker who now has to make the following decision:

**Start MP Decision**. *When an MP starts:* To assign a start time $\tau$ for an MP.

Once the Start MP Decision is taken, the position of the MP becomes fixed, and to solve the corresponding scheduling problem, we need to take CMP Decisions 1 and 2 above.

A certain flexibility regarding the start time of an MP allows us to introduce a more enhanced and again more applicable model, which also reduces to the sequence of decisions: Start MP Decision, CMP Decision 1, and CMP Decision 2. This enhanced model assumes that the duration of an MP is not a constant but a non-decreasing function $\Delta(\tau)$ of its start time $\tau$. Typically, the later maintenance starts, the worse are the equipment conditions and more work is needed to improve them. The models with start-time-dependent maintenance have been introduced by Kubzin and Strusevich (2005, 2006) for the two-machine shop environments.

We discuss the single machine problems with a single flexible MP to minimize the makespan $C_{\max}$, the total completion time $\sum C_j$, and its weighted counterpart $\sum w_j C_j$ in Chap. 14. We look at the models with a start-time deadline, the MP window, and also at the models in which an MP is repeated in an “almost” periodic basis, with a regular pattern for the windows in which the next MP should be placed.

## 12.3 Rate-Modifying Activities

A common drawback of the models with machine maintenance discussed earlier in this chapter is that they fail to address the issue of changing processing conditions, i.e., do not alter the processing times of the jobs scheduled before and after an MP. Thus, the introduced periods are maintenance periods only by name, not by nature. This calls for a study of enhanced models which treat a maintenance period as a *rate-modifying* activity.

One of the first papers that study an effect of maintenance on processing conditions is that by Lee and Leon (2001), who in fact have introduced the term "rate-modifying activity" into the scheduling literature. They look at the problem of scheduling a single MP and assume that the processing time of a job $j$ that is sequenced before the MP is $p_j$, while if it is sequenced after the MP, the processing time becomes $\lambda_j p_j$, where $\lambda_j > 0$ is a rate-modifying multiplier for job $j$. Notice that if an RMP is a maintenance period that improves processing conditions, then, following Lee and Leon (2001), we may assume that $\lambda_j < 1$; however, $\lambda_j > 1$ is possible, e.g., when an MP alters the machine to prevent a breakdown, but the alterations make it work slower.

In Chap. 15, we consider single machine scheduling problems with a single rate-modifying maintenance period (RMP). In the case of a single RMP, the decisions to be taken reduce to Start MP Decision, CMP Decision 1, and CMP Decision 2, as above.

The model with a single RMP can be extended to a general situation, in which the decision-maker is presented with a list ($\text{RMP}^{[1]}, \text{RMP}^{[2]}, \ldots, \text{RMP}^{[K]}$) of $K \geq 1$ possible rate-modifying activities, which can be either distinct or alike. The decision-maker may decide which of the listed RMPs to insert into a schedule, on which machine (in the case of parallel machines) and in which order. Notice that an order in which the selected RMPs are introduced on a machine can be different from the order in which the RMPs appear in the list.

Each RMP may have a different effect on the machine conditions, i.e., different RMPs may have different sets of the rate-modifying multiplies. Such a situation is natural if the selected RMPs are different in nature, e.g., one RMP replaces the cutting tool of the machine, whereas the other refills gas in the system. However, a similar situation can also arise if the RMPs are identical, but their efficiency in performing the desired task changes depending on their position in the schedule. For instance, consider a scenario in which the RMP is aimed at improving the machine conditions by running maintenance. In real life, it is often observed that even after a maintenance activity, some wear and tear might still remain in the machine, and if the same RMPs are performed every time, this deviation might get accumulated.

Consider a scheduling problem of minimizing an objective function $\Phi$, provided that several RMPs can be chosen from a given list of $K$ RMPs and to be included into a schedule. The decision-maker must make the following decisions:

**RMP Decision 1**. *The number of RMPs*: Decide how many of the $K$ available RMPs are to be included in the schedule.

**RMP Decision 2**. *The choice of RMPs*: For the chosen number of the RMPs, decide which particular available RMPs are to be included in the schedule.

**RMP Decision 3**. *The sequence of RMPs*: Determine the optimal order in which the selected RMPs are scheduled on a machine.

Suppose that RMP Decisions 1–3 have been taken. If out of the $K$ available RMPs $k-1$ are selected and included in a schedule on some machine, then the jobs on that machine will be divided into $k$, $1 \leq k \leq K+1$ groups, one to be scheduled before the first RMP and one after each of the $k-1$ RMPs. Since the RMPs are known to affect the machine conditions differently, it follows that the jobs contained in different groups are treated differently. In general, we allow groups to be empty, i.e., two or more RMPs can be scheduled as a block, one immediately after another.

We present further explanations assuming that the jobs of set $N$ are to be processed on a single machine; it is not hard to extend the notions introduced above to parallel machines, see Chap. 20. If $k-1$ RMPs are chosen to be introduced into a schedule, then renumber those RMPs in the order they appear in the schedule. The set of jobs is then split into $k$ groups, $N^{[x]}$, $1 \leq x \leq k$. The set of jobs $N^{[x]}$ forms group $x$, i.e., the jobs of this group are scheduled either before the first RMP (if $x = 1$) or between the $(x-1)$th RMP and the $x$th RMP (if $1 < x \leq k$).

In general, the duration of each of the chosen RMPs is start-time-dependent. In the most studied linear model, the duration of RMP$^{[y]}$ is associated with its own duration parameters, $\zeta^{[y]}$ and $\eta^{[y]}$, $1 \leq y \leq K$, so that its duration $\Delta^{[y]}(\tau)$ is given by

$$\Delta^{[y]}(\tau) = \zeta^{[y]}\tau + \eta^{[y]}, \tag{12.1}$$

where $\tau$ is the start time of the RMP, measured either from time zero in the case of the first RMP or from the completion time of the previous RMP. The value of $\eta^{[y]}$ can be understood as the duration of all tests that form the mandatory part of a maintenance procedure, while $\zeta^{[y]}\tau$ reflects any additional maintenance work that depends on how long the machine has worked without maintenance. If RMP$^{[y]}$ is inserted into a schedule as the $x$th RMP, then $\tau$ is the total actual processing time of the jobs of group $N^{[x]}$, $1 \leq x \leq k-1$.

A special case of (12.1) with $\zeta^{[y]} = 0$, $1 \leq y \leq K$ corresponds to the situation that the duration of RMP$^{[y]}$ is given by a constant $\eta^{[y]}$.

A more general interpretation of the variable duration of the RMP has been introduced by Finke et al. (2016). For a schedule with $k-1$ selected RMPs, the duration $\bar{\Delta}^{[x]}$ of the $x$th RMP, $1 \leq x \leq k-1$, is determined as a linear function of the actual durations of the jobs in the preceding group. Assuming that the actual processing time of job $j \in N^{[x]}$ is equal to $p_j^{[x]}$, this job contributes $\zeta_j^{[x]} p_j^{[x]}$ toward $\bar{\Delta}^{[x]}$ of the $x$th RMP, so that

$$\bar{\Delta}^{[x]} = \sum_{j \in N^{[x]}} \zeta_j^{[x]} p_j^{[x]} + \eta^{[x]}, \tag{12.2}$$

where $\zeta_j^{[x]}$ is a positive job-dependent coefficient. Notice that if $\zeta_j^{[x]}$ is job-independent, i.e., $\zeta_j^{[x]} = \zeta^{[x]}$ for all jobs in group $x$, then $\bar{\Delta}^{[x]}$ satisfies (12.1).

Thus, as a result of taking RMP Decisions 1–3, we determine $k - 1 \leq K$ RMPs, their type and the order in which they are inserted into a schedule. Then, we need to split the jobs into $k$ groups, one before the first RMP and others after each RMP. To find the resulting schedule, we need to find a sequence of jobs in each group. These actions are essentially equivalent to the CMP Decisions 1 and 2. The latter decisions will determine the start times of each RMP in accordance with a chosen sequence. In more detail, the main principles of solving relevant problems are formulated in Sect. 12.4, for more general settings.

The decision-making in the presence of multiple RMPs is illustrated in Chaps. 15 and 20, based on scheduling problems on a single machines and on parallel machines, respectively.

## 12.4 Changing Processing Times and Rate-Modifying Activities

Scheduling problems with various time-changing effects that define the actual processing times of the jobs have been considered in detail in Part II of this book. It must be admitted that the practical impact of research on scheduling models with a learning/deterioration effect alone is somewhat questionable. In practical situations, it is often observed that the machines/operators are subject to (periodic) maintenance activities or replacements. These activities modify the rate of change of processing times, so that the learning/deterioration process is disrupted. Indeed, for a large number of jobs, if the processing conditions are not altered by some sort of a rate-modifying activity, the processing times will either reduce to zero in the case of a learning effect or will grow to unacceptably large values in the case of deterioration. Such situations are not realistic.

This implies that to address practically relevant situations, we need integrated scheduling models that address (i) possible changes in actual processing times of the jobs due to learning/deterioration or an alternative non-monotone effect, and (ii) introduction of rate-modifying activities that alter the processing conditions and therefore influence the processing times of the jobs that follow such an RMP. In the most natural models of this type, the jobs are subject to deterioration and the RMPs are understood as maintenance periods that bring the processing machines to better conditions.

The following quotation from the influential paper by Gopalakrishnan et al. (1997) is especially close to the spirit of this book:

> Industrial systems used in the production of goods are subject to deterioration and wear with usage and age. System deterioration results in increased breakdowns leading to higher production costs and lower product quality. A well-implemented, planned preventive maintenance (PM) program can reduce costly breakdowns... Deciding what PM tasks to do, and when, constitutes a critical resource allocation and scheduling problem.

As seen from the quotation above, in the planning of rate-modifying activities in a processing sequence, the decision-maker is faced with a trade-off between two processes: (i) change of the processing conditions and (ii) allocation of a rate-modifying period in order to control the changing conditions. However, until very recently, the processes (i) and (ii) have not been fully integrated in the models studied in the scheduling literature. One of the purposes of Part III of this book is to provide a uniform formal treatment of such integrated scheduling models.

A systematic study of integrated models that include both various rules of changing the processing times and the introduction of the RMPs of different nature has been conducted by Rustogi and Strusevich (2012a, b, 2014, 2015). These papers make the basis of Chaps. 16–20. The considered models address most of known time-changing effects: positional, cumulative, start-time-dependent, and their combinations. Rate-modifying activities of different nature are considered.

If an RMP is an actual maintenance, there is no need to assume that such an RMP fully restores the machine to its default state every time it is performed. Besides, RMPs to be inserted into a schedule should not be limited to maintenance periods only. It is possible that introducing an RMP in fact slows down the processing, which happens, e.g., if the RMP is related to the replacement of an experienced operator by a trainee. Under another possible effect, an RMP is used to further enhance the learning capabilities of the machines/operators. In any case, as a result of introduction of several RMPs, the jobs are split into the corresponding groups, and the actual processing time of a job will be dependent on the group it is scheduled in. We refer to such an effect as a *group-dependent* effect. To the best of our knowledge, the group-dependent effects in conjunction with rate-modifying and maintenance activities have been first introduced in Rustogi and Strusevich (2012a).

Below, we give an illustration of a possible application of group-dependent effects.

*Example 12.2* A human operator uses a tool to process $n$ jobs. During the processing of the jobs, the tool undergoes deterioration, whereas the performance of the operator is influenced by both deterioration and learning effects. It is known that two RMPs will be included in the schedule. The first RMP is a maintenance period which restores the machine to its original condition. However, the deterioration rate of the machine becomes greater after the maintenance period, since original spare parts are not used. This RMP also provides the operator with sufficient rest, so that after this first RMP, the operator is as fresh as he/she was at the beginning of the schedule. Additionally, the operator gets a technical briefing from his supervisor, so that his learning curve changes. The second RMP does not repair the machine at all; instead, a new operator is brought in. Below, we give details how these effects are modeled mathematically and compute the resulting positional factors $g^{[x]}(r)$. The actual processing time of job $j$ scheduled in position $r$ of group $x$ is defined equal to $g^{[x]}(r)p_j$.

We distinguish between the positional factors associated with the machine and the operator by using the subscript "$m$" for the machine and "$w$" for the operator (worker), respectively. The actual positional factor is defined as the product $g^{[x]}(r) := g_m^{[x]}(r)g_w^{[x]}(r)$.

In a feasible schedule, the jobs will be split into $k = 3$ groups; let $n^{[x]}$ denote the number of jobs in the $x$th group. For the machine, a positional exponential deterioration effect with a rate $A_1 > 0$ is applied before the first RMP and a similar effect with a rate $A_2$ after the first RMP, where $A_2 > A_1$. As a result, the positional factors associated with the machine for the three groups are given as $g_m^{[1]}(r) = (A_1)^{r-1}$, $g_m^{[2]}(r) = (A_2)^{r-1}$, and $g_m^{[3]}(r) = (A_2)^{n^{[2]}+r-1}$, respectively. The two operators are subject to a positional polynomial deterioration effect with rates $B_1 > 0$ and $B_2 > 0$, respectively. They are also subject to positional polynomial learning effects. The rate with which Operator 1 learns before the first RMP is $C_1 < 0$, while the rate with which she learns after the RMP is $C_2 < C_1 < 0$. The learning rate of Operator 2 is given by $C_3 < 0$. As a result, the positional factors associated with the operators for the three groups are given as $g_w^{[1]}(r) = r^{B_1+C_1}$, $g_w^{[2]}(r) = r^{B_1}\left(n^{[1]} + r\right)^{C_2}$, and $g_w^{[3]}(r) = r^{B_2+C_3}$, respectively. Thus, the positional factors for the entire processing system can be given as

$$g^{[1]}(r) = g_m^{[1]}(r)g_w^{[1]}(r) = (A_1)^{r-1}r^{B_1+C_1};$$
$$g^{[2]}(r) = g_m^{[2]}(r)g_w^{[2]}(r) = (A_2)^{r-1}r^{B_1}\left(n^{[1]} + r\right)^{C_2};$$
$$g^{[3]}(r) = g_m^{[3]}(r)g_w^{[3]}(r) = (A_2)^{n^{[2]}+r-1}r^{B_2+C_3}.$$

Notice that this model allows us to assume that during an RMP, if Operator 1 is not replaced, he/she does not lose his/her skills which have been improved due to learning in the earlier groups of the schedule. Similarly, if during an RMP a machine is not fully repaired, our model is capable of handling the resulting situation in which the deterioration effect from the group before the RMP must be carried forward to the next group. These issues are captured by adjusting the relative position of a job in the relevant group. For example, the learning factor involved in the computation of $g^{[2]}(r)$ is given by $\left(n^{[1]} + r\right)^{C_2}$ (implying that Operator 1 has completed $n^{[1]}$ jobs before group 2 starts), and the deterioration factor involved in the computation of $g^{[3]}(r)$ is given by $(A_2)^{n^{[2]}+r-1}$ (implying that since its last RMP, the machine has completed $n^{[2]}$ jobs before group 3 starts).

Below, we discuss some general principles of solving single machine scheduling problems in which the jobs are subject to certain effects (positional, start-time related or combined), and the RMPs may be inserted into a schedule. The jobs of set $N = \{1, 2, \dots, n\}$ are to be processed on a single machine. Each job $j \in N$ is associated with a normal processing time $p_j$. The decision-maker is presented with a list (RMP$^{[1]}$, RMP$^{[2]}$, ..., RMP$^{[K]}$) of $K \geq 1$ possible rate-modifying activities. Let $1|\beta, RMP(K)|\Phi$ denote a generic problem of minimizing an objective function $\Phi(S)$, where the string $\beta$ denotes additional conditions, such as a time-changing effect and/or a rule for computing the duration of the RMPs chosen to be inserted into a schedule.

The original problem $1|\beta, RMP(K)|\Phi$ reduces to a sequence of the auxiliary problems, which we denote by $1|\beta, RMP(k-1)|\Phi$. An instance of each problem $1|\beta, RMP(k-1)|\Phi$ is defined by

**(A1)** an outcome of the RMP Decisions 1–3;
**(A2)** a numbering of the chosen $k-1$ RMPs by integers $x$, $1 \le x \le k-1$, in the order of their appearance in a schedule.

To solve problem $1|\beta, RMP(k-1)|\Phi$, we need to take three decisions, which can be seen as extensions of the CMP Decisions 1 and 2 from Sect. 12.1:

**(B1)** to determine the number $n^{[x]}$ of jobs in group $N^{[x]}$, $1 \le x \le k$, where the jobs of group $N^{[x]}$ are sequenced before the $x$th RMP, and the jobs of group $N^{[k]}$ are scheduled after the last RMP;
**(B2)** determine a partition of the jobs of set $N$ into $k$ groups $N^{[x]}$, $1 \le x \le k$;
**(B3)** find a permutation $\pi^{[x]}$ for the jobs of group $N^{[x]}$.

At the moment, we focus on the most general situation, in which there is no obvious way of taking Decision (B1) above, i.e., there is no method that would allow us to determine the number of jobs in each group in a schedule that is optimal for problem $1|\beta, RMP(k-1)|\Phi$. In this situation, we will need to generate all possible values $n^{[x]}$, $1 \le x \le k$.

For a particular outcome of Decision (B1), introduce a schedule $S_{\mathrm{B1}}(k)$ for an auxiliary problem $1|\beta, RMP(k-1)|\Phi$ associated with certain outcomes of Decisions (B2) and (B3). In schedule $S_{\mathrm{B1}}(k)$, the jobs are organized in groups $N^{[x]}$, $1 \le x \le k$; each group $N^{[x]}$ contains $n^{[x]}$ jobs, where $\sum_{x=1}^{k} n^{[x]} = n$. Further, let the jobs in $N^{[x]}$ be sequenced in accordance with a permutation $\pi^{[x]} = \left(\pi^{[x]}(1), \pi^{[x]}(2), \dots, \pi^{[x]}\left(n^{[x]}\right)\right)$, $1 \le x \le k$. The actual processing time of a job $j = \pi^{[x]}(r)$, scheduled in position $r$, $1 \le r \le n^{[x]}$, of the $x$th group, $1 \le x \le k$, is denoted by $p_j^{[x]}(r)$ and depends on particular features of the model, normally captured by the string $\beta$.

Associate schedule $S_{\mathrm{B1}}(k)$ with an overall permutation of jobs $\pi = (\pi^{[1]}, \pi^{[2]}, \dots, \pi^{[k]})$ of set $N$. In most problems considered in Chaps. 15–18, the objective function for schedule $S_{\mathrm{B1}}(k)$ admits a generic representation

$$\Phi(S_{\mathrm{B1}}(k)) = \Phi(\pi) = \sum_{x=1}^{k} \sum_{r=1}^{n^{[x]}} W_{\pi^{[x]}(r)}^{[x]}(r) p_{\pi^{[x]}(r)} + \Gamma(k), \tag{12.3}$$

where $\Gamma(k)$ depends only on $k$ and some constant terms, while $W_{\pi^{[x]}(r)}^{[x]}(r)$ is a positional weight that in general is both job-dependent and group-dependent. The product $W_{\pi^{[x]}(r)}^{[x]}(r) p_{\pi^{[x]}(r)}$ represents the contribution of job $j = \pi^{[x]}(r)$ scheduled in position $r$, $1 \le r \le n^{[x]}$, of group $x$, $1 \le x \le k$, to the objective function (12.3).

For an outcome of Decision (B1), let $S_{\mathrm{B1}}^{*}(k)$ be the best schedule that is defined by taking Decisions (B2) and (B3) in such a way that schedule $S_{\mathrm{B1}}^{*}(k)$ minimizes function (12.3), i.e., $\Phi\left(S_{\mathrm{B1}}^{*}(k)\right) \le \Phi(S_{\mathrm{B1}}(k))$ holds for all possible schedules $S_{\mathrm{B1}}(k)$. Further, let $S^{*}(k)$ denote a schedule that is optimal for problem $1|\beta, RMP(k-1)|\Phi$, i.e., $\Phi(S^{*}(k)) \le \Phi\left(S_{\mathrm{B1}}^{*}(k)\right)$ holds for all possible outcomes of Decision (B1).

Assuming that for a particular outcome of Decision (B1), each weight $W_j^{[x]}(r)$, $j \in N$, $1 \le x \le k$, $1 \le r \le n^{[x]}$, can be computed in advance in constant time, finding schedule $S^*_{\mathrm{B1}}(k)$ can be reduced to solving a linear assignment problem (LAP); see Sect. 4.1 for definitions and a review.

Suppose first that the weights $W_j^{[x]}(r)$ are job-dependent. In order to minimize $\Phi(S_{\mathrm{B1}}(k))$, define a LAP with an $n \times n$ cost matrix $\mathbf{C} = \left(c_{j,(x,r)}\right)$. Each row of matrix $\mathbf{C}$ corresponds to a job $j \in N$. It is convenient to label the columns of $\mathbf{C}$ by strings of the form $(x, r)$, where $x$, $1 \le x \le k$, refers to a group index, and $r$ indicates a position in a permutation of jobs assigned to a group. Since $\sum_{x=1}^{k} n^{[x]} = n$, there are exactly $n$ columns in matrix $\mathbf{C}$. Notice that the first $n^{[1]}$ columns $(1, 1), (1, 2), \cdots, (1, n^{[1]})$ of matrix $\mathbf{C}$ are associated with the positions in group 1, the next $n^{[2]}$ columns $(2, 1), (2, 2), \cdots, (2, n^{[2]})$ are associated with the positions in group 2, and so on. It follows that the elements of the cost matrix of the LAP can be written as

$$c_{j,(x,r)} = W_j^{[x]}(r)p_j,\ 1 \le x \le k, 1 \le r \le n^{[x]}. \tag{12.4}$$

With the cost values defined by (12.4), the problem of minimizing function (12.3) reduces to the following LAP

$$\begin{aligned}
\text{minimize } & \sum_{j=1}^{n}\sum_{x=1}^{k}\sum_{r=1}^{n^{[x]}} c_{j,(x,r)} z_{j,(x,r)} \\
\text{subject to } & \sum_{x=1}^{k}\sum_{r=1}^{n^{[x]}} z_{j,(x,r)} = 1, \qquad 1 \le j \le n; \\
& \sum_{j=1}^{n} z_{j,(x,r)} = 1, \qquad 1 \le x \le k,\ 1 \le r \le n^{[x]}; \\
& z_{j,(x,r)} \in \{0, 1\},\ 1 \le j \le n, 1 \le x \le k,\ 1 \le r \le n^{[x]}.
\end{aligned} \tag{12.5}$$

A problem of the form (12.5) can be solved in $O(n^3)$ time, as follows from Theorem 4.1. For the found solution, $z_{j,(x,r)} = 1$ implies that in schedule $S^*_{\mathrm{B1}}(k)$, job $j$ is assigned to the $r$th position of group $x$. The conditions of (12.5) mean that each job will be assigned to a position and no position will be used more than once.

In several scheduling problems, the positional weights appear to be job independent, i.e., for each $j \in N$ the equalities $W_j^{[x]}(r) = W^{[x]}(r)$ hold for $1 \le x \le k$, $1 \le r \le n^{[x]}$. In this case, the matrix with the elements (12.4) becomes a product matrix and the corresponding LAP can be solved in $O(n \log n)$ time by adapting Algorithm Match; see Sect. 4.1.3.

In any case, we can write out a generic procedure for finding a schedule $S^*$ that is optimal for problem $1|\beta, RMP(K)|\Phi$.

**Procedure RMP1**

**Step 1**. Given problem $1|\beta, RMP(K)|\Phi$, for each outcome (A1) and (A2) do

**(a)** Define an auxiliary problem $1|\beta, RMP(k-1)|\Phi$.

**(b)** For each outcome of Decision (B1) do

**(i)** Compute appropriate positional weights $W_j^{[x]}(r), j \in N, 1 \le x \le k, 1 \le r \le n^{[x]}$, and the constant $\Gamma(k)$ that define the objective (12.3).

**(ii)** Create the cost matrix $\mathbf{C} = \left(c_{j,(x,r)}\right)$ with the elements (12.4).

**(iii)** Find schedule $S^*_{\mathrm{B1}}(k)$ by solving the linear assignment problem, either in the full form (12.5) or with a product matrix, if appropriate.

**(c)** Determine schedule $S^*(k)$ that is optimal for the current problem $1|\beta, RMP(k-1)|\Phi$, i.e., such that $\Phi(S^*(k)) \le \Phi\left(S^*_{\mathrm{B1}}(k)\right)$ holds for all schedules $S^*_{\mathrm{B1}}(k)$ found in Step 1(b).

**Step 2**. Determine schedule $S^*$ that is optimal for the original problem $1|\beta, RMP(K)|\Phi$, i.e., such that $\Phi(S^*) \le \Phi(S^*(k))$ holds for all schedules $S^*(k)$ found in Step 1.

Notice that Procedure RMP1 is presented under the assumption that all outcomes (A1) and (A2) as well as all outcomes of Decision (B1) should be generated. There are, however, situations when this full enumeration is not needed, and faster approaches can be employed. Without going into technicalities, this happens when all possible positional weights $W_j^{[x]}(r)$, $1 \le r \le n$, $1 \le x \le k$, can be found without any prior knowledge of the values $n^{[x]}$, and the computed positional weights form a monotone sequence for each group.

The lemma below gives an estimation of the number of possible outcomes (A1) and (A2) in Step 1 of Procedure RMP1, as well as the number of all possible Decisions (B1) to be taken for a particular outcome (A1) and (A2). These estimations are used in the remaining chapters of this part for a purpose of evaluating the running times of algorithms based on Procedure RMP1. To count the number of the related instances, we use various combinatorial configurations and identities listed in Sect. 5.3. In the presented estimations, we assume that the number $K$ of available RMPs is a constant, which is reasonable since usually the number of possible rate-modifying activities to be performed is fairly small.

**Lemma 12.1** *The number of outcomes (A1) and (A2) in Step 1 of Procedure RMP1 can be estimated as $O\left(K^K\right)$. For a particular combination of outcomes (A1) and (A2), the number of possible outcomes of Decision (B1) that may lead to an overall optimal schedule $S^*$ is $O\left(n^{k-1}\right)$. Moreover, the total number of all possible outcomes (A1) and (A2) and outcomes of Decision (B1), i.e., the number of the assignment problems to be solved, is $O\left(n^K\right)$.*

*Proof* For a particular $k$, $1 \le k \le K+1$, the number of ways of selecting $k-1$ RMPs from $K$ available RMPs and order them (RMP Decision 2 and 3) is equal to the corresponding number of arrangements $\binom{K}{k-1}(k-1)!$, which according to (5.9) does not exceed $K^{k-1}$. Trying all possible values of $k$, $1 \le k \le K+1$ (i.e., trying all possible options of RMP Decision 1), an upper bound on the number of auxiliary problems $1|\beta, RMP(k-1)|\Phi$ to be solved is $\sum_{k=1}^{K+1} K^{k-1}$.

The number of all possible outcomes of Decision (B1) that need to be generated for each auxiliary problem $1|\beta, RMP(k-1)|\Phi$ is equal to the number of all non negative integers $n^{[x]}$, $1 \le x \le k$, such that $\sum_{x=1}^{k} n^{[x]} = n$. Thus, the number of all values $n^{[x]}$, $1 \le x \le k$, that may lead to an overall optimal schedule $S^*$ is equal to the number of compositions $C_n^{(\le k)}$ of $n$ in at most $k$ positive summands. Applying (5.12) with $u = n$ and $v = k$, we have that the total number of all values $n^{[x]}$, $1 \le x \le k$, to be generated for each auxiliary problem $1|\beta, RMP(k-1)|\Phi$ is

$$\binom{n+k-1}{k-1} \le \frac{n^{k-1}}{(k-1)!} = O\big((n+k)^{k-1}\big).$$

Since we may assume that there are less RMPs than jobs, i.e., $k \le n$, we further deduce $O\big((n+k)^{k-1}\big) = O\big(n^{k-1}\big)$.

Finally, the total number of the linear assignment problems to be solved is given by

$$\sum_{k=1}^{K+1} \binom{K}{k-1} (k-1)! \binom{n+k-1}{k-1} \le \sum_{k=1}^{K+1} \binom{K}{k-1} (n+k)^{k-1}$$
$$\le \sum_{k=1}^{K+1} \binom{K}{k-1} (2n)^{k-1} = (2n+1)^K = O\big(n^K\big),$$

as required. □

In Chaps. 15–18, the described Procedure RMP1 forms the basis of the design of the solution algorithms for single machine scheduling problems in which various (group-dependent) effects are combined with the introduction of available RMPs. An extension of Procedure RMP1 to parallel machines and examples of its adaptation to relevant scheduling problems is contained in Chap. 20.

## References

Finke G, Gara-Ali A, Espinouse ML, Jost V, Moncel J (2016) Unified matrix approach to solve production-maintenance problems on a single machine. Omega. doi:10.1016/j.omega.2016.02.005

Gopalakrishnan M, Ahire SL, Miller DM (1997) Maximizing the effectiveness of a preventive maintenance system: an adaptive modeling approach. Manag Sci 43:827–840

Kubzin MA, Strusevich VA (2005) Two-machine flow shop no-wait scheduling with machine maintenance. 4OR 3:303–313

Kubzin MA, Strusevich VA (2006) Planning machine maintenance in two-machine shop scheduling. Oper Res 54:789–800

Lee C-Y (1996) Machine scheduling with an availability constraint. J Glob Optim 9:395–416

Lee C-Y (2004) Machine scheduling with availability constraints. In Leung JY-T(ed) Handbook of scheduling: algorithms, models and performance analysis. Chapman & Hall/CRC, London, pp 22-1–22-13

Lee C-Y, Leon VJ (2001) Machine scheduling with a rate-modifying activity. Eur J Oper Res 128:119–128

Ma Y, Chu C, Zuo C (2010) A survey of scheduling with deterministic machine availability constraints. Comput Industr Eng 58:199–211

Nyman D, Levitt J (2010) Maintenance planning, scheduling and coordination, 2nd edn. Industrial Press, New York

Palmer D (2012) Maintenance planning and scheduling handbook, 3rd edn. McGraw Hill, New York

Rustogi K, Strusevich VA (2012a) Single machine scheduling with general positional deterioration and rate-modifying maintenance. Omega 40:791–804

Rustogi K, Strusevich VA (2012b) Simple matching vs linear assignment in scheduling models with positional effects: A critical review. Eur J Oper Res 222:393–407

Rustogi K, Strusevich VA (2014) Combining time and position dependent effects on a single machine subject to rate-modifying activities. Omega 42:166–178

Rustogi K, Strusevich VA (2015) Single machine scheduling with time-dependent deterioration and rate-modifying maintenance. J Oper Res Soc 66:500–515

# Chapter 13
# Scheduling with Fixed Compulsory Maintenance Periods

A large volume of research address scheduling models in which a machine is not permanently available but may become non-available for processing during one or more given intervals. Such intervals are referred to as *intervals of machine non-availability*. One of the possible meaningful interpretations of a machine non-availability interval is that a mandatory machine maintenance must be performed during such an interval.

In this chapter, we consider single machine problems, provided that several such *compulsory maintenance periods* (*CMPs*) have to be inserted into any feasible schedule.

In a general case, assume that the total number of CMPs is equal to $K \geq 1$, and the $k$th CMP is denoted by $I_k = [s_k, t_k]$, $1 \leq k \leq K$. The duration of a CMP $I_k$ is denoted by $\Delta_k = t_k - s_k$.

For a job $j \in N = \{1, 2, \ldots, n\}$, its processing time $p_j$ is given and remains constant throughout the planning period. For a given sequence of jobs, a job $\ell$ that starts before a CMP but cannot be fully completed before it begins is called a *crossover* job. Depending on how a crossover job is handled, we distinguish between two *scenarios*:

- *resumable*: a crossover job is interrupted when a CMP begins and resumes after the CMP from the point of interruption;
- *non-resumable*: a crossover job restarts from scratch after the CMP.

It is not within the scope of this book to give a comprehensive exposition of scheduling problems with CMPs. The main reason is that in these models the decision-maker simply faces the CMPs as intervals of machine non-availability, they do not affect the processing times of jobs which remain constant. This does not correspond to the spirit of this book, which focuses on changing times and the maintenance periods that may alter processing conditions. Thus, in this chapter we overview known results on scheduling with CMPs for a basic single machine

V.A. Strusevich and K. Rustogi, *Scheduling with Times-Changing Effects and Rate-Modifying Activities*, International Series in Operations Research & Management Science 243, DOI 10.1007/978-3-319-39574-6_13

environment. The main focus will be on the complexity and approximability issues of the problems under the resumable and non-resumable scenarios. In fact, unless stated otherwise, in the problems considered in the main body of the chapter there will be only one CMP. Alternatively, we will consider the model with periodic maintenance.

As elsewhere in the book, we mainly consider the objective functions of makespan and (weighted) total flow time. Notice that problems of this range have generated most publications in the area of scheduling with machine non-availability intervals. Section 13.5 contains discussions and reviews of relevant results not presented in detail in the main body of this chapter.

In scheduling problems discussed in this chapter, a job $j \in N$ may be associated with a weight $w_j$ associated with job $j$, which indicates its relative importance. All values $p_j$ and $w_j$ are positive integers. The machine processes at most one job at a time, with no processing during the CMPs. The completion time of job $j \in N$ in a feasible schedule $S$ is denoted by $C_j(S)$, or shortly $C_j$ if it is clear which schedule is referred to.

Extending standard notation for the scheduling problems, we generically denote the problem considered in this chapter by $1|CMP(K), Sc|\Phi$, where $CMP(K)$ means that there are $K \geq 1$ CMPs in a schedule, $\Phi \in \left\{C_{\max}, \sum C_j, \sum w_jC_j\right\}$ denotes the objective function and $Sc \in \{Res, N-Rres\}$ specifies either the resumable scenario or the non-resumable scenario. Throughout this chapter, we often use notation $F(S) = \sum C_j(S)$ and $Z(S) = \sum w_jC_j(S)$. Let $S^*$ denote a schedule that is optimal for a scheduling problem of minimizing a function $\Phi(S)$, i.e., $\Phi(S^*) \leq \Phi(S)$ for all feasible schedules $S$.

For those problems which are NP-hard, we present the results on design and analysis of approximation algorithms and schemes. As defined in Sect. 1.3.4, a polynomial-time algorithm that finds a feasible schedule $S^H$ such that the inequality $\Phi(S^H)/\Phi(S^*) \leq \rho$ holds for all instances of the problem is called a *$\rho$-approximation* algorithm and $\rho \geq 1$ is called a *worst-case ratio bound.* A fully polynomial-time approximation scheme (FPTAS) is a family of $\rho$-approximation algorithms such that for any positive $\varepsilon > 0$ (i) $\rho = 1 + \varepsilon$ and (ii) the running time depends polynomially on the length of the problem's input and $1/\varepsilon$.

For problem $1|CMP(1), Res|\Phi$ under the resumable scenario, suppose that the jobs are sequenced in accordance with some permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$, and let $\ell$ denote the position of the crossover job, i.e., $\pi(\ell)$ is the job that cannot be competed before time $s$ and is resumed after time $t$. Under the resumable scenario, the crossover job starts at time $\sum_{j=1}^{\ell-1} p_{\pi(j)}$ and then is processed for $x_{\pi(\ell)} = s - \sum_{j=1}^{\ell-1} p_{\pi(j)}$ time units, is interrupted at time $s$, and is processed after time $t$ for $y_{\pi(\ell)}$ time units, where $y_{\pi(\ell)} = p_{\pi(\ell)} - x_{\pi(\ell)} = \sum_{j=1}^{\ell} p_{\pi(j)} - s$. Thus, the completion times for the jobs can be written as

$$C_{\pi(r)} = \begin{cases} \sum_{j=1}^{r} p_{\pi(j)}, & 1 \le r \le \ell - 1; \\ \Delta + \sum_{j=1}^{\ell} p_{\pi(j)}, & \ell \le i \le n, \end{cases} \tag{13.1}$$

where $\Delta = t - s$ is the duration of the CMP.

A feasible schedule for problem $1|CMP(1), N - Res|\Phi$ under the non-resumable scenario is defined by a partition of set $N$ into two groups $N^{[1]}$ and $N^{[2]}$, and the sequences $\pi^{[1]}$ and $\pi^{[2]}$ of these jobs, respectively, such that

(i) the jobs of the first group $N^{[1]}$ are processed from time zero as a block, without intermediate idle, follow sequence $\pi^{[1]}$, and complete before time $s$, i.e., $p\left(N^{[1]}\right) \le s$;
(ii) the jobs of the second group $N^{[2]}$ are processed from time $t$ as a block, without intermediate idle, and follow sequence $\pi^{[2]}$.

Scanning the jobs of set $N$ in a certain order introduces a Boolean variable $x_j$ in such a way that

$$x_j = \begin{cases} 1, & \text{if job } j \text{ completes before the CMP } I \\ 0, & \text{otherwise.} \end{cases} \tag{13.2}$$

Thus, $N^{[1]} = \left\{j \in N | x_j = 1\right\}$ and $N^{[2]} = N \setminus N^{[1]}$. For problem $1|CMP(1), N - Res|\Phi$ with $\Phi \in \left\{C_{\max}, \sum C_j, \sum w_j C_j\right\}$ a feasible schedule $S$ is associated with a vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ with $n$ Boolean components such that

$$\sum_{j=1}^{n} p_j x_j \le s, \tag{13.3}$$

which corresponds to the requirement that the jobs of the first group $N^{[1]}$ must be completed by time $s$. In what follows, we may refer to the value of the objective function computed not with respect to a particular schedule $S$ but with respect to the associated vector $\mathbf{x}$, so that we may write $\Phi(S)$ and $\Phi(\mathbf{x})$, whichever is more convenient.

The structure of this chapter is as follows. In Sect. 13.1, we consider the complexity and approximability issues of the problem of minimizing the makespan under both scenarios, including the model with periodic maintenance. The complexity of the problem of minimizing the sum of the weighted completion times with a single CMP is resolved in Sect. 13.2 (for the non-resumable scenario) and in Sect. 13.3 (for the resumable scenario). Approximation algorithms and schemes for these problems are presented in Sect. 13.4.

## 13.1 Makespan: Complexity and Approximation

In this section, we consider the problems of minimizing the makespan in the presence of a single compulsory maintenance period (CMP) $I = [s, t]$ with a fixed start time $s$ and of fixed duration $\Delta = t - s$. The resumable and non-resumable scenarios are studied. Besides, we also consider the model in which a CMP of duration $\Delta$ occurs periodically, after each $T$ time units.

### *13.1.1 Single Maintenance*

It is fairly easy to establish the complexity status of the problem $1|CMP(K), Sc| C_{\max}$, of minimizing the makespan with $Sc \in \{Res, N - Res\}$.

Problem $1|CMP(1), Res|C_{\max}$, is trivially solvable. Indeed, for a schedule $S_{\mathrm{Res}}$ associated with an arbitrary permutation $\pi$ of jobs, the equalities (13.1) imply that

$$C_{\max}(S_{\mathrm{Res}}) = \Delta + \sum_{j=1}^{n} p_{\pi(j)} = \Delta + p(N). \tag{13.4}$$

This observation can be extended to any number of CMPs.

**Theorem 13.1** *Problem $1|CMP(K), Res|C_{\max}$, for any $K \geq 1$ is solvable in $O(n)$ by sequencing the jobs arbitrary.*

To resolve the complexity status of problem $1|CMP(1), N - Res|C_{\max}$, using (13.2), define a vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ with $n$ Boolean components such that (13.3) holds. The value of the makespan for a schedule $S$ associated with a feasible vector $\mathbf{x}$ is given by

$$C_{\max}(S) = C_{\max}(\mathbf{x}) = t + \sum_{j=1}^{n} p_j\left(1 - x_j\right) = t + p(N) - \sum_{j=1}^{n} p_j x_j. \tag{13.5}$$

Disregarding the additive constant $t + p(N)$, finding a schedule $S^*$ that is optimal for problem $1|CMP(1), N - Res|C_{\max}$, can be reduced to the subset-sum problem, see Sect. 4.2. In such a problem, it is required to find a vector $\mathbf{x}^* = \left(x_1^*, x_2^*, \ldots, x_n^*\right)$, which delivers an optimal solution to the problem

$$\begin{aligned} &\text{maximize} \sum_{j=1}^{n} p_j x_j \\ &\text{subject to} \sum_{j=1}^{n} p_j x_j \leq s \\ &\qquad x_j \in \{0, 1\},\ j = 1, 2, \ldots, n. \end{aligned} \tag{13.6}$$

According to Sect. 4.2, this problem is NP-hard but is solvable in pseudopolynomial time, so that we deduce the following statement.

**Theorem 13.2** *Problem* $1|CMP(1), N - Res|C_{\max}$*, is NP-hard in the ordinary sense.*

Recall that the subset-sum problem is known to admit a fully polynomial-time approximation scheme (FPTAS) that requires $O(n/\varepsilon)$ time; see Theorem 4.5. Based on this scheme, we establish the approximability status of problem $1|CMP(1), N - Res|C_{\max}$.

**Theorem 13.3** *Problem* $1|CMP(1), N - Res|C_{\max}$ *admits an FPTAS that requires* $O(n/\varepsilon)$ *time.*

*Proof* Problem $1|CMP(1), N - Res|C_{\max}$, reduces to the subset-sum problem (13.6). According to Theorem 4.5, an FPTAS for the problem (13.6) either finds an optimal solution $x_j^* \in \{0, 1\}$, $j \in N$, such that

$$\sum_{j \in N} p_j x_j^* < (1 - \varepsilon)s$$

or finds an approximate solution $x_j^\varepsilon \in \{0, 1\}$, $j \in N$, such that

$$(1 - \varepsilon)s \le \sum_{j \in N} p_j x_j^\varepsilon \le s.$$

In the case of the first outcome, associate the vector $\mathbf{x}^* = \left(x_1^*, x_2^*, \ldots, x_n^*\right)$ that delivers an optimal solution to the problem (13.6) with a schedule in which the jobs of the set $H^{[1]} = \left\{x_j^* = 1 | j \in N\right\}$ form the first group, while the jobs of the set $H^{[2]} = N \backslash H^{[1]}$ form the second group. For any partition of set $N$ into two subsets $N_1$ and $N_2$ such that $p(N_1) \le s$, we have that $p\left(H^{[2]}\right) \le p(N_2)$, i.e., due to (13.5) a schedule associated with the partition $N = H^{[1]} \cup H^{[2]}$ is an optimal schedule $S^*_{\text{N-Res}}$ with the makespan $C_{\max}\left(S^*_{\text{N-Res}}\right) = t + p\left(H^{[2]}\right)$.

In the case of the second outcome, define two groups of jobs $N^{[1]} := \left\{x_j^\varepsilon = 1 | j \in N\right\}$ and $N^{[2]} := N \backslash N^{[1]}$ and consider the associated schedule $S^\varepsilon$. Since $p\left(N^{[1]}\right) \ge (1 - \varepsilon)s$, we have that $p\left(N^{[2]}\right) = p(N) - p\left(N^{[1]}\right) \le p(N) - (1 - \varepsilon)s$. It follows that

$$C_{\max}(S^\varepsilon) = t + p\left(N^{[2]}\right) \le (t - s) + p(N) + \varepsilon s.$$

Due to (13.4), we have that $C_{\max}\left(S^*_{\text{N-Res}}\right) \ge C_{\max}\left(S^*_{\text{Res}}\right) = \Delta + p(N)$, and besides, $C_{\max}\left(S^*_{\text{N-Res}}\right) \ge s$. This results in

$$C_{\max}(S^\varepsilon) \le C_{\max}\left(S^*_{\text{N-Res}}\right) + \varepsilon C_{\max}\left(S^*_{\text{N-Res}}\right),$$

which proves the theorem. □

The approximability status of the problem changes dramatically if more than one CMPs is introduced, as demonstrated below.

**Theorem 13.4** *For a fixed $\rho \geq 1$, the existence of a polynomial-time $\rho$-approximation Algorithm H for problem $1|CMP(2), N-Res|C_{\max}$ implies that $\mathcal{P} = \mathcal{NP}$.*

*Proof* To prove the theorem, we show that Algorithm H, if existed, would solve to optimality an NP-hard problem $1|CMP(1), N-Res|C_{\max}$.

Let $S_1^*$ be a schedule that is optimal for problem $1|CMP(1), N-Res|C_{\max}$, with a single CMP $I_1$. Consider an arbitrary instance of the decision version of this problem, in which it is required to verify whether there exists a schedule $S$ such that $C_{\max}(S) \leq y$ for a given $y$.

Define the instance of problem $1|CMP(2), N-Res|C_{\max}$, which is obtained from the taken instance of the decision version of problem $1|CMP(1), N-Res|C_{\max}$, by inserting an extra CMP $I_2 = \left[y, \rho y\right]$. Let $S_2^*$ be a schedule that is optimal for this problem with two CMPs.

If $C_{\max}(S_1^*) \leq y$, then $C_{\max}(S_2^*) = C_{\max}(S_1^*)$. Otherwise, the jobs that are processed after time $y$ in schedule $S_1^*$ have to be processed after time $\rho y$ in schedule $S_2^*$. Since no preemption is allowed, it follows that $C_{\max}(S_2^*) \geq \rho y + C_{\max}(S_1^*) - y$.

Apply Algorithm H to the defined instance of problem $1|CMP(2), N-Res|C_{\max}$. It will find a schedule $S_H$ such that

$$\frac{C_{\max}(S_H)}{C_{\max}(S_2^*)} \leq \rho.$$

We show that by verifying the value of $C_{\max}(S_H)$ it is possible to deduce whether for problem $1|CMP(1), N-Res|C_{\max}$, a schedule $S_1^*$ with $C_{\max}(S_1^*) \leq y$ exists.

Suppose that $C_{\max}(S_H) \leq \rho y$. Then, the actual completion time of schedule $S_H$ is before the second CMP, i.e., $C_{\max}(S_H) \leq y$. It is obvious then that

$$C_{\max}(S_1^*) \leq C_{\max}(S_2^*) \leq C_{\max}(S_H) \leq y,$$

so that for problem $1|CMP(1), N-Res|C_{\max}$ the required schedule exists.

Suppose now that $C_{\max}(S_H) > \rho y$. Then, the inequality $C_{\max}(S_2^*) \leq y$ would imply

$$\frac{C_{\max}(S_H)}{C_{\max}(S_2^*)} > \frac{\rho y}{y} = \rho,$$

a contradiction. Therefore, $C_{\max}(S_2^*) > y$. If $C_{\max}(S_1^*) \leq y$, then we would have $C_{\max}(S_2^*) = C_{\max}(S_1^*) \leq y$, a contradiction. Thus, for problem $1|CMP(1), N-Res|C_{\max}$ the required schedule does not exist. □

### 13.1.2 Periodic Maintenance

In industrial environment, the compulsory maintenance often takes place periodically, with a fixed period of time $T$ that should elapse before the first CMP and between any pair of consecutive CMPs. In the remainder of this section, we consider the problem of minimizing the makespan under the non-resumable scenario and periodic maintenance. As before, let $\Delta$ denote the duration of a CMP. We refer to this problem as $1|CMP(period), N-Res|C_{\max}$. It is assumed that $T \geq \max\{p_j | j \in N\}$; otherwise, the problem has no feasible solution.

In any schedule $S$ that is feasible for problem $1|CMP(period), N-Res|C_{\max}$, the set $N$ of jobs is split into $b \geq 1$ groups $N^{[1]}, N^{[2]}, \ldots, N^{[b]}$, where the jobs in group $N^{[q]}$ are processed in any order in the time interval $\left[(q-1)(T+\Delta), qT+(q-1)\Delta\right]$, $1 \leq q \leq b-1$, while the jobs of group $N^{[b]}$ are processed starting from time $(b-1)(T+\Delta)$. In other words, the jobs of group $N^{[1]}$ are processed before the first CMP, the jobs of each group $N^{[q]}$ are processed between the $(q-1)$th CMP and the $q$th CMP, $2 \leq q \leq b-1$, while the jobs of group $N^{[b]}$ are processed after the last used CMP. Due to feasibility, $p\left(N^{[q]}\right) \leq T$, $1 \leq q \leq b$.

Below we show that for problem $1|CMP(period), N-Res|C_{\max}$ a 2-approximation algorithm is the best possible. The proof uses the following NP-complete problem; see Sect. 1.3.2.

Partition: Given positive integers $e_1, \ldots, e_r$ and the index set $R = \{1, \ldots, r\}$ such that $e(R) = \sum_{i \in R} e_i = 2R$, is it possible to partition set $R$ into disjoint subsets $R_1$ and $R_2$ such that $e(R_1) = \sum_{i \in R_1} e_i = E$ and $e(R_2) = \sum_{i \in R_2} e_i = E$?

**Theorem 13.5** *If there exists a polynomial-time Algorithm H, which for problem $1|CMP(period), N-Res|C_{\max}$ finds a schedule $S_H$ such that for some $\varepsilon$, $0 < \varepsilon < 1$, the bound $C_{\max}(S_H)/C_{\max}(S^*) \leq 2-\varepsilon$ holds, then $\mathcal{P} = \mathcal{NP}$.*

*Proof* Given an instance of Partition, we define a particular instance of problem $1|CMP(period), N-Res|C_{\max}$, such that from a solution found by Algorithm H applied to that instance it would be possible to deduce whether Partition has a solution.

Take an $\varepsilon$, $0 < \varepsilon < 1$, define

$$N = R;\ n = r;\ p_j = e_j;\ T = E;\ \Delta = \lceil 2E(1-\varepsilon)/\varepsilon \rceil.$$

Suppose that Partition has a solution, and $R_1$ and $R_2$ are found subsets of $R$ with $e(R_1) = e(R_2) = E$. For the constructed instance of problem $1|CMP(period), N-Res|C_{\max}$, consider a schedule $S$, with two groups $N^{[q]} = R_q$, $q \in \{1, 2\}$. Since $p\left(N^{[1]}\right) = E$, the jobs of this group are completed before time $T$, i.e., before the first CMP. Thus, schedule $S$ is feasible and $C_{\max}(S) = p(N) + \Delta = 2E + \Delta$, i.e., this schedule is in fact an optimal schedule $S^*$, due to (13.4). Thus, if $C_{\max}(S^*) > 2E + \Delta$, then Partition has no solution.

On the other hand, if Partition has no solution, then in any feasible schedule $S$ for the first group $N^{[1]}$, we have that $p\left(N^{[1]}\right) \leq E-1$. To process the remaining jobs, they have to be partitioned in at least two groups $N^{[2]}$ and $N^{[3]}$, with $p\left(N^{[3]}\right) \geq 1$. This implies that $C_{\max}(S^*) \geq p(N) + 2\Delta + 1 = 2E + 2\Delta + 1$. Thus, if $C_{\max}(S^*) \leq 2E + 2\Delta$, then Partition has a solution.

Suppose that Algorithm H exists and for the constructed instance of problem $1|CMP(period), N - Res|C_{\max}$ outputs a schedule $S_H$ such that $C_{\max}(S_H)/C_{\max}(S^*) \leq 2 - \varepsilon$.

If $C_{\max}(S_H) \leq 2(E + \Delta)$, then $C_{\max}(S^*) \leq 2(E + \Delta)$, and Partition has a solution. Otherwise, if $C_{\max}(S_H) > 2(E + \Delta)$, then

$$C_{\max}\left(S^*\right) \geq \frac{C_{\max}(S_H)}{2 - \varepsilon} > \frac{2(E + \Delta)}{2 - \varepsilon}.$$

By definition of $\Delta$, we derive that $\Delta\varepsilon \geq 2E(1 - \varepsilon)$, so that $\varepsilon \geq 2E/(2E + \Delta)$. Substituting this inequality, we obtain

$$C_{\max}\left(S^*\right) > \frac{2(E + \Delta)}{2 - \frac{2E}{2E+\Delta}} = 2E + \Delta,$$

and Partition has no solution. □

In fact, problem $1|CMP(period), N - Res|C_{\max}$ admits several polynomial-time 2-approximation algorithms, which due to Theorem 13.5 should be seen as best possible. Most of these algorithms are applicable to problems more general than $1|CMP(period), N - Res|C_{\max}$; some of them are discussed in Sect. 14.2.2. Below, for illustration, we present one such algorithm.

Assume that the jobs are numbered in non-increasing order of their processing times, i.e., in the LPT order given by

$$p_1 \geq p_2 \geq \cdots \geq p_n. \tag{13.7}$$

Consider the following approximation algorithm for problem $1|CMP(period), N - Res|C_{\max}$.

**Algorithm LPT_Period**

**Step 1**. If required, renumber the jobs in accordance with (13.7). Define

$$b := 1,\ N^{[1]} := \varnothing,\ p(N^{[1]}) := 0,\ q := 1.$$

**Step 2**. For $j$ from 1 to $u$ do

**(a)** If $p\left(N^{[q]}\right) + p_j > T$, then go to Step 2(b); otherwise, update

$$N^{[q]} := N^{[q]} \cup \{j\};\ p\left(N^{[q]}\right) := p\left(N^{[q]}\right) + p_j$$

and go to Step 2(c).

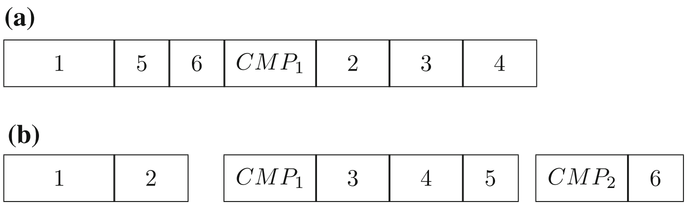


**Fig. 13.1** **a** An optimal schedule $S^*$; **b** schedule $S_{\text{LPT}}$

**(b)** If $q < b$, then update $q := q + 1$ and return to Step 2(a); otherwise, define

$$b := b + 1,\ N^{[b]} := \{j\},\ p(N^{[b]}) := p_j.$$

**(c)** Restore $q := 1$ and take the next job.

**Step 3**. Output a schedule $S_{\text{LPT}}$ that consists of $b$ found groups $N^{[1]}, N^{[2]}, \dots, N^{[b]}$.

Algorithm LPT_Period takes the jobs in the LPT order and assigns the next job to the first available group it fits. If the job does not fit into any available group, then a new group is started. The running time of the algorithm is $O(n^2)$. To see that, notice that in the worst case to find a placement for job $j$ up to $j - 1$ comparisons may be needed. The following theorem regarding its worst-case performance can be proved.

**Theorem 13.6** *For schedule $S_{\text{LPT}}$ the following bound*

$$\frac{C_{\max}(S_{\text{LPT}})}{C_{\max}(S^*)} \le 2 \qquad (13.8)$$

*holds, and this bound is tight.*

A discussion of algorithms similar to Algorithm LPT_Period is given in Sect. 13.5. The proof of Theorem 13.6 can be deduced from the proofs of more general statements given in Sect. 14.2.2. Below we only present a numerical example which demonstrates that a worst-case bound of 2 in (13.8) is tight.

*Example 13.1* To see that the bound (13.8) is tight, consider the instance of problem $1|CMP(period), N - Res|C_{\max}$ with the following parameters

$$p_1 = 6,\ p_2 = p_3 = p_4 = 4,\ p_5 = p_6 = 3;\ T = 12,$$

and the duration of a CMP set equal to some value $\Delta$.

An optimal schedule $S^*$ shown in Fig. 13.1a has no gaps and uses only one CMP and all jobs are completed by the beginning of the second CMP, so that $C_{\max}(S^*) = 24 + \Delta$. Schedule $S_{\text{LPT}}$ shown in Fig. 13.1b has two gaps of total duration 3 and uses two CMPs, so that $C_{\max}(S_{\text{LPT}}) = 27 + 2\Delta$. We deduce that

$$\frac{C_{\max}(S_{\text{LPT}})}{C_{\max}(S^*)} = \frac{27 + 2\Delta}{24 + \Delta},$$

and this ratio approaches 2 as $\Delta \to \infty$.

## 13.2 Weighted Total Flow Time: Complexity for the Non-resumable Scenario

In this section, we consider problem $1|CMP(1), N - Res|\sum w_j C_j$ of minimizing the sum of the weighted completion times with a single compulsory maintenance period (CMP) represented by a given time interval $I = [s, t]$ of length $\Delta = t - s$. The focus is on the non-resumable scenario.

### *13.2.1 Properties of the Objective Function*

We state the properties of the objective function and establish the link between problem $1|CMP(1), N - Res|\sum w_j C_j$ and the knapsack problem with a quadratic objective function related to the half-product; see Sects. 4.3 and 4.4. Consider a schedule $S_{N-Res}$ that is feasible for problem $1|CMP(1), N - Res|\sum w_j C_j$. Assume that $S_{N-Res}$ is associated with two groups $N^{[q]}$ which are ordered in accordance with permutation $\pi^{[q]}$, respectively, $q \in \{1, 2\}$. Denote $n^{[q]} = \left|N^{[q]}\right|$, let $\pi^{[q]}(i)$ be the job located in the $i$th position of permutation $\pi^{[q]}$, where $1 \le i \le n^{[q]}$. We deduce that

$$C_{\pi^{[1]}(j)} = \sum_{i=1}^{j} p_{\pi^{[1]}(i)},\ 1 \le j \le n^{[1]};\ C_{\pi^{[2]}(j)} = t + \sum_{i=1}^{j} p_{\pi^{[2]}(i)},\ 1 \le j \le n^{[2]}. \quad (13.9)$$

Thus, for a schedule $S_{\text{N-Res}}$ associated with the partition $N^{[1]}$ and $N^{[2]}$ and sequences $\pi^{[1]}$ and $\pi^{[2]}$, the value of the objective function can be written as

$$\begin{aligned} Z(S_{\text{N-Res}}) &= \sum_{j=1}^{n^{[1]}} w_{\pi^{[1]}(j)} \sum_{i=1}^{j} p_{\pi^{[1]}(i)} + \sum_{j=1}^{n^{[2]}} w_{\pi^{[2]}(j)} \left( \sum_{i=1}^{j} p_{\pi^{[2]}(i)} + t \right) \qquad (13.10) \\ &= \sum_{j=1}^{n^{[1]}} w_{\pi^{[1]}(j)} \sum_{i=1}^{j} p_{\pi^{[1]}(i)} + \sum_{j=1}^{n^{[2]}} w_{\pi^{[2]}(j)} \sum_{i=1}^{j} p_{\pi^{[2]}(i)} + t \sum_{j=1}^{n^{[2]}} w_{\pi^{[2]}(j)}. \end{aligned}$$

It is clear that if a partition $N = N^{[1]} \cup N^{[2]}$ is fixed, then in order to minimize $Z(S)$ of the form (13.10) we need to minimize the sum of the weighted completion times of the jobs in set $N^{[1]}$ and in set $N^{[2]}$. This can be done by solving problems $1||\sum w_jC_j$ with a permanently available machine for the jobs of set $N^{[1]}$ and for those of set $N^{[2]}$. It follows from Theorem 2.6 that for this purpose the jobs of each set must be ordered in accordance with the WSPT rule, i.e., in non-decreasing order of the ratios $p_j/w_j$. Thus, for problem $1|CMP(1), N - Res|\sum w_jC_j$ throughout of this chapter, unless stated otherwise, the jobs are numbered in such a way that

$$\frac{p_1}{w_1} \le \frac{p_2}{w_2} \le \cdots \le \frac{p_n}{w_n}. \tag{13.11}$$

Notice that for the unweighted version of the problem, i.e., for problem $1|CMP(1), N - Res|\sum C_j$, we assume that the jobs are numbered in accordance with the SPT rule, i.e., in such a way that

$$p_1 \le p_2 \le \cdots \le p_n. \tag{13.12}$$

Thus, our further consideration of the problems under the non-resumable scenario is based on the following statement.

**Lemma 13.1** *For problem* $1|CMP(1), N - Res|\sum w_jC_j$ *(correspondingly, problem* $1|CMP(1), N - Res|\sum C_j$*) there exists an optimal schedule in which the jobs scheduled before the CMP and those scheduled after the CMP are processed in the WSPT order (correspondingly, in the SPT order).*

For function $\Phi \in \{F, Z\}$, consider the value $\Phi_0(\pi)$ of the objective function in problem $1||\Phi$, with a continuously available machine. In particular, define

$$F_0(\pi) := \sum_{j=1}^{n} \sum_{i=1}^{j} p_{\pi(i)} \tag{13.13}$$

$$Z_0(\pi) := \sum_{j=1}^{n} w_{\pi(j)} \sum_{i=1}^{j} p_{\pi(i)}. \tag{13.14}$$

For the numbering (13.11), let $\pi^{\text{WSPT}}$ be the identity permutation $(1, 2, \ldots, n)$. It follows from Theorem 2.6 that the inequality

$$Z_0\left(\pi^{\text{WSPT}}\right) = \sum_{1 \le i \le j \le n} p_i w_j \le Z_0(\pi) \tag{13.15}$$

holds for any permutation $\pi$.

We now establish links between problem $1|CMP(1), N-Res|\sum w_jC_j$ and problems of Boolean quadratic programming.

Using (13.2), take the jobs in the order of their numbering and define a vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ with $n$ Boolean components such that (13.3) holds. If job $j$ completes before interval $I$, then

$$C_j = \sum_{i=1}^{j} p_i x_i, \tag{13.16}$$

while if it completes after interval $I$, then

$$C_j = t + \sum_{i=1}^{j} p_j(1-x_j),$$

Thus, for a vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ with $n$ Boolean components, the sum of the weighted completion times can be written as

$$\begin{aligned} Z(\mathbf{x}) &= \sum_{j=1}^{n} w_j x_j \sum_{i=1}^{j} p_i x_i + \sum_{j=1}^{n} w_j (1-x_j)\left(t + \sum_{i=1}^{j} p_i(1-x_i)\right) \\ &= \sum_{1\le i\le j\le n} p_i w_j x_i x_j + \sum_{1\le i\le j\le n} p_i w_j (1-x_i)(1-x_j) + t\sum_{j=1}^{n} w_j(1-x_j). \end{aligned}$$

For a vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ with Boolean components, define

$$f(\mathbf{x}) := \sum_{1\le i\le j\le n} p_i w_j x_i x_j + \sum_{1\le i\le j\le n} p_i w_j (1-x_i)(1-x_j), \tag{13.17}$$

so that

$$Z(\mathbf{x}) = f(\mathbf{x}) + t\sum_{j=1}^{n} w_j(1-x_j). \tag{13.18}$$

Taking into account that

$$x_j^2 = x_j,\ j \in N \tag{13.19}$$

which holds for $x_j \in \{0, 1\}$, we deduce that problem $1|CMP(1), N-Res|\sum w_jC_j$ can be formulated as the following Boolean quadratic programming problem

$$\begin{aligned}\text{minimize } Z(\mathbf{x}) = &\sum_{1\le i<j\le n} p_i w_j x_i x_j + \sum_{1\le i<j\le n} p_i w_j (1-x_i)(1-x_j)\\ &+t\sum_{j=1}^{n} w_j(1-x_j) + \sum_{j=1}^{n} p_j w_j \\ \text{subject to } &\sum_{j=1}^{n} p_j x_j \le s \\ &x_j \in \{0,1\},\ j=1,2,\ldots,n.\end{aligned} \tag{13.20}$$

Recall the definition (4.31) of a symmetric quadratic function. If in (4.31) we define

$$\alpha_j = p_j,\ \beta_j = w_j,\ \mu_j = 0,\ \nu_j = w_j t,\ j=1,2,\ldots,n;\ A = s,\ K = \sum_{j=1}^{n} p_j w_j,$$

then the objective function in (13.20) becomes a special case of (4.31). Thus, according to the terminology adapted in Sect. 4.4 problem (13.20) is an instance of a symmetric quadratic knapsack problem.

### 13.2.2 *Useful Lower Bounds*

Notice that the value $f(\mathbf{x})$ defined by (13.17) can be understood as the sum of the weighted completion times in problem $P2| |\sum w_j C_j$ on two parallel identical machines for a schedule in which the jobs of set $\{j \in N | x_j = 1\}$ are assigned to one machine, while the other jobs are assigned to the other, and on each machine the jobs are processed in accordance with their numbering given by (13.11); see Sect. 4.3.1. Intuitively, $f(\mathbf{x})$ should not be larger than $Z_0(\pi^{\mathrm{WSPT}})$ defined in (13.15). The formal proof of this fact is below.

**Lemma 13.2** *For any Boolean vector* $\mathbf{x} = (x_1, \ldots, x_n)$*, the inequality*

$$Z_0\left(\pi^{\mathrm{WSPT}}\right) = \sum_{1\le i\le j\le n} p_i w_j \ge \sum_{1\le i\le j\le n} p_i w_j x_i x_j + \sum_{1\le i\le j\le n} p_i w_j (1-x_i)(1-x_j) = f(\mathbf{x}) \tag{13.21}$$

*holds.*

*Proof* For any Boolean vector $\mathbf{x} = (x_1, \ldots, x_n)$, we have that for any pair of indices $i$ and $j$, $1 \le i \le j \le n$, the inequality $x_i x_j + (1-x_i)(1-x_j) \le 1$ holds, so that

$$\sum_{1\le i\le j\le n} p_i w_j \ge \sum_{1\le i\le j\le n} p_i w_j x_i x_j + \sum_{1\le i\le j\le n} p_i w_j (1-x_i)(1-x_j) = f(\mathbf{x}),$$

as required. □

The following lower bound can be derived.

**Lemma 13.3** *For problem (13.20), let* $\mathbf{x} = (x_1, \dots, x_n)$ *be a Boolean vector that defines a feasible solution vector, and* $\mathbf{x}^* = (x_1^*, \dots, x_n^*)$ *be a Boolean vector that is associated with an optimal solution. The following lower bound*

$$f(\mathbf{x}^*) + s\sum_{j=1}^{n} w_j(1 - x_j^*) \geq f(\mathbf{x}). \tag{13.22}$$

*holds.*

*Proof* Notice that $\sum_{1\leq i\leq j\leq n} p_i w_j x_i^* x_j^* + \sum_{1\leq i\leq j\leq n} p_i w_j (1 - x_i^*)(1 - x_j^*) + s \sum_{j=1}^{n} w_j(1 - x_j^*)$ is the value of the objective function in a single machine scheduling problem to minimize the sum of the weighted completion times, provided that each job $j$ with $x_j^* = 1$ completes before time $s$, and each job $j$ with $x_j^* = 0$ starts after time $s$. Thus,

$$\sum_{1\leq i\leq j\leq n} p_i w_j x_i^* x_j^* + \sum_{1\leq i\leq j\leq n} p_i w_j (1 - x_i^*)(1 - x_j^*) + s\sum_{j=1}^{n} w_j(1 - x_j^*) \geq \sum_{1\leq i\leq j\leq n} p_i w_j,$$

where the right-hand side is equal to $Z_0\left(\pi^{\mathrm{WSPT}}\right)$, the optimal value of the objective function in problem $1||\sum w_j C_j$. The required lower bound follows immediately from Lemma 13.2. $\square$

Lemma 13.3 is used in the analysis of approximation algorithms for problem $1|CMP(1), N - Res|\sum w_j C_j$.

### 13.2.3 Computational Complexity

We resolve the complexity status of problem $1|CMP(1), N - Res|\sum w_j C_j$ by showing that problem $1|CMP(1), N - Res|\sum C_j$ with the unweighted objective function is NP-hard. In the proof of the NP-hardness, the following NP-complete problem is used for reduction; see Sect. 1.3.2.

Even-Odd Partition: Given positive integers $e_1, \dots, e_{2r}$ and the index set $R = \{1, \dots, 2r\}$ such that $e_i \leq e_{i+1}$ for $1 \leq i < 2r$ and $e(R) = \sum_{i\in R} e_i = 2R$, is it possible to partition set $R$ into disjoint subsets $R_1$ and $R_2$ such that $e(R_1) = \sum_{i\in R_1} e_i = E$ and $e(R_2) = \sum_{i\in R_2} e_i = E$ and for each $i$, $1 \leq i \leq r$, each set $R_1$ and $R_2$ contains exactly one element of the pair $\{2i - 1, 2i\}$?

**Theorem 13.7** *Problem* $1|CMP(1), N - Res|\sum C_j$ *is NP-hard in the ordinary sense.*

*Proof* Given an instance of EVEN-ODD PARTITION, choose integers $M$ and $P$ such that

$$M > 2rE;\ P > Q,$$

where

$$Q = \sum_{i=1}^{r}(r-i+1)(e_{2i-1}+e_i) + \left(2r^2+4r+1\right)M + (r+2)E.$$

Define the instance of problem $1|CMP(1), N-Res|\sum C_j$ with

$$\begin{aligned} &n = 2r+1;\ p_j = M+e_i,\ 1 \le i \le 2r;\ p_{2r+1} = P; \\ &s = rM+E;\ \Delta = M;\ t = (r+1)M+E. \end{aligned}$$

We show that EVEN-ODD PARTITION has a solution if and only if in the constructed instance of the problem there exists a schedule $S_0$ such that $F(S_0) \le y = Q+P$.

Suppose that EVEN-ODD PARTITION has a solution represented by the sets $R_1$ and $R_2$. Then, schedule $S_0$ with $C_{\max}(S_0) = y$ exists and can be found as follows. Define $N^{[q]} := R_q$ and let $\varphi^{[q]}$ denote a sequence of jobs of set $N^{[q]}$ sorted in the SPT order, $q \in \{1,2\}$. In schedule $S_0$, process the jobs of the first group $N^{[1]}$ in accordance with sequence $\varphi^{[1]}$ before the CMP, and the jobs of in the second group $N^{[2]}$ in accordance with sequence $\varphi^{[2]}$ starting at time $t$. Additionally, assign job $2r+1$ to be processed last. Notice that the structure of schedule $S_0$ satisfies Lemma 13.1.

It follows that

$$\sum_{i\in N^{[q]}} p_i = rM + \sum_{i\in R_q} e_i = rM+E,\ q \in \{1,2\}.$$

Since $p\left(N^{[1]}\right) = s$, we see that schedule $S_0$ is feasible. Using (13.9), compute

$$\begin{aligned} \sum_{i=1}^{r} C_{\varphi^{[1]}(i)} &= \sum_{i=1}^{r}(r-i+1)p_{\varphi^{[1]}(i)}; \\ \sum_{i=1}^{r} C_{\varphi^{[2]}(i)} &= rt + \sum_{i=1}^{r}(r-i+1)p_{\varphi^{[2]}(i)} \\ &= r((r+1)M+E) + \sum_{i=1}^{r}(r-i+1)p_{\varphi^{[2]}(i)}; \\ C_{2r+1} &= \sum_{i\in N^{[1]}} p_i + \Delta + \sum_{i\in N^{[2]}} p_i + p_{2r+1} = 2(rM+E)+M+P \\ &= (2r+1)M+2E+P. \end{aligned}$$

Since the $e$-values form a non-decreasing sequence, we obtain

$$\begin{aligned}\sum_{i=1}^{2r+1} C_i(S_0) &= \sum_{i=1}^{r}(r-i+1)p_{\varphi^{[1]}(i)} + \sum_{i=1}^{r}(r-i+1)p_{\varphi^{[2]}(i)} \\ &\quad + r((r+1)M+E) + (2r+1)M + 2E + P \\ &= \sum_{i=1}^{r}(r-i+1)(e_{2i-1}+e_{2i}) + 2M\sum_{i=1}^{r}(r-i+1) \\ &\quad + r((r+1)M+E) + (2r+1)M + 2E + P \\ &= \sum_{i=1}^{r}(r-i+1)(e_{2i-1}+e_{2i}) + r(r+1)M \\ &\quad + r((r+1)M+E) + (2r+1)M + 2E + P \\ &= \sum_{i=1}^{r}(r-i+1)(e_{2i-1}+e_{2i}) \\ &\quad + 2r(r+1)M + (2r+1)M + (r+2)E + P \\ &= Q + P = y,\end{aligned}$$

i.e., $S_0$ is indeed a required schedule.

Suppose now that a schedule $S_0$ such that $\sum_{i=1}^{2r+1} C_i(S_0) \le y$ exists, and set $N^{[1]}$ (set $N^{[2]}$) is the subset of jobs of set $\{1, 2, \dots, 2r\}$ processed before the CMP (after the CMP, respectively) in schedule $S_0$. Since $p_{2r+1} > s$, job $2r+1$ must be processed after the CMP. Due to Lemma 13.1, we may assume that the jobs of each set $N^{[1]}$ and $N^{[2]} \cup \{2r+1\}$ are processed in the SPT order; in particular, this implies that job $2r+1$ is the last job. For $q \in \{1, 2\}$, denote the sequences of jobs of set $N^{[q]}$ by $\pi^{[q]}$ and $n^{[q]} = \left|N^{[q]}\right|$.

Since

$$p_i > M, \; 1 \le i \le 2r, \tag{13.23}$$

and $s = rM + E < (r+1)M$, it follows that $n^{[1]} \le r$. Suppose that $n^{[1]} \le r-1$, so that $n^{[2]} \ge r+1$. Compute

$$\begin{aligned}\sum_{i=1}^{2r+1} C_i(S_0) &= \sum_{i=1}^{n^{[1]}}\left(n^{[1]}-i+1\right)p_{\pi^{[1]}(i)} + \sum_{i=1}^{n^{[2]}}\left(n^{[2]}-i+1\right)p_{\pi^{[2]}(i)} \\ &\quad + n^{[2]}((r+1)M+E) + C_{2r+1}.\end{aligned}$$

Due to (13.23), ignoring the contributions of the $e$-values to the processing times, we have

$$\sum_{i=1}^{n^{[q]}}\left(n^{[q]}-i+1\right)p_{\pi^{[q]}(i)} > \frac{n^{[q]}\left(n^{[q]}+1\right)}{2}M, \; q \in \{1, 2\}.$$

Notice that

$$\begin{aligned}
&\frac{n^{[1]}\left(n^{[1]}+1\right)}{2}+\frac{n^{[2]}\left(n^{[2]}+1\right)}{2}\\
&=\frac{n^{[1]}\left(n^{[1]}+1\right)}{2}+\frac{\left(2r-n^{[1]}\right)\left(2r-n^{[1]}+1\right)}{2}\\
&=2r^2-2rn^{[1]}+r+\left(n^{[1]}\right)^2=\left(r^2+r^2\right)-2rn^{[1]}+r+\left(n^{[1]}\right)^2\\
&=r(r+1)+(r-n_1)^2>r(r+1)+1,
\end{aligned}$$

where the last inequality is due to $n_1 \le r-1$.

The block of jobs of set $N^{[2]}$ starts at time $t=(r+1)M+E$, so that $C_{2r+1}>(r+1)M+E+n^{[2]}M+P$. Thus, due to $n^{[2]} \ge r+1$ we deduce

$$\begin{aligned}
\sum_{i=1}^{2r+1} C_i(S_0) &> r(r+1)M+M+(r+1)((r+1)M+E)\\
&\quad +(r+1)M+E+(r+1)M+P\\
&=r(r+1)M+M+r(r+1)M+(r+1)E+3(r+1)M+E+P\\
&=2r(r+1)M+3(r+1)M+(r+2)E+M+P\\
&=\left(2r^2+4r+1\right)M+(r+2)E+(r+3)M+P.
\end{aligned}$$

Observe that

$$\sum_{i=1}^{r}(r-i+1)(e_{2i-1}+e_{2i})<r\sum_{i=1}^{r}(e_{2i-1}+e_{2i})=2rE<M<(r+3)M,$$

so that

$$\sum_{i=1}^{2r+1} C_i(S_0)>y,$$

and we deduce that $n^{[1]}=n^{[2]}=r$. Compute

$$\begin{aligned}
\sum_{i=1}^{2r+1} C_i(S_0) &= \sum_{i=1}^{r}(r-i+1)p_{\pi^{[1]}(i)}+\left(\sum_{i=1}^{r}(r-i+1)p_{\pi^{[2]}(i)}+rt\right)+C_{2r+1}\\
&=\sum_{i=1}^{r}(r-i+1)p_{\pi^{[1]}(i)}+\left(\sum_{i=1}^{r}(r-i+1)p_{\pi^{[2]}(i)}+rt\right)\\
&\quad+\left(t+\sum_{i=1}^{r}p_{\pi^{[2]}(i)}\right)+P
\end{aligned}$$

$$= \sum_{i=1}^{r}(r-i+1)p_{\pi^{[1]}(i)} + \sum_{i=1}^{r}(r-i+1)p_{\pi^{[2]}(i)} + \sum_{i=1}^{r} p_{\pi^{[2]}(i)}$$
$$+(r+1)((r+1)M+E)+P.$$

Transforming further, we obtain

$$\begin{aligned}\sum_{i=1}^{2r+1} C_i(S_0) &= \left(\sum_{i=1}^{r}(r-i+1)e_{\pi^{[1]}(i)} + \frac{r(r+1)}{2}M\right)\\ &\quad+\left(\sum_{i=1}^{r}(r-i+1)e_{\pi^{[2]}(i)} + \frac{r(r+1)}{2}M\right)+\left(\sum_{i=1}^{r} e_{\pi^{[2]}(i)} + rM\right)\\ &\quad+(r+1)((r+1)M+E)+P\\ &= \sum_{i=1}^{r}(r-i+1)e_{\pi^{[1]}(i)} + \sum_{i=1}^{r}(r-i+1)e_{\pi^{[2]}(i)} + \sum_{i=1}^{r} e_{\pi^{[2]}(i)}\\ &\quad+\left(2r^2+4r+1\right)M+(r+1)E+P.\end{aligned}$$

Since $\sum_{i=1}^{2r+1} C_i(S_0) \le y$, we must have that

$$\begin{aligned}&\sum_{i=1}^{r}(r-i+1)e_{\pi^{[1]}(i)} + \sum_{i=1}^{r}(r-i+1)e_{\pi^{[2]}(i)} + \sum_{i=1}^{r} e_{\pi^{[2]}(i)}\\ &\le \sum_{i=1}^{r}(r-i+1)(e_{2i-1}+e_i)+E.\end{aligned}$$

We may think of the expression $\sum_{i=1}^{r}(r-i+1)(e_{2i-1}+e_i)$ as the smallest value of the sum of the completion times on two identical parallel machines for $2r$ jobs with the processing times $e_i$, $1 \le i \le 2r$, since this expression gives the value of the objective function delivered by Algorithm PSumSPT presented in Sect. 2.3.2. Thus,

$$\sum_{i=1}^{r}(r-i+1)e_{\pi^{[1]}(i)} + \sum_{i=1}^{r}(r-i+2)e_{\pi^{[2]}(i)} \ge \sum_{i=1}^{r}(r-i+1)(e_{2i-1}+e_i),$$

which implies that

$$\sum_{i=1}^{r} e_{\pi^{[2]}(i)} \le E.$$

Since $S_0$ is a feasible schedule, the inequality

$$\sum_{i=1}^{r} p_{\pi^{[1]}(i)} \le s$$

holds, which simplifies to

$$\sum_{i=1}^{r} e_{\pi^{[1]}(i)} \le E.$$

Thus, we deduce that

$$\sum_{i=1}^{r} e_{\pi^{[1]}(i)} = \sum_{i=1}^{r} e_{\pi^{[2]}(i)} = E.$$

If we set $R_q = N^{[q]}$, $q \in \{1, 2\}$, we obtain a solution to EVEN-ODD PARTITION. □

To fully resolve the complexity status of problem $1|CMP(1), N-Res|\sum w_j C_j$, we show that it can be solved in pseudopolynomial time by a dynamic programming (DP) algorithm. We present the corresponding algorithm in terms of solving the symmetric quadratic knapsack problem (13.20).

The decision variables $x_1, x_2, \ldots, x_n$ are scanned in the order of their numbering and are given either the value of 1 (an item is put into the knapsack) or 0 (an item is not put into the knapsack).

Define

$$A_k := \sum_{j=1}^{k} p_j,\, k = 1, 2, \ldots, n.$$

and suppose that the values $x_1, x_2, \ldots, x_k$ have been assigned. The described DP algorithm deals with partial solutions associated with states of the form

$$(k, Z_k, y_k),$$

where

$k$ is the number of the assigned variables;
$Z_k$ is the current value of the objective function;
$y_k := \sum_{j=1}^{k} p_j x_j$, the total weight of the items put into the knapsack.

We now give a formal statement and implementation details of the DP algorithm.

**Algorithm NResDP**

**Step 1.** Start with the initial state $(0, Z_0, y_0) = (0, \sum p_j w_j, 0)$. Compute the values $A_k = \sum_{j=1}^{k} p_j$, $k = 1, 2, \ldots, n$.

**Step 2**. For all $k$ from 0 to $n-1$ make transitions from each stored state of the form $(k, Z_k, y_k)$ into the states of the form $(k+1, Z_{k+1}, y_{k+1})$ by assigning the next variable $x_{k+1}$.

**(a)** Define $x_{k+1} = 1$, provided that item $k+1$ fits into the knapsack, i.e., if the inequality $y_k + p_{k+1} \leq s$ holds. If feasible, the assignment $x_{k+1} = 1$ changes a state $(k, Z_k, y_k)$ to a state of the form $(k+1, Z_{k+1}, y_{k+1})$ where

$$Z_{k+1} = Z_k + w_{k+1} y_k, \; y_{k+1} = y_k + p_{k+1}.$$

**(b)** Define $x_{k+1} = 0$, which is always feasible. This assignment changes a state $(k, Z_k, y_k)$ to a state of the form $(k+1, Z_{k+1}, y_{k+1})$ such that

$$Z_{k+1} = Z_k + w_{k+1}(A_k - y_k) + w_{k+1} t; \; y_{k+1} = y_k.$$

**Step 3**. Output the optimal value of the function that corresponds to the smallest value of $Z_n$ among all found states of the form $(n, Z_n, y_n)$.

Algorithm NResDP can be implemented in $O(ns)$ time.

## 13.3 Weighted Total Flow Time: Complexity for the Resumable Scenario

In this section, we consider problem $1|CMP(1), Res|\sum w_j C_j$ of minimizing the sum of the weighted completion times with a single compulsory maintenance period (CMP) represented by a given time interval $I = [s, t]$ of length $\Delta = t - s$. The focus is on the resumable scenario.

### *13.3.1 Properties of the Objective Function*

For problem $1|CMP(1), Res|\sum w_j C_j$ under the resumable scenario, given a permutation permutation $\pi$ of jobs with the crossover job $\pi(\ell)$, we may assume that the jobs of set $N \backslash \{\pi(\ell)\}$ are split into two subsets, $N_1(\pi)$ and $N_2(\pi)$ such that the jobs of set $N_1(\pi) = \{\pi(1), , \ldots, \pi(\ell-1)\}$ are sequenced before the crossover job, while the jobs of set $N_2(\pi) = \{\pi(\ell+1), , \ldots, \pi(n)\}$ are sequenced after the crossover job. Similarly to the non-resumable scenario, for the purpose of minimizing the total weighted completion time the jobs prior to the crossover job and those after the crossover job should be ordered in the WSPT order. Thus, the following statement holds.

**Lemma 13.4** *For problem $1|CMP(1), Res|\sum w_jC_j$ (correspondingly, problem $1|CMP(1), Res|\sum C_j$) there exists an optimal schedule in which the jobs scheduled before the crossover job and those scheduled after the crossover job are processed in the WSPT order (correspondingly, in the SPT order).*

Notice that unlike in Lemma 13.4 that handles the non-resumable scenario, Lemma 13.4 does not say anything about the crossover job itself. For example, in the case of problem $1|CMP(1), Res|\sum w_jC_j$ in an optimal schedule associated with a permutation $\pi$ of jobs with a crossover job $\pi(\ell)$ the sign of each of differences $\frac{p_{\pi(\ell-1)}}{w_{\pi(\ell-1)}} - \frac{p_{\pi(\ell)}}{w_{\pi(\ell)}}$ and $\frac{p_{\pi(\ell)}}{w_{\pi(\ell)}} - \frac{p_{\pi(\ell+1)}}{w_{\pi(\ell+1)}}$ can be arbitrary.

For problem $1|CMP(1), Res|\sum w_jC_j$, we use (13.1) to write out the objective function for a schedule $S_{\text{Res}}$ associated with a permutation $\pi$ and the crossover job $\pi(\ell)$ as

$$\begin{aligned}
Z(S_{\text{Res}}) &= \sum_{j=1}^{\ell-1} w_{\pi(j)} \sum_{i=1}^{j} p_{\pi(i)} + w_{\pi(\ell)}\left(t + \left(\sum_{i=1}^{\ell} p_{\pi(j)} - s\right)\right) \\
&\quad + \sum_{j=\ell+1}^{n} w_{\pi(j)}\left(t + \left(\sum_{i=1}^{j} p_{\pi(j)} - s\right)\right) \\
&= \sum_{j=1}^{\ell-1} w_{\pi(j)} \sum_{i=1}^{j} p_{\pi(i)} + w_{\pi(\ell)} \sum_{j=1}^{\ell} p_{\pi(j)} + \sum_{j=\ell+1}^{n} w_{\pi(j)} \sum_{i=1}^{j} p_{\pi(i)} \\
&\quad + \sum_{j=\ell}^{n} w_{\pi(j)}(t-s).
\end{aligned}$$

Using (13.14), we may write

$$Z(S_{\text{Res}}) = \sum_{j=1}^{n} w_{\pi(j)} \sum_{i=1}^{j} p_{\pi(i)} + \sum_{j=\ell}^{n} w_{\pi(\ell)}(t-s) = Z_0(\pi) + (t-s)\sum_{j=\ell}^{n} w_{\pi(j)}. \tag{13.24}$$

For problem $1|CMP(1), Res|\sum w_jC_j$, suppose that a certain job is chosen as the crossover job. Denote the processing time and the weight of the chosen crossover job by $p$ and $w$, respectively, and renumber the remaining jobs taken according to the WSPT rule by the integers $1, 2, \ldots, m$, where $m = n-1$.

A feasible schedule for problem $1|CMP(1), Res|\sum w_jC_j$ with a fixed crossover job can be found by inserting the crossover job into a schedule for processing the jobs $1, 2, \ldots, m$ under the non-resumable scenario. For problem $1|CMP(1), Res|$ $\sum w_jC_j$, let $S^*$ denote the optimal schedule that delivers the smallest value $Z(S^*)$ of the objective function, while $S(p, w)$ denote a feasible schedule with a fixed crossover job with the processing time $p$ and weight $w$. Denote the smallest value of the function among all schedules with the chosen crossover job by $Z^*(p, w)$.

Define the Boolean decision variables $x_j$ such that (13.2) holds for each $j$, $1 \le j \le m$. It follows from (13.20) that for an arbitrary assignment of variables $x_j$

the sum of the weighted completion times of the corresponding schedule $S_m$ under the non-resumable scenario is given by

$$Z_m = \sum_{1 \le i < j \le m} p_i w_j x_i x_j + \sum_{1 \le i < j \le m} p_i w_j (1 - x_i)(1 - x_j) + t \sum_{j=1}^{m} w_j (1 - x_j) + \sum_{j=1}^{m} p_j w_j,$$

where

$$\sum_{j=1}^{m} p_j x_j \le s, \ x_j \in \{0, 1\}, \ j = 1, 2, \ldots, m.$$

To convert a schedule $S_m$ into a schedule $S(p, w)$ that is feasible for problem $1|CMP(1), Res| \sum w_j C_j$ with the chosen crossover job, we process the crossover job for $px$ time units before the non-availability interval starting at time

$$y_m = \sum_{j=1}^{m} p_j x_j,$$

where either $x = 1$, if

$$p < s - y_m,$$

or

$$x = \frac{s - y_m}{p},$$

otherwise. The former case should be ignored, since the chosen job completes earlier than time $s$ and is not a crossover job. In the latter case, the chosen job is the crossover job that is additionally processed for $p(1 - x)$ time units starting at time $t$, and this increases the starting (and the completion) time of each job with $x_j = 0$ by $p(1 - x)$.

The value of the objective function of the resulting schedule $S(p)$ can be written as

$$Z(p) = Z_m + H(y_m, W_m, x), \tag{13.25}$$

where $W_m = \sum_{j=1}^{m} w_j (1 - x_j)$ and

$$H(y_m, W_m, x) = w(t + p(1 - x)) + W_m p(1 - x).$$

## 13.3.2 Computational Complexity

First, we establish that problem $1|CMP(1), Res| \sum C_j$ to minimize the sum of the completion times under the resumable scenario can be solved in polynomial time. In fact, the corresponding optimal schedule is specified by a sequence of jobs sorted in the SPT order, i.e., in non-decreasing order of their processing times. This should be seen as an extension of the classical result which states that problem $1||\sum C_j$ is solvable by the SPT rule; see Theorem 2.2.

**Theorem 13.8** *For problem* $1|CMP(1), Res|\sum C_j$, *an optimal schedule can be found in* $O(n \log n)$ *time, by sequencing the jobs in accordance with the SPT rule.*

*Proof* Take an arbitrary time $\tau > 0$, and let for a permutation $\pi$ of jobs the number of jobs that are completed by time $\tau$ be denoted by $n(\tau; \pi)$.

For an arbitrary sequence $\pi$ of jobs, the equality (13.24) applied to problem $1|CMP(1), Res|\sum C_j$ can be written as

$$F(S_{\text{Res}}) = \sum_{j=1}^{n}\sum_{i=1}^{j} p_{\pi(i)} + \sum_{j=\ell}^{n}(t-s) = F_0(\pi) + (t-s)(n - n(s; \pi)).$$

The nature of the SPT ordering is such that for the sequence $\pi^{\text{SPT}} = (1, 2, \ldots, n)$ the inequality $n\left(\tau; \pi^{\text{SPT}}\right) \geq n(\tau; \pi)$ holds for any $\tau$ and any $\pi$. Since for any permutation $\pi$, the inequalities

$$F_0\left(\pi^{\text{SPT}}\right) \leq F_0(\pi),\ n - n\left(s; \pi^{\text{SPT}}\right) \leq n - n(s; \pi),$$

hold, it follows that a schedule associated with permutation $\pi^{\text{SPT}}$ is optimal. □

The complexity status of the problem under the resumable scenario changes if the function become the sum of the weighted completion times. Below, we prove that problem $1|CMP(1), Res|\sum w_j C_j$ is NP-hard. As in the proof of Theorem 13.5, we use PARTITION to prove the NP-hardness.

**Theorem 13.9** *Problem* $1|CMP(1), Res|\sum w_j C_j$ *is NP-hard in the ordinary sense, even if* $p_j = w_j$, $j \in N$.

*Proof* Given an instance of PARTITION, define the instance of problem $1|CMP(1), Res|\sum w_j C_j$ with

$$N = R;\ p_j = w_j = e_j,\ j \in N;\ s = E,\ t = 2E.$$

We show that PARTITION has a solution if and only if in the constructed problem there exists a schedule $S_0$ such that $Z(S_0) \leq y = \sum_{1\leq i\leq j\leq n}\left(e_i e_j\right) + E^2$.

Suppose that PARTITION has a solution represented by the sets $R_1$ and $R_2$. Then, schedule $S_0$ with $Z(S_0) = y$ exists and can be found as follows. Define $N_q := R_q$, $q \in \{1, 2\}$ and let in $S_0$ the jobs of set $N_1$ be processed before the CMP, while the jobs of set $N_2$ be processed as a block starting from time $t$. Notice that since $p_j = w_j$, $j \in N$, the jobs of each set $N_q$ can be processed in any order. Since $p(N_1) = e(R_1) = s$, $S_0$ is a feasible schedule in which the jobs of set $N_1$ are completed before time $s$ and there is no crossover job. Notice also that $w(N_2) = E$.

Due to (13.24), we have that

$$Z(S_0) = \sum_{j=1}^{n} w_j \sum_{i=1}^{j} p_i + (t-s) \sum_{j \in N_2} w_j = \sum_{1 \le i \le j \le n} \left(e_i e_j\right) + E^2 = y,$$

as required.

Suppose now that a schedule $S_0$ such that $C_{\max}(S_0) \le y$ exists. Assume that in this schedule the jobs are processed in accordance with a permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$, and $N_q$, $q \in \{1, 2\}$ are the sets of jobs completed before the CMP and after the CMP, respectively. Thus, due to (13.24),

$$Z(S_0) = Z_0(\pi) + (t-s) \sum_{j \in N_2} w_j \le y.$$

The value $Z_0(\pi)$ reaches its minimum if the jobs are ordered in the WSPT order; however, in our case, since $p_j = w_j, j \in N$, we have that

$$Z_0(\pi) \ge \sum_{j=1}^{n} w_j \sum_{i=1}^{j} p_i = \sum_{1 \le i \le j \le n} \left(e_i e_j\right).$$

Since $S_0$ is a feasible schedule, we must have that $p(N_1) \le E$, which implies that $w(N_2) \ge E$.

Thus,

$$y \ge Z(S_0) \ge \sum_{1 \le i \le j \le n} \left(e_i e_j\right) + E^2 = y,$$

which is only possible when $Z(S_0) = y$ and $w(N_2) = E$, so that $p(N_1) = E$.

If we set $R_q = N_q$, $q \in \{1, 2, \}$, we obtain a solution to PARTITION. □

Based on the consideration presented in Sect. 13.3.1, we may apply the following approach to finding a schedule $S^*$ that is feasible for problem $1|CMP(1), Res|\sum w_j C_j$. For $j$ from 1 to $n$, choose job $j$ as the crossover job. Let the processing time of the chosen job be $p$ and its weight be $w$. Then,

(i) Find and store an optimal schedule under the non-resumable scenario.
(ii) Temporarily remove the chosen crossover job. Find schedule $S_m$ that minimizes the sum of the weighted completion times for the remaining $m = n - 1$ jobs under the non-resumable scenario.
(iii) Insert the chosen job into schedule $S_m$ to start at the completion time of the last job completed before the CMP in schedule $S_m$. Ignore the situation that the inserted job also completes before the CMP; otherwise, store the resulting schedule $S(p, w)$.
(iv) Among the stored schedules, find the one that minimizes the objective function and output it as schedule $S^*$.

This approach is based on solving $O(n)$ problems $1|CMP(1), N-Res|\sum w_j C_j$ with $O(n)$ jobs. It follows from Sect. 13.2.3 that each such problem can be solved in

$O(ns)$ time. Thus, we conclude that $1|CMP(1), Res|\sum w_jC_j$ is solvable in $O(n^2s)$ time, i.e., admits a pseudopolynomial algorithm. Together with Theorem 13.9, this observation fully resolves the complexity issue of problem $1|CMP(1), Res|\sum w_jC_j$.

## 13.4 Weighted Total Flow Time: Approximation Algorithms and Schemes

As shown in Sects. 13.2 and 13.3, problems $1|CMP(1), N-Res|\sum w_jC_j$ and $1|CMP(1), Res|\sum w_jC_j$ are NP-hard, even in the case of a single CMP. This fact has stimulated a large volume of research on design of approximation algorithms and schemes for relevant problems.

### *13.4.1 Constant Ratio Approximation Algorithms*

Given the role that the WSPT rule plays in minimizing the sum of the weighted completion times, it is most natural to study the worst-case performance of the algorithm in which the jobs are processed in the order of their numbering given by (13.11). In our description of the algorithm, we exclude from consideration the case that $p(N) \le s$, i.e., when all jobs can be completed before the CMP and the WSPT sequence is optimal.

**Algorithm CMP_WSPT**

**Step 1**. Scanning jobs in the order of their numbering (13.11), assign them to be processed from time zero until a job $\ell$ is identified such that

$$\sum_{j=1}^{\ell-1} p_j \le s, \quad \sum_{j=1}^{\ell} p_j > s.$$

**Step 2**. If $\sum_{j=1}^{\ell-1} p_j = s$, then irrespective of the scenario start job $\ell$ at time $t$; otherwise, either start job $\ell$ at time $t$ (for the non-resumable scenario) or start job $\ell$ at time $\sum_{j=1}^{\ell-1} p_j$, interrupt its processing at time $s$, and resume at time $t$ to complete at time $\sum_{j=1}^{\ell} p_j + \Delta$.

**Step 3**. Assign the remaining jobs in the order of their numbering to be processed starting from the completion time of job $\ell$. Call the resulting schedule $S_{\mathrm{Sc}}^{\mathrm{WSPT}}$ where $\mathrm{Sc} \in \{\mathrm{Res}, \mathrm{N-Res}\}$.

The running time of Algorithm CMP_WSPT is $O(n \log n)$. Below we analyze its worst-case performance.

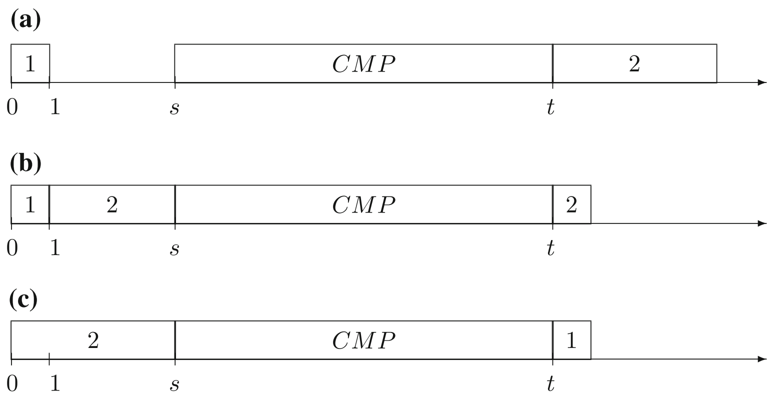


**Fig. 13.2** **a** Schedule $S_{\text{WSPT}}$ for the non-resumable scenario; **b** schedule $S_{\text{WSPT}}$ for the resumable scenario; **c** optimal schedule $S^*$

**Lemma 13.5** *For problem* $1|CMP(1), Sc|\sum w_jC_j$ *with* $Sc \in \{Res, N - Res\}$, *let* $S^*_{\text{Sc}}$ *and* $S^{\text{WSPT}}_{\text{Sc}}$ *be an optimal schedule and a schedule found by Algorithm CMP_WSPT, respectively. Then the ratio* $Z\left(S^{\text{WSPT}}_{\text{Sc}}\right)/Z\left(S^*_{\text{Sc}}\right)$ *can be arbitrary large.*

*Proof* Consider an instance of problem $1|CMP(1), Sc|\sum w_jC_j$ with two jobs such that

$$p_1 = w_1 = 1;\ p_2 = W + 1,\ w_2 = W;\ s = W + 1,\ t = W^2 + W + 1.$$

In schedule $S^{\text{WSPT}}_{\text{Sc}}$, the jobs are processed in the order $(1, 2)$, so that irrespective of the scenario $C_1\left(S^{\text{WSPT}}_{\text{Sc}}\right) = 1$. For problem $1|CMP(1), N - Res|\sum w_jC_j$, job 2 starts after the CMP, so that

$$C_2\left(S^{\text{WSPT}}_{\text{N-Res}}\right) = \left(W^2 + W + 1\right) + (W + 1) = W^2 + 2W + 2;$$

see Fig. 13.2a.

On the other hand, for problem $1|CMP(1), Res|\sum w_jC_j$ job 2 in schedule $S^{\text{WSPT}}_{\text{Res}}$ is the crossover job that is processed for 1 time unit after the CMP, so that

$$C_2\left(S^{\text{WSPT}}_{\text{Res}}\right) = W^2 + W + 2;$$

see Fig. 13.2b.

Thus, for the non-resumable scenario,

$$Z\left(S^{\text{WSPT}}_{\text{N-Res}}\right) = 1 \cdot 1 + W\left(W^2 + 2W + 2\right) = W^3 + 2W^2 + 2W + 1.$$

while for the resumable scenario,

$$Z\left(S_{\text{Res}}^{\text{WSPT}}\right) = 1 \cdot 1 + W\left(W^2 + W + 2\right) = W^3 + W^2 + 2W + 1.$$

No matter which scenario is chosen, in an optimal schedule $S_{\text{Sc}}^*$ the sequence of jobs is (2, 1), so that

$$C_2\left(S_{\text{Sc}}^*\right) = W + 1,\ C_1\left(S_{\text{Sc}}^*\right) = W^2 + W + 2,$$

and

$$Z\left(S_{\text{Sc}}^*\right) = W(W+1) + 1 \cdot \left(W^2 + W + 2\right) = 2W^2 + 2W + 2;$$

see Fig. 13.2c.

In either case, the ratio $Z\left(S_{\text{Sc}}^{\text{WSPT}}\right)/Z\left(S_{\text{Sc}}^*\right)$ goes to infinity, as $W \to \infty$. □

Define

$$\delta := \frac{t}{s}. \tag{13.26}$$

The lemma below analyzes the performance of Algorithm CMP_WSPT with respect to $\delta$ for the resumable scenario.

**Lemma 13.6** *For problem* $1|CMP(1), Res|\sum w_j C_j$ *the bound*

$$\frac{Z\left(S_{\text{Res}}^{\text{WSPT}}\right)}{Z\left(S_{\text{Res}}^*\right)} \le \delta \tag{13.27}$$

*holds.*

*Proof* For problem $1|CMP(1), Res|\sum w_j C_j$, let $\pi^* = (\pi^*(1), \pi^*(2), \ldots, \pi^*(n))$ be the sequence according to which the jobs are processed in an optimal schedule $S_{\text{Res}}^*$. Let $\pi^{\text{WSPT}} = (1, 2, \ldots, n)$ be the permutation in which the jobs are scheduled in the order of their numbering that defines the corresponding schedule $S_{\text{Res}}^{\text{WSPT}}$.

Assume that in schedule $S_{\text{Res}}^{\text{WSPT}}$ job $\ell$ is the crossover job and let $N_1$ and $N_2$ denote the sets of jobs that in schedule $S_{\text{Res}}^{\text{WSPT}}$ precede and follow job $\ell$, respectively, i.e., $N_1 = \{1, \ldots, \ell - 1\}$ and $N_2 = \{\ell + 1, \ldots, n\}$. For an optimal schedule $S_{\text{Res}}^*$, define the sets $N_q^*$ of jobs that complete before and after the CMP, respectively, $q \in \{1, 2\}$.

Using (13.24), we have that

$$Z\left(S_{\text{Res}}^{\text{WSPT}}\right) = Z_0\left(\pi^{\text{WSPT}}\right) + (t - s)w(N_2 \cup \{\ell\});$$
$$Z\left(S_{\text{Res}}^*\right) = Z_0\left(\pi^*\right) + (t - s)w\left(N_2^*\right).$$

Since $Z\left(S_{\text{Res}}^{\text{WSPT}}\right) \ge Z\left(S_{\text{Res}}^*\right)$ and $Z_0\left(\pi^{\text{WSPT}}\right) \le Z_0(\pi^*)$, it follows that $w(N_2 \cup \{\ell\}) \ge w\left(N_2^*\right)$.

For an optimal schedule $S_{\text{Res}}^*$, define sets

$$N' = \left\{j \in N | C_j\left(S^*\right) \le C_\ell\left(S_{\text{Res}}^{\text{WSPT}}\right)\right\};\ N'' = N \backslash N'.$$

The nature of the WSPT ordering is such that the total weight of jobs completed by time $\tau = C_\ell\left(S_{\text{Res}}^{\text{WSPT}}\right)$ in schedule $S_{\text{Res}}^{\text{WSPT}}$ is no less than the total weight of jobs completed by time $\tau$ in a feasible schedule $S$ associated with an arbitrary permutation $\pi$. In particular, for $\pi = \pi^*$ this means that

$$w(N_1 \cup \{\ell\}) \geq w\left(N'\right),$$

so that we deduce

$$w(N_2) = w(N) - w(N_1 \cup \{\ell\}) \leq w(N) - w\left(N'\right) = w\left(N''\right).$$

Notice that since job $\ell$ is the crossover job in schedule $S_{\text{Res}}^{\text{WSPT}}$, it follows that $C_\ell(S_{\text{WSPT}}) \geq t$, so that

$$N_2^* = \left\{j \in N | C_j\left(S^*\right) > t\right\} \supseteq \left\{j \in N | C_j\left(S^*\right) > C_\ell\left(S_{\text{Res}}^{\text{WSPT}}\right)\right\} = N'',$$

and therefore

$$w(N_2) \leq w\left(N''\right) \leq w\left(N_2^*\right).$$

This and (13.24) applied to schedule $S_{\text{Res}}^{\text{WSPT}}$ allow us to deduce an estimate of the value $Z\left(S_{\text{Res}}^{\text{WSPT}}\right)$ as follows

$$\begin{aligned} Z\left(S_{\text{Res}}^{\text{WSPT}}\right) &= Z_0\left(\pi^{\text{WSPT}}\right) + (t-s)w(N_2 \cup \{\ell\}) \\ &\leq Z_0\left(\pi^*\right) + (t-s)w(N_2) + (t-s)w_\ell \\ &\leq Z_0\left(\pi^*\right) + (t-s)w\left(N_2^*\right) + (t-s)w_\ell \\ &= Z\left(S_{\text{Res}}^*\right) + (t-s)w_\ell = Z\left(S_{\text{Res}}^*\right) + (\delta - 1)sw_\ell. \end{aligned}$$

Besides, since job $\ell$ is the crossover job in schedule $S_{\text{Res}}^{\text{WSPT}}$, it follows that $\sum_{i=1}^{\ell} p_i \geq s$, and therefore, we derive a lower bound

$$Z\left(S_{\text{Res}}^*\right) \geq Z_0\left(\pi^*\right) \geq Z_0\left(\pi^{\text{WSPT}}\right) = \sum_{j=1}^{n} w_j \sum_{i=1}^{j} p_i \geq w_\ell \sum_{i=1}^{\ell} p_i \geq sw_\ell.$$

Thus, for the resumable scenario, we obtain

$$Z\left(S_{\text{Res}}^{\text{WSPT}}\right) \leq Z\left(S_{\text{Res}}^*\right) + (\delta - 1)Z\left(S_{\text{Res}}^*\right) = \delta Z\left(S_{\text{Res}}^*\right),$$

so that (13.27) holds. □

For problem $1|CMP(1), Sc|\sum w_j C_j$, introduce the following minimization linear knapsack problem, which we call *Problem KP*.

$$\begin{aligned}\text{minimize } & \sum_{j=1}^{n} w_j y_j \\ \text{subject to } & \sum_{j=1}^{n} p_j y_j \geq \sum_{j=1}^{n} p_j - s \\ & y_j \in \{0, 1\}, \quad j = 1, \ldots, n.\end{aligned}$$

In Problem KP, we aim at finding the smallest possible total weight of the jobs processed after the CMP (for these jobs $y_j = 1$), provided that the other jobs will complete before the CMP (this is represented by the knapsack constraint). We can think of this problem as the problem $1\left|d_j = s\right| \sum w_j U_j$ of minimizing the weighted number of late jobs, provided that the jobs have a common due date $s$.

Suppose that a vector $\bar{\mathbf{y}} = (\bar{y}_1, \ldots, \bar{y}_n)$ delivers an optimal solution to Problem KP, and we have found its approximate solution associated with a Boolean vector $\mathbf{y}^H = (y_1^H, \ldots, y_n^H)$ such that

$$\sum_{j=1}^{n} w_j y_j^H \leq \rho \sum_{j=1}^{n} w_j \bar{y}_j, \tag{13.28}$$

where $\rho \geq 1$. Based on this approximate solution, we can define a heuristic schedule $S^H$ that is feasible for problem $1|CMP(1), N - Res| \sum w_j C_j$, and, therefore, for problem $1|CMP(1), Res| \sum w_j C_j$.

**Algorithm HKP**($\rho$)

**Step 1**. For Problem KP, find a Boolean vector $\mathbf{y}^H = (y_1^H, \ldots, y_n^H)$ such that (13.28) holds for some $\rho \geq 1$.

**Step 2**. Define the variables $x_j^H = 1 - y_j^H = 0$, $1 \leq j \leq n$, and define the sets of jobs $N_1^H = \left\{j \in N | x_j^H = 1\right\}$ and $N_2^H = N \backslash N_1^H$. Output schedule $S^H$, in which the block of jobs of set $N_1^H$, is processed starting from time zero, and the block of jobs of set $N_2^H$ is processed starting from time $t$; within each block, the jobs are processed in the order of their numbering.

We now analyze the worst-case performance of Algorithm HKP($\rho$).

**Theorem 13.10** *For problem $1|CMP(1), Sc| \sum w_j C_j$ with $Sc \in \{Res, N - Res\}$, let $S_{\mathrm{Sc}}^*$ be an optimal schedule, and $S^H$ be a schedule found by Algorithm HKP($\rho$). The bounds*

$$\frac{Z(S^H)}{Z\left(S_{\mathrm{N-Res}}^*\right)} \leq 1 + \rho \tag{13.29}$$

*and*

$$\frac{Z(S^H)}{Z\left(S_{\mathrm{Res}}^*\right)} \leq \rho + \frac{1}{\delta}, \tag{13.30}$$

*hold, where $\delta$ is defined by (13.26).*

*Proof* For a Boolean vector $\mathbf{x} = (x_1, x_2, \dots, x_n)$ define

$$w(\mathbf{x}) = \sum_{j=1}^{n} w_j (1 - x_j).$$

Let $\mathbf{x}^H = \left(x_1^H, x_2^H, \dots, x_n^H\right)$ be the vector found in Step 2 of Algorithm HKP($\rho$). For problem $1|CMP(1), Sc|\sum w_j C_j$ with $Sc \in \{Res, N - Res\}$, an optimal schedule $S_{\text{Sc}}^*$ can be associated with a Boolean vector $\mathbf{x}_{\text{Sc}}^* = \left(x_1^*, x_2^*, \dots, x_n^*\right)$ such that job $j \in N$ is completed before the CMP if and only if $x_j^* = 1$.

Since for vector $\mathbf{y}^H$ the knapsack constraint in Problem KP is satisfied, it follows that $p\left(N_1^H\right) \le s$, i.e., the obtained schedule $S^H$ is feasible. Due to (13.28), we deduce

$$w(\mathbf{x}^H) = \sum_{j=1}^{n} w_j y_j^H \le \rho \sum_{j=1}^{n} w_j \bar{y}_j.$$

Since vector $\bar{\mathbf{y}}$ delivers an optimal solution to Problem KP, for any Boolean vector $\mathbf{x}$ the inequality

$$\sum_{j=1}^{n} w_j \bar{y}_j \le \sum_{j=1}^{n} w_j \left(1 - x_j\right)$$

holds. In particular, for vector $\mathbf{x}_{\text{Sc}}^*$ we have that

$$\sum_{j=1}^{n} w_j \bar{y}_j \le \sum_{j=1}^{n} w_j \left(1 - x_j^*\right) = w\left(\mathbf{x}_{\text{Sc}}^*\right),$$

so that

$$w(\mathbf{x}^H) \le \rho w\left(\mathbf{x}_{\text{Sc}}^*\right). \tag{13.31}$$

It follows from (13.18) that

$$Z\left(S^H\right) = Z(\mathbf{x}^H) = f(\mathbf{x}^H) + tw(\mathbf{x}^H), \tag{13.32}$$

and due to (13.22), we obtain

$$\begin{aligned} Z(\mathbf{x}^H) &\le f(\mathbf{x}_{\text{N-Res}}^*) + sw(\mathbf{x}_{\text{N-Res}}^*) + tw(\mathbf{x}^H) \\ &\le f(\mathbf{x}_{\text{N-Res}}^*) + sw(\mathbf{x}_{\text{N-Res}}^*) + \rho tw(\mathbf{x}_{\text{N-Res}}^*) \\ &\le Z\left(\mathbf{x}_{\text{N-Res}}^*\right) + sw(\mathbf{x}_{\text{N-Res}}^*) + (\rho - 1)tw(\mathbf{x}_{\text{N-Res}}^*) \\ &\le Z\left(\mathbf{x}_{\text{N-Res}}^*\right) + \rho tw(\mathbf{x}_{\text{N-Res}}^*) \le (1 + \rho)Z\left(\mathbf{x}_{\text{N-Res}}^*\right), \end{aligned}$$

where the last inequality due to

$$Z\left(\mathbf{x}^*_{\mathrm{Sc}}\right) = Z\left(S^*_{\mathrm{Sc}}\right) \geq \sum_{j=1}^{n} w_j\left(1 - x^*_j\right) \sum_{i=1}^{j}\left(p_i\left(1 - x^*_i\right) + t\right) \geq tw(\mathbf{x}^*_{\mathrm{Sc}}). \tag{13.33}$$

Thus, for the non-resumable scenario (13.29) holds.

In order to analyze the resumable scenario, we apply Lemma 13.2 with $\mathbf{x} = \mathbf{x}^H$ to derive

$$\begin{aligned} f(\mathbf{x}^H) &= \sum_{1\leq i\leq j\leq n} p_i w_j x^H_i x^H_j + \sum_{1\leq i\leq j\leq n} p_i w_j (1 - x^H_i)(1 - x^H_j) \\ &\leq \sum_{j=1}^{n} w_j \sum_{i=1}^{j} p_i = Z_0\left(\pi^{\mathrm{WSPT}}\right) \leq Z_0\left(\pi^*\right), \end{aligned}$$

where $\pi^{\mathrm{WSPT}} = (1, 2, \ldots, n)$ and $\pi^*$ is an optimal permutation according to which the jobs are processed in schedule $S^*_{\mathrm{Res}}$. We deduce from and (13.31) and (13.32) that

$$\begin{aligned} Z\left(S^H\right) &= Z(\mathbf{x}^H) = f(\mathbf{x}^H) + tw(\mathbf{x}^H) \leq Z_0\left(\pi^*\right) + tw(\mathbf{x}^H) \leq Z_0\left(\pi^*\right) + \rho tw(\mathbf{x}^*_{\mathrm{Res}}) \\ &= Z_0\left(\pi^*\right) + (t - s)w(\mathbf{x}^*_{\mathrm{Res}}) + (\rho - 1)tw(\mathbf{x}^*_{\mathrm{Res}}) + sw(\mathbf{x}^*_{\mathrm{Res}}). \end{aligned}$$

Using (13.24), we further obtain

$$Z\left(S^H\right) \leq Z\left(S^*_{\mathrm{Res}}\right) + (\rho - 1)tw(\mathbf{x}^*_{\mathrm{Res}}) + sw\left(\mathbf{x}^*_{\mathrm{Res}}\right),$$

and applying (13.26) and (13.33), we finally deduce

$$Z\left(S^H\right) \leq Z\left(S^*_{\mathrm{Res}}\right) + \left(\rho - 1 + \frac{1}{\delta}\right)tw(\mathbf{x}^*_{\mathrm{Res}}) \leq \left(\rho + \frac{1}{\delta}\right)Z\left(S^*_{\mathrm{Res}}\right),$$

i.e., (13.30) holds as required. □

Notice that Problem KP admits a fully polynomial-time approximation scheme (FPTAS) that requires $O(n/\varepsilon)$ time for any positive $\varepsilon$; see Sect. 4.2 for a discussion of the FPTAS for the linear knapsack problem in the maximization form. Thus, in Step 1 of Algorithm HKP($\rho$) we may apply such an FPTAS, which guarantees that $\rho = 1 + \varepsilon$. This results in the following statement on constant ratio algorithms for problems problem $1|CMP(1), Sc|\sum w_j C_j$ with $Sc \in \{Res, N - Res\}$.

**Theorem 13.11** *For problem* $1|CMP(1), Sc|\sum w_j C_j$ *with* $Sc \in \{Res, N - Res\}$, *let* $S^*_{\mathrm{Sc}}$ *be an optimal schedule, and* $S^H$ *be a schedule found by Algorithm HKP*($\rho$) *with* $\rho = 1 + \varepsilon$. *Additionally, let* $S^{\mathrm{WSPT}}_{\mathrm{Res}}$ *be an a schedule found by Algorithm CMP_WSPT for the problem under the resumable scenario. The following bounds hold:*

$$\frac{Z(S^H)}{Z(S^*_{\text{N-Res}})} \le 2 + \varepsilon \tag{13.34}$$

*and*

$$\frac{\min\{Z(S^{\text{WSPT}}_{\text{Res}}), Z(S^H)\}}{Z(S^*_{\text{Res}})} \le \varepsilon + \delta_0, \tag{13.35}$$

*where* $\delta_0 = \frac{1}{2}\sqrt{5} + \frac{1}{2} = 1.618\ldots$ *is the positive root of the equation* $\delta = 1 + \frac{1}{\delta}$.

*Proof* The inequality (13.34) follows immediately from (13.29) applied with $\rho = 1 + \varepsilon$.

For the resumable scenario, Lemma 13.6 and Theorem 13.10 (applied with $\rho = 1 + \varepsilon$) guarantee that

$$\min\{Z(S^{\text{WSPT}}_{\text{Res}}), Z(S^H)\} \le \min\left\{\delta, 1 + \varepsilon + \frac{1}{\delta}\right\} Z(S^*_{\text{Res}}) \le (\varepsilon + \delta_0) Z(S^*_{\text{Res}}),$$

as required. □

Notice that finding schedules that deliver the bounds stated in Theorem 13.11 requires $O(n \log n + n/\varepsilon)$ time.

## 13.4.2 Approximation Schemes

Consider a problem of minimizing a function $Z(\mathbf{x})$, where $\mathbf{x}$ is a collection of decision variables, e.g., a Boolean vector or a schedule. Recall that if $\mathbf{x}^*$ is an optimal solution such that $Z(\mathbf{x}^*) > 0$, an FPTAS delivers a feasible solution $\mathbf{x}^\varepsilon$ such that $Z(\mathbf{x}^\varepsilon) \le (1 + \varepsilon) Z(\mathbf{x}^*)$.

Each problem $1|CMP(1), Sc|\sum w_j C_j$ with $Sc \in \{Res, N - Res\}$ is known to admit an FPTAS; however, the corresponding schemes are rather technical and their detailed description is beyond the scope of this book. Below, we only briefly present the ideas based on which an FPTAS can be designed.

As follows from (13.20), problem $1|CMP(1), N - Res|\sum w_j C_j$ reduces to a symmetric quadratic knapsack problem, denoted by Problem SQK in Sect. 4.4. According to Theorem 4.13, Problem SQK admits an FPTAS, provided that is admits a constant ratio approximation algorithm. We know from Theorem 13.11 that such an algorithm exists and requires $O(n \log n + n/\varepsilon)$ time. Thus, Theorem 4.13 is applicable and the following statement holds.

**Theorem 13.12** *Problem* $1|CMP(1), N - Res|\sum w_j C_j$ *admits an FPTAS that requires* $O(n^4/\varepsilon^2)$ *time. In the unweighted case, the running time becomes* $O(n^3/\varepsilon^2)$.

We know that a schedule for problem $1|CMP(1), Res|\sum w_j C_j$ can be obtained by inserting one of the jobs as the crossover job into a schedule found for problem

$1|CMP(1), N-Res|\sum w_jC_j$ with the set of the remaining jobs. Thus, an FPTAS from Theorem 13.12 can be used as a subroutine for each choice of the crossover job. Besides, extra care should be taken regarding the total weight of the jobs scheduled after the CMP, which additionally requires $O(n/\varepsilon)$ time for each choice of the crossover job. Thus, the following statement holds.

**Theorem 13.13** *Problem* $1|CMP(1), Res|\sum w_jC_j$ *admits an FPTAS that requires* $O(n^6/\varepsilon^3)$ *time.*

See Sect. 13.5 for references and discussions.

## 13.5 Bibliographic Notes

This section provides a brief bibliographic review of the relevant results. Additional information can be found in surveys by Lee (2004) and Ma et al. (2010).

### *13.5.1 Minimizing Makespan*

Theorem 13.4 is proved in Breit et al. (2003).

The studies on problem $1|CMP(period), N-Res|C_{\max}$, have been initiated in Ji et al. (2007), where Algorithm LPT_Period is introduced and Theorem 13.6 is proved. An alternative 2-approximation algorithms for problem $1|CMP(period)$, $N-Res|C_{\max}$, is given in Yu et al. (2014), where a detailed analysis of the number of groups in schedules created by various approximation algorithms is presented.

Analysis of approximation algorithms in Ji et al. (2007) and Yu et al. (2014) is based on a useful link between problem $1|CMP(period), N-Res|C_{\max}$, and a well-studied problem of combinatorial optimization, known as the *bin-packing*. The bin-packing problem is NP-hard, and the main focus of its studies in on design and analysis of approximation algorithms. In terms of bin-packing, an instance of problem $1|CMP(period), N-Res|C_{\max}$, can be interpreted as follows. The jobs correspond to items; the size of item $j \in N$ is equal to the processing time $p_j$ of job $j$. The items are to be packed into bins of size $T$. In the bin-packing problem, it is required to find the smallest number of bins, which we denote by $b^*$. For schedule $S^*$ that is optimal for problem $1|CMP(period), N-Res|C_{\max}$, the number of groups is denoted by $b^*$, and it is equal to the number of bins used in an optimal solution to the associated bin-packing problem. Although the objective in problem $1|CMP(period), N-Res|C_{\max}$, and in the associated bin-packing problem is different, similar decisions should be taken to solve these problems, so that various algorithmic ideas known in the bin-packing studies can be employed to handle problem $1|CMP(period), N-Res|C_{\max}$. In particular, Algorithm LPT_Period can be seen as a scheduling adaptation of a bin-packing algorithm that implements the strategy known as *First Fit Decreasing*; see, e.g., Simchi-Levi (1994).

### 13.5.2 Minimizing Weighted Total Flow Time: Complexity

The reformulation of problem $1|CMP(1), N-Res|\sum w_j C_j$ in terms of a symmetric quadratic knapsack problem (13.20) is presented in Kellerer and Strusevich (2010); see also the survey Kellerer and Strusevich (2012) and its updated version Kellerer and Strusevich (2016). The presented proof of Theorem 13.7 is given by Lee and Liman (1992). Their proof is simpler than a previously known proof by Adiri et al. (1989).

Algorithm NResDP is an adaptation of a dynamic programming algorithm for solving a symmetric quadratic knapsack problem given in Kellerer and Strusevich (2010).

Lee (1996) studies problem $1|CMP(1), Res|\sum w_j C_j$. He proves the formulation of the objective function in the form (13.24) and mentions the fact that problem $1|CMP(1), Res|\sum C_j$ is solvable by ordering the jobs in the SPT order.

Theorem 13.9 on the complexity of problem $1|CMP(1), Res|\sum w_j C_j$, is proved by Lee (1996). For problem $1|CMP(1), Res|\sum w_j C_j$, Lee (1996) gives a DP algorithm that requires $O(nsp_{\max})$ time, where $p_{\max} = \max\{p_j | j \in N\}$. The approach to solving problem $1|CMP(1), Res|\sum w_j C_j$, by inserting a chosen crossover job into a schedule for the non-resumable scenario, that results into an algorithm that requires $O(n^2 s)$ time, is developed in Kellerer and Strusevich (2010).

### 13.5.3 Minimizing Weighted Total Flow Time: Approximation

The first approximation algorithms for the problems of the range under consideration have been developed for problem $1|CMP(1), N-Res|\sum C_j$, with the unweighted objective function. Lee and Liman (1992) by refining the analysis by Adiri et al. (1989) demonstrate that an algorithm that finds schedule $S_{SPT}$ in which the jobs are sequenced in the SPT order is a (9/7)-approximation algorithm. An algorithm with a worst-case performance ratio of 20/17 is given by Sadfi (2005). A further improvement is done by Breit (2007). A polynomial-time approximation scheme (PTAS) for the problem is presented in He et al. (2006).

Lemma 13.5 has been proved by several authors, e.g., by Lee (1996) for the resumable scenario and by Kacem and Chu (2008) for the non-resumable scenario; see also Megow and Verschae (2009). For the non-resumable scenario, Kacem and Chu (2008) provide a modified algorithm that outputs a schedule $S_{\text{MWSPT}}$. They show that under some additional assumptions $\min\{Z(S_{\text{MWSPT}}), Z(S_{\text{WSPT}})\} \leq 3Z(S^*)$, but in general, the performance of the modified algorithm remains arbitrary bad. For the resumable scenario, Lee (1996) provides an algorithm that combines the WSPT heuristic and an attempt to schedule the jobs with large weights before time $s$. It is proved that for schedule $S_{\text{Comb}}$ produced by this combined algorithm the inequality $Z(S_{\text{Comb}}) \leq 2Z(S^*)$ holds, provided that $p_j = w_j, j \in N$; but in general, the performance of the combined algorithm remains arbitrary bad.

For problem $1|CMP(1), Res|\sum w_jC_j$, Wang et al. (2005) design a 2-approximation algorithm that requires $O(n^2)$ time. Kellerer et al. (2009) demonstrate that the existence of a $\rho$-approximation algorithm for problem $1|CMP(1), Res|\sum w_jC_j$ implies the existence of a $(2\rho)$-approximation algorithm for problem $1|CMP(1), N-Res|\sum w_jC_j$ with the same set of jobs.

Lemma 13.6 is proved in Megow and Verschae (2009). Theorems 13.10 and 13.11 for the non-resumable case are given in Kellerer et al. (2009), while their versions for the resumable scenario are proved in Epstein et al. (2012). For problem $1|CMP(1), N-Res|\sum w_jC_j$, a 2-approximation algorithm that requires $O(n^2)$ time is due to Kacem (2008).

An FPTAS for Problem KP that is needed for Algorithm HKP($\rho$) applied with $\rho = 1+\varepsilon$ can be found in Kellerer et al. (2004); see also Sect. 4.2.

Theorems 13.12 and 13.13 are proved in Kellerer and Strusevich (2010). This paper also presents detailed descriptions of the corresponding approximation schemes. Problem $1|CMP(1), N-Res|\sum w_jC_j$, is known to admit a faster FPTAS that requires $O(n^3/\varepsilon^2)$ time; see Epstein et al. (2012) and Kacem and Mahjoub (2009).

## References

Adiri I, Bruno J, Frostig E, Rinnooy Kan AHG (1989) Single machine flow-time scheduling with a single breakdown. Acta Inform 26:679–696

Breit J (2007) Improved approximation for non-preemptive single machine flow-time scheduling with an availability constraint. Eur J Oper Res 183:516–524

Breit J, Schmidt G, Strusevich VA (2003) Non-preemptive two-machine open shop scheduling with non-availability constraints. Math Meth Oper Res 57:217–234

Epstein L, Levin A, Marchetti-Spaccamela A, Megow N, Mestre J, Skutella M, Stougie L (2012) Universal sequencing on an unrelaible machine. SIAM J Comput 41:565–586

He Y, Zhong W, Gu H (2006) Improved algorithms for two single machine scheduling problems. Theor Comput Sci 363:257–265

Ji M, He Y, Cheng TCE (2007) Single-machine scheduling with periodic maintenance to minimize makespan. Comp Oper Res 34:1764–1770

Kacem I (2008) Approximation algorithm for the weighted flow-time minimization on a single machine with a fixed non-availability interval. Comput Industr Eng 54:401–410

Kacem I, Chu C (2008) Worst-case analysis of the WSPT and MWSPT rules for single machine scheduling with one planned setup period. Eur J Oper Res 187:1080–1089

Kacem I, Mahjoub AR (2009). Fully polynomial time approximation scheme for the weighted flow-time minimization on a single machine with a fixed non-availability interval. Comput Industr Eng 56:1708–1712; see also Kacem I, Mahjoub AR (2011) Erratum, Comput Industr Eng 61:1351

Kellerer H, Kubzin MA, Strusevich VA (2009) Two simple constant ratio approximation algorithms for minimizing the total weighted completion time on a single machine with a fixed non-availability interval. Eur J Oper Res 199:111–116

Kellerer H, Pferschy U, Pisinger D (2004) Knapsack problems. Springer, Berlin

Kellerer H, Strusevich VA (2010) Fully polynomial approximation schemes for a symmetric quadratic knapsack problem and its scheduling applications. Algorithmica 57:769–795

Kellerer H, Strusevich VA (2012) The symmetric quadratic knapsack problem: approximation and scheduling applications. 4OR - Quart J. Oper Res 10:111–161

Kellerer H, Strusevich VA (2016) Optimizing the Half-Product and related quadratic Boolean functions: approximation and scheduling applications. Ann Oper Res 240:39–94

Lee C-Y (1996) Machine scheduling with an availability constraint. J Glob Opti 9:395–416

Lee C-Y (2004) Machine scheduling with availability constraints. In Leung JY-T(ed) Handbook of scheduling: algorithms, models and performance analysis. Chapman & Hall/CRC, London, pp. 22-1–22-13

Lee C-Y, Liman SD (1992) Single machine flow time scheduling with scheduled maintenance. Acta Inform 29:375–382

Ma Y, Chu C, Zuo C (2010) A survey of scheduling with deterministic machine availability constraints. Comput Industr Eng 58:199–211

Megow N, Verschae J (2009) Short note on scheduling on a single machine with one non-availability period. Matheon, Preprint 557

Sadfi C, Penz B, Rapin C, Błaž ewicz J, Formanowicz P (2005) An improved approximation algorithm for the single machine total completion time scheduling problem with availability constraints. Eur J Oper Res 161: 3–10

Simchi-Levi D (1994) New worst-case results for the bin packing problem. Naval Res Log 41: 579–585

Wang G, Sun H, Chu C (2005) Preemptive scheduling with availability constraints to minimize total weighted completion times. Ann Oper Res 133:183–192

Yu X, Zhang Y, Steiner G (2014) Single-machine scheduling with periodic maintenance to minimize makespan revisited. J Sched 17:263–270

# Chapter 14
# Scheduling with Flexible Maintenance

In most problems considered in Chap. 13, a single machine is subject to a single compulsory maintenance, and the start time $s$ of the compulsory maintenance period (CMP) and its duration $\Delta$ are fixed. In this chapter, we also consider single machine scheduling problems with maintenance activities; however, now, the location of a maintenance period (MP) is less restrictive, and in fact, it is a decision variable so that its value may affect the objective function. Unless stated otherwise, in scheduling problems discussed in this chapter, the jobs of set $N = \{1, 2, \ldots, n\}$ are to be processed on a single machine. The processing of job $j \in N$ takes $p_j$ time units. There is a weight $w_j$ associated with job $j$, which indicates its relative importance. All values $p_j$ and $w_j$ are positive integers. The machine processes at most one job at a time. The completion time of job $j \in N$ in a feasible schedule $S$ is denoted by $C_j(S)$, or shortly $C_j$, if it is clear which schedule is referred to.

In one of the simplest models, a single *flexible* machine maintenance interval can be viewed as a non-availability period of length $\Delta$ that may start at any time $\tau$ that does not exceed a given deadline $D_{MP}$. For an objective function $\Phi \in \left\{C_{\max}, \sum C_j, \sum w_j C_j\right\}$, we denote problems of this type either by $1|\tau \leq D_{MP}, \Delta|\Phi$, provided that the duration of an MP is constant and is equal to $\Delta$, or by $1|\tau \leq D_{MP}, \Delta(\tau)|\Phi$, provided that the duration of an MP is a non-decreasing function of its start time $\tau$.

An extension of the model with the maintenance start-time deadline is the model with the due window, where it is assumed that the MP must start and finish within a given time window $[s, t]$. Similar to the above, we denote problems of this type either by $1|MP \in [s, t], \Delta|\Phi$, provided that the duration of an MP is constant and is equal to $\Delta$, or by $1|MP \in [s, t], \Delta(\tau)|\Phi$, provided that the duration of an MP is a non-decreasing function of its start time $\tau$, where $\tau \geq s$ and $\tau + \Delta(\tau) \leq t$.

For those problems which are NP-hard, we present the results on design and analysis of approximation algorithms and schemes. Let $S^*$ denote a schedule that is optimal for a scheduling problem of minimizing a function $\Phi(S)$, i.e., $\Phi(S^*) \leq \Phi(S)$ for all feasible schedules $S$. Recall that a polynomial-time algorithm that finds a feasible schedule $S^H$ such that the inequality $\Phi(S^H)/\Phi(S^*) \leq \rho$ holds for all instances of the problem is called a $\rho$-*approximation* algorithm and $\rho \geq 1$ is called a *worst-case ratio bound.* A fully polynomial-time approximation scheme (FPTAS) is a family of $\rho$-approximation algorithms such that for any positive $\varepsilon > 0$ (i) $\rho = 1+\varepsilon$ and (ii) the running time depends polynomially on the length of the problem's input and $1/\varepsilon$.

The structure of this chapter is as follows. In Sect. 14.1, we consider the complexity and approximability issues of the problem of minimizing the makespan, the total flow time, and the sum of the weighted completion times, provided that a single flexible maintenance period (either of a constant duration or of a start-time-dependent duration) must start before a given due date. Section 14.2 considers the model with an MP that must be scheduled within a given window, including its extension to periodic maintenance.

## 14.1 Flexible Maintenance: Start-Time Deadline

We start with single machine models in which a single maintenance has to be completed before a given deadline.

Recall that in the case of a compulsory maintenance periods, in Chap. 13, we consider a possible resumable scenario of processing the jobs that might be affected by the introduction of the MP. Below, we show that for the flexible maintenance, there is no advantage in using the resumable scenario.

**Lemma 14.1** *For problem $1|\tau \leq D_{MP}, \Delta(\tau)|\Phi$, where $\Phi \in \left\{C_{\max}, \sum C_j, \sum w_j C_j\right\}$, there exists an optimal schedule in which no job is interrupted and the MP starts at the completion time of some job.*

*Proof* For problem $1|\tau \leq D_{MP}, \Delta(\tau)|\Phi$, consider a feasible schedule $S$, in which the jobs are processed according to the sequence $\pi = (\pi(1), \ldots, \pi(n))$ and the MP starts at time $\tau$. Let $\pi(\ell)$ be the crossover job that starts at time $\sum_{j=1}^{\ell-1} p_{\pi(j)}$, but cannot be completed before time $\tau$. The transformation of schedule $S$ is illustrated in Fig. 14.1, where it is for simplicity assumed that the duration of the MP is constant and is equal to $\Delta$.

Under the resumable scenario, job $\pi(\ell)$ is processed for $x$ time units before the MP and resumes at time $\tau + \Delta(\tau)$, so that the duration of its processing after the MP is $p_{\pi(\ell)} - x$ (see Fig. 14.1a). Transform schedule $S$ into a schedule $S'$ by swapping the MP and the time interval of length $x$ in which the part of the crossover job is processed before the MP. In schedule $S'$, the MP starts at the completion time of job $\pi(\ell - 1)$ and completes at time $\tau - x + \Delta(\tau - x)$, and is now followed by the whole job $\pi(\ell)$; i.e., $S'$ is feasible for the original problem $1|\tau \leq D_{MP}, \Delta(\tau)|\Phi$ (see Fig. 14.1b). In schedule $S'$, the duration of the MP is no longer than it is in schedule $S$, and the completion time of each job $\pi(\ell), \pi(\ell+1), \ldots, \pi(n)$ in $S'$ does not exceed that in $S$, i.e., $\Phi(S') \leq \Phi(S)$. $\square$

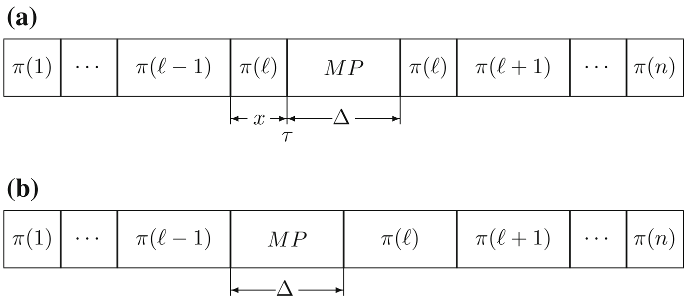


**Fig. 14.1** **a** Schedule $S$; **b** schedule $S'$

Further in this chapter, unless stated otherwise, we only consider schedules in which the MP starts at the completion time of some job. Similar to the non-resumable scenario for problems with a single CMP considered in Chap. 13, a feasible schedule $S$ is defined by a partition of set $N$ into two groups $N^{[1]}$ and $N^{[2]}$ and the sequences $\pi^{[1]}$ and $\pi^{[2]}$ of these jobs, respectively, such that

**(i)** the jobs of the first $N^{[1]}$ are processed from time zero as a block, without intermediate idle time, follow sequence $\pi^{[1]}$, and the MP starts at time $p\left(N^{[1]}\right)$;
**(ii)** the jobs of the second group $N^{[2]}$ are processed after the MP as a block, without intermediate idle time time, and follow sequence $\pi^{[2]}$.

First, observe that problem $1|\tau \le D_{MP}, \Delta(\tau)|C_{\max}$ is trivial, since it is optimal to start the MP at time zero and create a schedule in which the MP is followed by an arbitrary sequence of all jobs. The makespan of such a schedule is $p(N) + \Delta(0)$, which is obviously the smallest possible. If the MP duration is constant and equal to $\Delta$, then optimal makespan becomes $p(N) + \Delta$, so that the MP can start either at time zero or after a suitable number of jobs. In any case, the following statement holds.

**Theorem 14.1** *Problem* $1|\tau \le D_{MP}, \Delta(\tau)|C_{\max}$ *is solvable is* $O(n)$ *time.*

For minimizing the total flow time $F(S) = \sum C_j$, it is useful to renumber the jobs in the SPT order given by

$$p_1 \le p_2 \le p_3 \le \cdots \le p_n, \tag{14.1}$$

since, as shown in Theorem 2.2, permutation $(1, 2, \ldots, n)$ delivers an optimal solution to problem $1||\sum C_j$, with a continuously available machine.

For problem $1|\tau \le D_{MP}, \Delta(\tau)|\sum C_j$, it is easy to verify that there exists an optimal schedule, the jobs of the first group and the jobs of the second group are processed in the SPT order. Moreover, a stronger property holds: Below, we prove that there exists an optimal schedule, in which the jobs are processed in the SPT order.

**Lemma 14.2** *For any non-decreasing function $\Delta(\tau)$ of the MP duration, problem $1|\tau \leq D_{MP}, \Delta(\tau)|C_{\max}$ admits an optimal schedule $S^*$ in which the jobs are sequenced in the SPT order.*

*Proof* Assume that the jobs are numbered in accordance with (14.1), and let $\pi_0 = (1, 2, \dots, n)$ be the SPT sequence of jobs.

For problem $1| |\sum C_j$ with continuously available machines, for a schedule $S$ associated with an arbitrary permutation $\pi$, the objective function can be written as

$$F_0(\pi) = \sum_{j=1}^{n}\sum_{i=1}^{j} p_{\pi(i)}. \tag{14.2}$$

For problem $1|\tau \leq D_{MP}, \Delta(\tau)|\sum C_j$, introduce two feasible schedules $S_k$ and $S'_k$ such that (i) in schedule $S_k$, the jobs are processed in accordance with permutation $\pi_0$; (ii) in schedule $S'_k$, the jobs are processed in accordance with some permutation permutation $\pi = (\pi(1), \pi(2), \dots, \pi(n))$; and (iii) in both schedules, the MP starts at the completion time of the first $k$ jobs. Notice that due to the SPT ordering, if a schedule $S'_k$ is feasible, then schedule $S_k$ is also feasible, but not vice versa. It follows that

$$\begin{aligned} F(S_k) &= \sum_{j=1}^{n} C_j(S_k) = \sum_{j=1}^{k}\sum_{i=1}^{j} p_i + \sum_{j=k+1}^{n}\sum_{i=1}^{j}\left(p_i + \Delta\left(\sum_{i=1}^{k} p_i\right)\right) \\ &= F_0(\pi_0) + (n-k)\Delta\left(\sum_{i=1}^{k} p_i\right); \\ F\left(S'_k\right) &= \sum_{j=1}^{n}\sum_{i=1}^{j} p_{\pi(i)} + (n-k)\Delta\left(\sum_{i=1}^{k} p_{\pi(i)}\right) = F_0(\pi) + (n-k)\Delta\left(\sum_{i=1}^{k} p_{\pi(i)}\right). \end{aligned} \tag{14.3}$$

Due to Theorem 2.2, we know that $F_0(\pi_0) \leq F_0(\pi)$. Besides, due to the SPT ordering, the expression $\sum_{i=1}^{k} p_i$ is the smallest sum of arbitrary selected $k$ jobs of set $N$, and since $\Delta(\tau)$ is non-decreasing, it follows that $\Delta\left(\sum_{i=1}^{k} p_i\right) \leq \Delta\left(\sum_{i=1}^{k} p_{\pi(i)}\right)$. As a result, we deduce that $F(S_k) \leq F\left(S'_k\right)$, which proves the lemma. □

Lemma 14.2 implies that in order to find an optimal schedule, we need to determine the position of the MP within the SPT sequence. Such a position can be found by comparing values $F(S_k)$ defined by (14.3).

**Algorithm MP1TFT_DD**

**Step 1**. For problem $1|\tau \leq D_{MP}, \Delta(\tau)|\sum C_j$, renumber the jobs in accordance with (14.1) and let $\pi_0 = (1, 2, \dots, n)$. Compute

$$F_0(\pi_0) = \sum_{j=1}^{n}\sum_{i=1}^{j} p_i = \sum_{j=1}^{n}(n-j+1)p_j. \tag{14.4}$$

Compute $F(S_0) := F_0(\pi_0) + n\Delta(0)$.

**Step 2**. Considering the jobs in accordance with permutation $\pi_0$, find the job $\ell$ such that

$$\sum_{j=1}^{\ell-1} p_j \le D_{MP}, \quad \sum_{j=1}^{\ell} p_j > D_{MP}.$$

**Step 3**. Compute the values $\tau_k := \sum_{j=1}^{k} p_j$, $k = 1, \dots, \ell - 1$.

**Step 4**. For each $k$ from 1 to $\ell - 1$, compute $F(S_k) := F_0(\pi_0) + (n-k)\Delta(\tau_k)$.

**Step 5**. Determine $k^*$, $1 \le k^* \le \ell - 1$, such that $F(S_{k^*}) := \min\{F(S_k)|0 \le k \le \ell - 1\}$. Output an optimal schedule $S^* = S_{k^*}$, in which $N^{[1]} = \{1, \dots, k\}$, $N^{[2]} = \{k+1, \dots, n\}$ and the MP starts at time $\tau_k = \sum_{j=1}^{k} p_j$, where $k = k^*$.

Algorithm MP1TFT_DD requires $O(n \log n)$ time due to the SPT ordering; the other steps of the algorithm can be implemented in $O(n)$ time, under the assumption that for any $\tau$, the value of $\Delta(\tau)$ can be computed in constant time. The algorithm determines an optimal way of inserting the MP into the SPT sequence of jobs. Thus, the following statement holds.

**Theorem 14.2** *Problem* $1|\tau \le D_{MP}, \Delta(\tau)|\sum C_j$ *is solvable in* $O(n \log n)$ *time by Algorithm MP1TFT_DD.*

Notice that in the case of problem $1|\tau \le D_{MP}, \Delta|\sum C_j$, with a constant duration of the MP, Algorithm MP1TFT_DD can be slightly simplified, since it is optimal to insert the MP after job $\ell - 1$ found in Step 2. Indeed, comparing $F(S_{\ell-1}) = F_0(\pi_0) + (n - \ell + 1)\Delta$ and $F(S_k) = F_0(\pi_0) + (n-k)\Delta$, we obtain that

$$F\left(S^*\right) = F(S_{\ell-1}) \le \min\{F(S_k)|0 \le k \le \ell - 1\}.$$

Now, we consider problem $1|\tau \le D_{MP}, \Delta|\sum w_j C_j$ to minimize the weighted total flow time $Z(S) = \sum w_j C_j(S)$. Introduce the *associated* problem $1|CMP(1), Res|\sum w_j C_j$ studied in Sect. 13.3, in which the fixed CMP of length $\Delta$ is defined by $[s, t] = [D_{MP}, D_{MP} + \Delta]$, while the processing times of the jobs and their weights remain equal to $p_j$ and $w_j$, respectively, and the resumable scenario applies.

**Lemma 14.3** *Problem* $1|\tau \le D_{MP}, \Delta|\sum w_j C_j$ *and the associated problem* $1|CMP(1), Res|\sum w_j C_j$ *are equivalent.*

*Proof* For the associated problem $1|CMP(1), Res|\sum w_j C_j$, consider a schedule $S_{\text{Res}}$ in which the jobs are processed according to the sequence $\pi = (\pi(1), \dots, \pi(n))$, with $\pi(\ell)$ as the crossover job. Applying the transformation described in the proof of Lemma 14.1, we obtain schedule $S$ in which the MP now starts at time $\tau = \sum_{i=1}^{\ell-1} p_{\pi(i)}$ and is followed by the whole job $\pi(\ell)$. Schedule $S$ is feasible for the original problem

$1|\tau \le D_{MP}, \Delta|\sum w_jC_j$, and in $S$, the completion times of all jobs remain as in schedule $S_{\text{Res}}$, so that $Z(S) = Z(S_{\text{Res}})$ (see Fig. 14.1 for illustration).

Now, for problem $1|\tau \le D_{MP}, \Delta|\sum w_jC_j$, consider a feasible schedule $S$ in which the jobs are processed according to the sequence $\pi = (\pi(1), \ldots, \pi(n))$. Due to Lemma 14.1, we may assume that the MP of length $\Delta$ starts exactly when the first $\ell - 1$ jobs are completed, i.e., at time $\tau = \sum_{j=1}^{\ell-1} p_{\pi(j)} \le D_{MP}$. Compute $y := D_{MP} - \tau$. We only need to consider the case that $y > 0$. The processing of job $\pi(\ell)$ that follows the MP can be seen as consisting of two parts: part 1, of $y$ time units, that starts at time $\tau + \Delta$ and completes at time $\tau + \Delta + y = D_{MP} + \Delta$, and part 2 that starts at time $D_{MP} + \Delta$ (see Fig. 14.2).

Interchange part 1 of the processing of job $\pi(\ell)$ with the MP. The obtained schedule $S_{\text{Res}}$ is feasible for the associated problem $1|CMP(1), Res|\sum w_jC_j$, with the MP that starts at time $\tau + y = D_{MP}$ and with job $\pi(\ell)$ as the crossover job. In schedule $S_{\text{Res}}$, the completion times of all jobs remain as in schedule $S$, so that $Z(S_{\text{Res}}) = Z(S)$. $\square$

Lemma 14.3 implies that any schedule feasible for the associated problem $1|CMP(1), Res|\sum w_jC_j$, can be transformed into a schedule for the original problem $1|\tau \le D_{MP}, \Delta|\sum w_jC_j$, and vice versa, without any change in the objective function value. Due to Lemma 14.3, we may use Theorems 13.9 and 13.13 to deduce the complexity and approximability statuses of problem $1|\tau \le D_{MP}, \Delta|\sum w_jC_j$.

**Theorem 14.3** *Problem $1|\tau \le D_{MP}, \Delta|\sum w_jC_j$ is NP-hard in the ordinary sense, even if $p_j = w_j$, $j \in N$, and admits an FPTAS that requires $O(n^6/\varepsilon^3)$ time.*

In the rest of this subsection, we consider problem $1|\tau \le D_{MP}, \Delta(\tau)|\sum w_jC_j$, in which a single MP has to be introduced into a schedule so that (i) its start time $\tau$ does not exceed a given deadline $D_{MP}$ and (ii) its duration $\Delta(\tau)$ is a non-decreasing concave function of $\tau$. It is required to find a schedule $S^*$ that minimizes the sum of the weighted completion times, i.e., function $Z(S) = \sum w_jC_j(S)$.

Each feasible schedule $S$ is associated with a partition of set $N$ into two groups $N^{[1]}$ and $N^{[2]}$, where the jobs of group $N^{[q]}$ are scheduled in accordance with a permutation $\pi^{[q]}$, $q \in \{1, 2\}$. We only consider schedules, in which the MP starts exactly at the completion time of the block of jobs $N^{[1]}$. Similar to Sect. 13.2, the jobs are assumed to be numbered according to the WSPT rule, i.e.,

$$\frac{p_1}{w_1} \le \frac{p_2}{w_2} \le \cdots \le \frac{p_n}{w_n}. \tag{14.5}$$

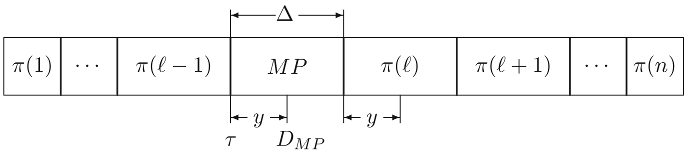


**Fig. 14.2** Schedule $S$

Problem $1|\tau \leq D_{MP}, \Delta(\tau)|\sum w_j C_j$ is NP-hard (see Theorem 14.3 for the complexity of its special case with the MP of a constant duration). Consider the following approximation algorithm.

**Algorithm MP1WTFT_DD**

**Step 1**. Scanning the jobs in the order of their numbering given by (14.5), determine the job $k_1 \in N$ such that

$$\sum_{j=1}^{k_1-1} p_j \leq D_{MP} < \sum_{j=1}^{k_1} p_j.$$

**Step 2**. Define a schedule $S^0$ in which the first group is empty, the second group of jobs consists of all jobs and the MP starts at time zero.

**Step 3**. For each $k$, $1 \leq k \leq k_1 - 1$, define a schedule $S^k$ in which the first group and the second group of jobs are given by $\{1, \dots, k\}$ and by $\{k+1, \dots, n\}$, respectively, and the MP starts at time $\tau_k = \sum_{j=1}^{k} p_j$.

**Step 4**. Introduce the following minimization linear knapsack problem, which we call *Problem KP*.

$$\begin{aligned} \text{minimize} \quad & \sum_{j=1}^{n} w_j y_j \\ \text{subject to} \quad & \sum_{j=1}^{n} p_j y_j \geq \sum_{j=1}^{n} p_j - D_{MP} \\ & y_j \in \{0, 1\}, \quad j = 1, \dots, n. \end{aligned}$$

Use an FPTAS for the linear knapsack problem that finds a Boolean vector $\mathbf{y}^H = (y_1^H, \dots, y_n^H)$. Define schedule $S^{k_1}$ in which the first group and the second group of jobs are given by $\left\{j \in N | y_j^H = 0\right\}$ and by $\left\{j \in N | y_j^H = 1\right\}$, respectively, and the MP starts at time $\tau_{k_1} = \sum_{j \in N} p_j \left(1 - y_j^H\right)$.

**Step 5**. Output schedule $S^H$, which is the best of the found schedules.

Step 3 of Algorithm MP1WTFT_DD is similar to actions performed by Algorithm HKP($\rho$) of Sect. 13.4.1. Both algorithms involve finding an approximate solution to Problem KP, aimed at determining the smallest possible total weight of the jobs processed after the MP (for these jobs $y_j = 1$), provided that the other jobs will complete before the MP (this is represented by the knapsack constraint). We can think of this problem as the problem of minimizing the weighted number of late jobs, provided that the jobs have a common due date $D_{MP}$.

The running time of Algorithm MP1WTFT_DD is defined by (i) sorting the jobs in the WSPT order, which requires $O(n \log n)$ time, and (ii) solving Problem KP by an FPTAS, which can be done in $O(n/\varepsilon)$ time. Thus, the overall running time is $O(n \log n + n/\varepsilon)$.

Below, we analyze the worst-case performance of the algorithm. In particular, we prove the following statement.

**Theorem 14.4** *For problem* $1|\tau \le D_{MP}, \Delta(\tau)|\sum w_j C_j$ *with a concave function* $\Delta(\tau)$ *of the MP duration, suppose that in an optimal schedule* $S^*$ *the MP starts at time* $\tau^*$. *If* $\tau^* \le \tau_{k_1-1}$ *then*

$$\min\left\{Z\left(S^k\right)|0 \le k \le k_1 - 1\right\} \le \frac{4}{3}Z\left(S^*\right).$$

In order to prove Theorem 14.4, we start with proving several auxiliary statements. Suppose that the value of $\tau^*$ is known. Scanning the jobs in the order of their numbering, determine the job $\ell \in N$ such that

$$\sum_{j=1}^{\ell-1} p_j < \tau^* \le \sum_{j=1}^{\ell} p_j,$$

and find such a $\theta$, $0 \le \theta < 1$, that

$$\sum_{j=1}^{\ell-1} p_j + \theta p_\ell = \tau^*.$$

Modify the set $N$ of jobs by replacing job $\ell$ by a pair of jobs $\ell'$ and $\ell''$ such that

$$p_{\ell'} = \theta p_\ell,\ w_{\ell'} = \theta w_\ell;\ p_{\ell''} = (1-\theta)p_\ell,\ w_{\ell''} = (1-\theta)w_\ell.$$

Denote the modified set of jobs by $N_{(\ell)}$, i.e., define $N_{(\ell)} = \{1, \dots, \ell-1, \ell', \ell'', \ell+1, \dots, n\}$. For jobs of set $N_{(\ell)}$, consider a schedule $S_{(\ell)}$ such that the sequence of jobs $(1, \dots, \ell-1, \ell')$ is processed before time $\tau^*$, while the sequence of jobs $(\ell'', \ell+1, \dots, n)$ is processed after the MP. It follows that

$$\begin{aligned} Z\left(S_{(\ell)}\right) = &\sum_{j=1}^{\ell-1} w_j \sum_{i=1}^{j} p_i + \theta w_\ell \left(\sum_{i=1}^{\ell-1} p_i + \theta p_\ell\right) \\ &+(1-\theta)w_\ell \left(\sum_{i=1}^{\ell} p_i + \Delta\left(\sum_{i=1}^{\ell-1} p_i + \theta p_\ell\right)\right) \\ &+ \sum_{j=\ell+1}^{n} w_j \left(\sum_{i=1}^{j} p_i + \Delta\left(\sum_{i=1}^{\ell-1} p_i + \theta p_\ell\right)\right). \end{aligned} \tag{14.6}$$

**Lemma 14.4** *The lower bound*

$$Z\big(S^*\big) \geq Z\big(S_{(\ell)}\big)$$

*holds.*

*Proof* Suppose that in schedule $S^*$, job $\ell$ starts at time $R_\ell$. Take schedule $S^*$ and replace job $\ell$ by a sequence of jobs $\big(\ell', \ell''\big)$ and let the modified schedule be called $S'$. The completion times of all jobs other than $\ell$ are the same in both schedules, $S^*$ and $S'$. We also have

$$\begin{aligned} w_\ell C_\ell\big(S^*\big) &= w_\ell(R_\ell + p_\ell); \\ w_{\ell'} C_{\ell'}\big(S'\big) &= \theta w_\ell(R_\ell + \theta p_\ell); \\ w_{\ell''} C_{\ell''}\big(S'\big) &= (1-\theta) w_\ell(R_\ell + p_\ell), \end{aligned}$$

so that

$$w_\ell C_\ell\big(S^*\big) - w_{\ell'} C_{\ell'}\big(S'\big) + w_{\ell''} C_{\ell''}\big(S'\big) = \theta(1-\theta) w_\ell p_\ell \geq 0,$$

which proves the lemma. □

If in schedule $S_{(\ell)}$ the pair of jobs $\ell'$ and $\ell''$ is replaced by job $\ell$, then the resulting schedule is not feasible for the original problem $1|\tau \leq D_{MP}, \Delta(\tau)| \sum w_j C_j$, since job $\ell$ is partitioned. Convert $S_{(\ell)}$ into schedules, called $S_{(\ell)}^A$ and $S_{(\ell)}^B$, that do not allow any job splitting.

In schedule $S_{(\ell)}^A$, the MP starts at the completion time of job $\ell - 1$, while job $\ell$ starts immediately after the MP, so that

$$Z\big(S_{(\ell)}^A\big) = \sum_{j=1}^{\ell-1} w_j \sum_{i=1}^{j} p_i + \sum_{j=\ell}^{n} w_j \left( \sum_{i=1}^{j} p_i + \Delta\left( \sum_{i=1}^{\ell-1} p_i \right)\right). \tag{14.7}$$

Since in schedule $S_{(\ell)}^A$ the MP starts no later than in schedule $S^*$, we know that $S_1^A$ is feasible for the original problem.

In schedule $S_{(\ell)}^B$, the MP starts at the completion time of job $\ell$, while job $\ell + 1$ starts immediately after the MP, so that

$$Z\big(S_{(\ell)}^B\big) = \sum_{j=1}^{\ell} w_j \sum_{i=1}^{j} p_i + \sum_{j=\ell+1}^{n} w_j \left( \sum_{i=1}^{j} p_i + \Delta\left( \sum_{i=1}^{\ell} p_i \right)\right). \tag{14.8}$$

There is no guarantee that schedule $S_{(\ell)}^B$ is feasible for the original problem, since the MP may start after the deadline.

**Lemma 14.5** *The inequality*

$$\min\{Z(S^A_{(\ell)}), Z(S^B_{(\ell)})\} \le \frac{4}{3} Z(S_{(\ell)})$$

*holds.*

*Proof* Since function $\Delta(\tau)$ is concave, we have that

$$\Delta\left(\sum_{i=1}^{\ell-1} p_i + \theta p_\ell\right) \ge \theta\Delta\left(\sum_{i=1}^{\ell} p_i\right) + (1-\theta)\Delta\left(\sum_{i=1}^{\ell-1} p_i\right).$$

Compare $Z(S_{(\ell)})$ and $\theta Z\left(S^B_{(\ell)}\right)$. Notice that

$$\begin{aligned}
&\sum_{j=1}^{\ell-1} w_j \sum_{i=1}^{j} p_i + \theta w_\ell \left(\sum_{i=1}^{\ell-1} p_i + \theta p_\ell\right) - \theta \sum_{j=1}^{\ell} w_j \sum_{i=1}^{j} p_i \\
&= \sum_{j=1}^{\ell-1} w_j \sum_{i=1}^{j} p_i + \theta w_\ell \sum_{i=1}^{\ell-1} p_i + \theta^2 w_\ell p_\ell - \theta \sum_{j=1}^{\ell-1} w_j \sum_{i=1}^{j} p_i - \theta w_\ell \sum_{i=1}^{\ell-1} p_i - \theta w_\ell p_\ell \\
&= (1-\theta) \sum_{j=1}^{\ell-1} w_j \sum_{i=1}^{j} p_i + \theta(\theta-1) w_\ell p_\ell.
\end{aligned}$$

Besides, since function $\Delta(\tau)$ is concave, the inequality

$$(1-\theta) w_\ell \sum_{i=1}^{\ell} p_i + \Delta\left(\sum_{i=1}^{\ell-1} p_i + \theta p_\ell\right) \ge (1-\theta) w_\ell \left(\sum_{i=1}^{\ell} p_i + \Delta\left(\sum_{i=1}^{\ell-1} p_i\right)\right)$$

holds. Further, we deduce that

$$\begin{aligned}
&\sum_{j=\ell+1}^{n} w_j \sum_{i=1}^{j} p_i + \Delta\left(\left(\sum_{i=1}^{\ell-1} p_i + \theta p_\ell\right)\right) - \theta \sum_{j=\ell+1}^{n} w_j \left(\sum_{i=1}^{j} p_i + \Delta\left(\sum_{i=1}^{\ell} p_i\right)\right) \\
&= (1-\theta) \sum_{j=\ell+1}^{n} w_j \sum_{i=1}^{j} p_i + \sum_{j=\ell+1}^{n} w_j \left(\Delta\left(\sum_{i=1}^{\ell-1} p_i + \theta p_\ell\right) - \theta\Delta\left(\sum_{i=1}^{\ell} p_i\right)\right) \\
&\ge (1-\theta) \sum_{j=\ell+1}^{n} w_j \sum_{i=1}^{j} p_i + (1-\theta) \sum_{j=\ell+1}^{n} w_j \Delta\left(\sum_{i=1}^{\ell-1} p_i\right) \\
&= (1-\theta) \sum_{j=\ell+1}^{n} w_j \left(\sum_{i=1}^{j} p_i + \Delta\left(\sum_{i=1}^{\ell-1} p_i\right)\right).
\end{aligned}$$

Thus, the following inequality

$$\begin{aligned} Z\big(S_{(\ell)}\big) - \theta Z\big(S^B_{(\ell)}\big) \ge {} & (1-\theta)\sum_{j=1}^{\ell-1} w_j\left(\sum_{i=1}^{j} p_i\right) + \theta(1-\theta)w_\ell p_\ell \\ & +(1-\theta)w_\ell\left(\sum_{i=1}^{\ell} p_i + \Delta\left(\sum_{i=1}^{\ell-1} p_i\right)\right) \\ & +(1-\theta)\sum_{j=\ell+1}^{n} w_j\left(\sum_{i=1}^{j} p_i + \Delta\left(\sum_{i=1}^{\ell-1} p_i\right)\right) \\ = {} & (1-\theta)Z\big(S^A_{(\ell)}\big) + \theta(\theta-1)w_\ell p_\ell \end{aligned}$$

holds.

Since $\theta < 1$, we obtain $0 > \theta(\theta-1)w_\ell p_\ell \ge \theta(\theta-1)Z\left(S^B_{(\ell)}\right)$, so that

$$Z\big(S_{(\ell)}\big) - \theta Z\big(S^B_{(\ell)}\big) \ge (1-\theta)Z\big(S^A_{(\ell)}\big) + \theta(\theta-1)Z\big(S^B_{(\ell)}\big),$$

i.e.,

$$Z\big(S_{(\ell)}\big) \ge (1-\theta)Z\big(S^A_{(\ell)}\big) + \theta^2 Z\big(S^B_{(\ell)}\big) \ge \big(\theta^2-\theta+1\big)\min\big\{Z\big(S^A_{(\ell)}\big), Z\big(S^B_{(\ell)}\big)\big\}.$$

It is easy to verify that the minimum of $\theta^2-\theta+1$ is achieved for $\theta = \frac{1}{2}$ and is equal to $\frac{3}{4}$, which proves the lemma. □

If we knew the value of $\tau^*$, we would be able to find schedules $S^A_{(\ell)}$ and $S^B_{(\ell)}$, and provided that the latter schedule is feasible, for the better of the two schedules the value of the function would be at most $\frac{4}{3}$ times the optimum. Having found job $k_1$ in Step 1 of Algorithm MP1WTFT_DD, divide the time interval $[0, D_{MP}]$ into $k_1$ intervals

$$[0, p_1], (p_1, p_1+p_2], \ldots, \left(\sum_{j=1}^{k_1-2} p_j, \sum_{j=1}^{k_1-1} p_j\right], \left(\sum_{j=1}^{k_1-1} p_j, D_{MP}\right],$$

or, equivalently, into the intervals

$$[\tau_0, \tau_1], [\tau_1, \tau_2], \ldots, \big[\tau_{k_1-2}, \tau_{k_1-1}\big], \big[\tau_{k_1-1}, D_{MP}\big].$$

The optimal start time $\tau^*$ belongs to one of these intervals. The assumption of Theorem 14.4 says that $\tau^*$ belongs to one of the intervals $[\tau_0, \tau_1], [\tau_1, \tau_2], \ldots, \big[\tau_{k_1-2}, \tau_{k_1-1}\big]$. Under this assumption, job $\ell$ is one of the jobs $1, 2, \ldots, k_1-1$. Thus, both schedules $S^A_{(\ell)}$ and $S^B_{(\ell)}$ are contained among schedules $S^k$, $1 \le k \le k_1-1$, and Theorem 14.4 follows from Lemmas 14.4 and 14.5.

**Table 14.1** Complexity and approximability of problems $1|\tau \leq D_{MP}, \Delta(\tau)|\Phi$

| Objective Function | $\Delta$ | $\Delta(\tau)$ |
|---|---|---|
| $C_{\max}$ | $O(n)$ | $O(n)$ |
| $F(S) = \sum C_j$ | $O(n \log n)$ | $O(n \log n)$ |
| $Z(S) = \sum w_j C_j$ | NP-hard | NP-hard |
| | FPTAS $O\left(n^6/\varepsilon^3\right)$ | $\left(1 + \frac{\sqrt{2}}{2} + \varepsilon\right)$-approximation |

To complete the analysis of Algorithm MP1WTFT_DD, we must consider the assumption that $\tau^* \in \left[\tau_{k_1-1}, D_{MP}\right]$. The following statement can be proved.

**Theorem 14.5** *For problem* $1|\tau \leq D_{MP}, \Delta(\tau)|\sum w_j C_j$ *with a concave function* $\Delta(\tau)$ *of the MP duration, suppose that in an optimal schedule* $S^*$ *the MP starts at time* $\tau^* \in \left[\tau_{k_1-1}, D_{MP}\right]$*. Then*

$$\min\left\{C_{\max}\left(S^{k_1-1}\right), C_{\max}\left(S^{k_1}\right)\right\} \leq \left(1 + \frac{\sqrt{2}}{2} + \varepsilon\right) C_{\max}\left(S^*\right).$$

The proof of this theorem is rather technical and is not presented here (see Sect. 14.3 for details).

The results of this section on the complexity and approximability of the problems with a single MP that has to start by a given deadline are summarized in Table 14.1.

## 14.2 Flexible Maintenance Within a Window

In this section, we study extended versions of the problems considered in Sect. 14.1, in which the MP should be placed within a given window $[s, t]$. The focus is on the problems with a constant duration MP.

### *14.2.1 Minimizing Makespan: Single Maintenance*

In a schedule that is feasible for problem $1|MP \in [s, t], \Delta|C_{\max}$, the MP starts at time $\tau$ such that $\tau \geq s$ and $\tau + \Delta \leq t$. If it is allowed to interrupt the processing of a job by the MP and resume it immediately after the MP, i.e., if the resumable scenario is applied, then it is trivial to find a schedule that is optimal for the problem of minimizing the makespan. Indeed, in such a schedule, the jobs are processed in an arbitrary order and the MP starts at time $s$. This schedule remains optimal even if the duration of the MP is given by a non-decreasing function $\Delta(\tau)$ that depends on start time $\tau$ of the maintenance period. The makespan in such a schedule is equal to $p(N) + \Delta(s)$, which cannot be reduced.

Below, we assume that the non-resumable scenario is applied. In a feasible schedule $S$, the jobs are split into two groups: $N^{[1]}$ scheduled in accordance with a sequence $\pi^{[1]}$ before the MP and $N^{[2]}$ scheduled in accordance with a sequence $\pi^{[2]}$ after the MP. We discuss the complexity status of problem $1|MP \in [s,t], \Delta|C_{\max}$ and prove that the problem is NP-hard. We also demonstrate that an FPTAS for the subset-sum problem can be adapted to result into an FPTAS for problem $1|MP \in [s,t], \Delta|C_{\max}$.

In the proof of the NP-hardness, the following NP-complete problem is used for reduction (see Sect. 1.3.2).

PARTITION: Given positive integers $e_1, \dots, e_r$ and the index set $R = \{1, \dots, r\}$ such that $e(R) = \sum_{i \in R} e_i = 2E$, is it possible to partition set $R$ into disjoint subsets $R_1$ and $R_2$ such that $e(R_1) = \sum_{i \in R_1} e_i = E$ and $e(R_2) = \sum_{i \in R_2} e_i = E$?

**Theorem 14.6** *Problem* $1|MP \in [s,t], \Delta|C_{\max}$ *is NP-hard in the ordinary sense.*

*Proof* Given an instance of PARTITION, define the instance of problem $1|MP \in [s,t], \Delta|C_{\max}$ with

$$N = R;\ \ p_j = 3e_j,\ \ j \in N;\ \ s = 3E - 1,\ \ t = 3E + 1,\ \ \Delta = 1.$$

Thus, in a feasible schedule, the MP of duration $\Delta = 1$ must be placed within the window $[3E - 1, 3E + 1]$ of length 2.

We show that PARTITION has a solution if and only if in the constructed problem there exists a schedule $S_0$ such that $C_{\max}(S_0) \le 6E + 1$.

Suppose that PARTITION has a solution represented by the sets $R_1$ and $R_2$. Then, schedule $S_0$ with $C_{\max}(S_0) = 6E + 1$ exists and can be found as follows. Define $N^{[q]} = R_q$, $q \in \{1, 2\}$, and let in $S_0$ the jobs of set $N^{[1]}$ be processed before the MP, while the jobs of set $N^{[2]}$ be processed as a block starting from time $t$. Notice that the jobs of each set $N^{[q]}$ can be processed in any order. Since $p\left(N^{[1]}\right) = e(R_1) = 3E$, the MP starts at time $3E \in [s,t]$ and completes at time $t = 3E + 1$. Thus, we have that

$$C_{\max}(S_0) = p\left(N^{[1]}\right) + \Delta + p\left(N^{[2]}\right) = 3E + 1 + 3E = 6E + 1,$$

as required.

Now suppose that a schedule $S_0$ such that $C_{\max}(S_0) \le 6E + 1$ exists. The MP cannot start after time $3E$. Let $N^{[1]}$ denote the set of jobs completed before the MP. If $p\left(N^{[1]}\right) < 3E - 1$, then in $S_0$ the machine is idle before the MP may start, and we have that $C_{\max}(S_0) \ge s + \Delta + p\left(N \backslash N^{[1]}\right) > (3E - 1) + 1 + (3E + 1) = 6E + 1$, which is impossible.

Thus, $3E - 1 \le p\left(N^{[1]}\right) \le 3E$. Since each $p_j$ is an integer that is a multiple of 3, we may only have that $p\left(N^{[1]}\right) = 3E$. Thus, if we define $R_1 := N^{[1]}$ and $R_2 := N \backslash N^{[1]}$, we obtain a solution to PARTITION. $\square$

Since the value $\tau - \Delta$ serves as the deadline for the completion time of the jobs of the first group, the NP-hardness of problem $1|MP \in [s,t], \Delta|C_{\max}$, can also be deduced from Theorem 13.2 that proves that problem $1|CMP(1), N - Res|C_{\max}$ is NP-hard in the ordinary sense.

Associate problem $1|MP \in [s,t], \Delta|C_{\max}$ with the following subset-sum problem studied in Sect. 4.2. In such a problem, it is required to find a vector $\mathbf{x}^* = (x_1^*, x_2^*, \ldots, x_n^*)$, which delivers an optimal solution to the problem

$$\begin{aligned} &\text{maximize} \sum_{j=1}^{n} p_j x_j \\ &\text{subject to} \sum_{j=1}^{n} p_j x_j \le t - \Delta \\ &\qquad\qquad x_j \in \{0,1\},\ j = 1,2,\ldots,n. \end{aligned} \tag{14.9}$$

Any feasible solution $\mathbf{x} = (x_1, x_2, \ldots, x_2)$ defines a feasible schedule $S$ for problem $1|MP \in [s,t], \Delta|C_{\max}$, in which the jobs are split into two sets, $N^{[1]} = \{j \in N | x_j = 1\}$ and $N^{[2]} = \{j \in N | x_j = 0\}$. The block of jobs $N^{[1]}$ is processed starting from time zero, the MP starts at time $\tau = \max\{s, p(N^{[1]})\}$, and the block of jobs $N^{[2]}$ starts on the completion of the MP. In particular, an optimal schedule $S^*$ is defined by the two groups $H^{[1]} := \left\{j \in N | x_j^* = 1\right\}$ and $H^{[2]} := \left\{j \in N | x_j^* = 0\right\}$.

Recall that the subset-sum problem is known to admit a fully polynomial-time approximation scheme (FPTAS) that requires $O(n/\varepsilon)$ time (see Theorem 4.5). Based on this scheme, we establish the approximability status of problem $1|MP \in [s,t], \Delta|C_{\max}$.

**Theorem 14.7** *Problem $1|MP \in [s,t], \Delta|C_{\max}$ admits an FPTAS that requires $O(n/\varepsilon)$ time.*

*Proof* Let the vector with the components $x_j^\varepsilon \in \{0,1\}$, $j \in N$, be a solution delivered by the FPTAS for problem (14.9). Define $N^{[1]} := \left\{j \in N | x_j^\varepsilon = 1\right\}$, $N^{[2]} := \left\{j \in N | x_j^\varepsilon = 0\right\}$, and let $S^\varepsilon$ be a schedule associated with these sets. Consider an optimal schedule $S^*$ in which the block of jobs $H^{[1]}$ is processed before the MP and the block of jobs $H^{[2]}$ is processed after the MP. According to Theorem 4.5, the inequality

$$p(N^{[1]}) \ge (1-\varepsilon) p(H^{[1]}) \tag{14.10}$$

holds. We want to prove that

$$C_{\max}(S^\varepsilon) \le (1+\varepsilon) C_{\max}(S^*). \tag{14.11}$$

If $p(N^{[1]}) \ge s$, then in schedule $S^\varepsilon$ the MP may start at time $p(N^{[1]})$, so that $C_{\max}(S^\varepsilon) = p(N^{[1]}) + \Delta + p(N^{[2]}) = p(N) + \Delta$ and this schedule is in fact optimal. Thus, in the rest of this proof, we assume that $p(N^{[1]}) < s$, so that $C_{\max}(S^\varepsilon) = s + \Delta + p(N^{[2]}) = s + \Delta + p(N) - p(N^{[1]})$.

If $p\left(H^{[1]}\right) \geq s$, we deduce

$$s - p\left(N^{[1]}\right) \leq p\left(H^{[1]}\right) - p\left(N^{[1]}\right) \leq \varepsilon p\left(H^{[1]}\right),$$

so that

$$C_{\max}(S^{\varepsilon}) = s + \Delta + p(N) - p\left(N^{[1]}\right) \leq \Delta + p(N) + \varepsilon p\left(H^{[1]}\right) \leq (1+\varepsilon) C_{\max}\left(S^{*}\right),$$

On the other hand, if $p\left(H^{[1]}\right) < s$, then $C_{\max}(S^{*}) = s + \Delta + p\left(H^{[2]}\right) = s + \Delta + p(N) - p\left(H^{[1]}\right)$, so that due to (14.10), we have that

$$\begin{aligned} C_{\max}(S^{\varepsilon}) &= s + \Delta + p(N) - p\left(N^{[1]}\right) = C_{\max}\left(S^{*}\right) - \left(p\left(N^{[1]}\right) - p\left(H^{[1]}\right)\right) \\ &\leq C_{\max}\left(S^{*}\right) - (1-\varepsilon) p\left(H^{[1]}\right) + p\left(H^{[1]}\right) \leq C_{\max}\left(S^{*}\right) + \varepsilon p\left(H^{[1]}\right) \end{aligned}$$

and (14.11) holds. □

### 14.2.2 *Minimizing Makespan: Periodic Maintenance*

In Sect. 13.1.2, we consider a single machine problem to minimize the makespan in which the compulsory maintenance takes place periodically, with a fixed period of time $T$ that should elapse before the first CMP and between any pair of consecutive CMPs. In this subsection, we consider a more general version of that problem in which each maintenance should be placed inside a window that is repeated periodically.

More formally, let $\Delta$ denote the duration of an MP. For consistency of explanation, assume that each schedule includes a dummy "zero" MP that starts and completes at time $T_0 = 0$. For $q \geq 1$, let $N^{[q]}$ be the group of jobs that is scheduled between time $T_{q-1}$, the completion of the $(q-1)$th MP, and the beginning of the $q$th MP. The $q$th MP must start no earlier than $s$ time units since the completion of the previous MP and must finish no later than $t$ time units since the completion of the previous MP, i.e., the $q$th MP must be placed into a window $\left[T_{q-1} + s, T_{q-1} + t\right]$. If the jobs of group $N^{[q]}$ complete within the window, i.e., if $T_{q-1} + p\left(N^{[q]}\right) \in \left[T_{q-1} + s, T_{q-1} + t\right]$, or equivalently, $p\left(N^{[q]}\right) \in [s, t]$, then the $q$th MP starts at time $\tau_q := T_{q-1} + p\left(N^{[q]}\right)$; otherwise, if $p\left(N^{[1]}\right) < s$, then the $q$th MP starts at time $\tau_q := T_{q-1} + s$.

We refer to this problem as $1|MP(period), \Delta|C_{\max}$. It is assumed that $s \geq \max\left\{p_j | j \in N\right\}$; otherwise, the problem has no feasible solution.

Problem $1|CMP(period), N-Res|C_{\max}$ studied in Sect. 13.1.2 is a special case of problem $1|MP(period), \Delta|C_{\max}$, and the two problems coincide if $t = s + \Delta$, i.e., if each MP starts exactly after $s$ time units after the completion of the previous MP.

For problem $1|CMP(period), N - Res|C_{\max}$, Theorem 13.5 states that the problem does not accept a $\rho$-approximation algorithm with $\rho < 2$, unless $\mathcal{P} = \mathcal{NP}$. This immediately implies that for problem $1|MP(period), \Delta|C_{\max}$, the best approximation algorithm we could hope for is a 2-approximation algorithm. One such algorithm is presented below.

**Algorithm LS_Period**

```
Step 1.  Define
             $b := 1,\ N^{[1]} := \varnothing,\ p(N^{[1]}) := 0,\ q := 1.$

Step 2.  For $j$ from 1 to $n$ do
  (a)  If $p(N^{[q]}) + p_j > t$ then go to Step 2(b); otherwise update
             $N^{[q]} := N^{[q]} \cup \{j\};\ p(N^{[q]}) := p(N^{[q]}) + p_j$
       and go to Step 2(c).
  (b)  If $q < b$ then update $q := q + 1$ and return to Step 2(a); otherwise define
             $b := b + 1,\ N^{[b]} := \{j\},\ p(N^{[b]}) := p_j.$
  (c)  Restore $q := 1$ and take the next job.

Step 3.  Output a schedule $S_{LS}$ that consists of $b$ found groups $N^{[1]}, N^{[2]}, \ldots, N^{[b]}$.
```

Algorithm LS_Period implements the idea of list scheduling (see Sect. 2.4). It scans the jobs in an arbitrary order and assigns the next job to the first available group it fits. If the job does not fit into any available group, then a new group is started. The running time of the algorithm is $O(n^2)$.

**Theorem 14.8** *For schedule $S_{LS}$ the following bound*

$$\frac{C_{\max}(S_{LS})}{C_{\max}(S^*)} \le 2 \tag{14.12}$$

*holds, and this bound is tight.*

*Proof* Consider schedule $S_{LS}$ with $b$ groups. If $b = 1$, then all jobs are completed before the first MP, so that $S_{SL}$ is in fact an optimal schedule. Thus, throughout the remainder of this proof we assume that $b \ge 2$.

Since for each $q$, $1 \le q \le b$, we have that $p(N^{[q]}) \le t - \Delta$. It follows that

$$C_{\max}(S_{LS}) \le b(t - \Delta) + (b - 1)\Delta.$$

For any two groups $N^{[q_1]}$ and $N^{[q_2]}$ such that $1 \le q_1, q_2 \le b$, $q_1 \ne q_2$, we have that

$$p(N^{[q_1]}) + p(N^{[q_2]}) > t - \Delta; \tag{14.13}$$

otherwise, these groups could be merged into one.

If $b$ is even then we may split the groups into $b/2$ pairs $N^{[q_1]}$ and $N^{[q_2]}$ for which (14.13) holds, so that

$$p(N) > \frac{b}{2}(t - \Delta). \tag{14.14}$$

If $b$ is odd, then there is a group $N^{[q_3]}$ such that $p\left(N^{[q_3]}\right) > \frac{1}{2}(t - \Delta)$; otherwise, (14.13) would not hold for any pair of groups. Taking such a group and splitting the remaining groups in $(b-1)/2$ pairs $N^{[q_1]}$ and $N^{[q_2]}$ for which (14.13) holds, we obtain

$$p(N) > \frac{1}{2}(t - \Delta) + \frac{b-1}{2}(t - \Delta),$$

so that again (14.14) holds.

This implies that there must be at least $\lfloor b/2 \rfloor$ MPs in any optimal schedule $S^*$, i.e.,

$$C_{\max}\left(S^*\right) \geq p(N) + \lfloor b/2 \rfloor \Delta > \frac{b}{2}(t - \Delta) + \lfloor b/2 \rfloor \Delta.$$

Since $b - 1 \leq 2\lfloor b/2 \rfloor$, we deduce

$$C_{\max}(S_{\mathrm{LS}}) \leq b(t - \Delta) + (b-1)\Delta \leq b(t - \Delta) + 2\lfloor b/2 \rfloor \Delta \leq 2C_{\max}\left(S^*\right),$$

which proves the bound (14.12).

To see that the bound (14.12) is tight, consider the instance of problem $1|MP(period), \Delta|C_{\max}$ in which the duration of an MP set equal to some value $\Delta$, while the other parameters are as follows:

$$p_1 = 5,\ p_2 = 6,\ p_3 = 5,\ p_4 = 6,\ p_5 = 2;\ s = 9;\ t - \Delta = 12,$$

i.e., in any feasible schedule an MP starts no earlier that 9 and no later than 12 time units from the completion of the previous MP.

An optimal schedule $S^*$ shown in Fig. 14.3a uses only one MP, and all jobs are completed by the beginning of the second MP, so that $C_{\max}(S^*) = 24 + \Delta$. Schedule $S_{\mathrm{LS}}$ obtained by scanning the jobs in the order of their numbering is shown in Fig. 14.3b. It uses two MPs, so that $C_{\max}(S_{\mathrm{LS}}) = 24 + 2\Delta$. We deduce that

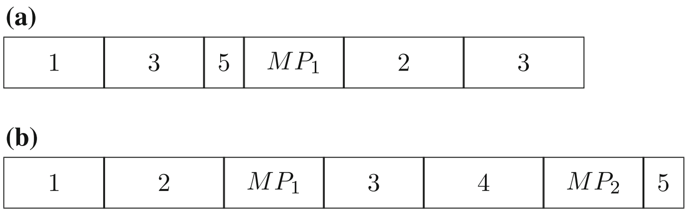


**Fig. 14.3** **a** An optimal schedule $S^*$; **b** schedule $S_{\mathrm{LS}}$

$$\frac{C_{\max}(S_{\mathrm{LS}})}{C_{\max}(S^*)} = \frac{24 + 2\Delta}{24 + \Delta},$$

and this ratio approaches 2 as $\Delta \to \infty$. □

### 14.2.3 Minimizing Total Completion Time

In this subsection, we study problem $1|MP \in [s,t], \Delta|\sum C_j$ of minimizing the sum of the completion times, i.e., the objective function $F(S) = \sum C_j(S)$.

If it is allowed to interrupt the processing of a job by the MP and resume it immediately after the MP, i.e., if the resumable scenario is applied, then problem $1|MP \in [s,t], \Delta|\sum C_j$ becomes equivalent to problem $1|CMP(1), Res|\sum C_j$, provided that in the latter problem, the maintenance period start at time $s$, the left point of the window in problem $1|MP \in [s,t], \Delta|\sum C_j$. Thus, Theorem 13.8 applies and leads to the following statement.

**Theorem 14.9** *Under the resumable scenario problem $1|MP \in [s,t], \Delta|\sum C_j$ is solvable in $O(n \log n)$ time by processing jobs in accordance with the SPT sequence (14.1).*

Further below in this subsection, we focus on the non-resumable version of problem $1|MP \in [s,t], \Delta|\sum C_j$. In a schedule that is feasible for problem $1|MP \in [s,t], \Delta|\sum C_j$, the jobs are split into two groups, $N^{[1]}$ and $N^{[2]}$, scheduled before and after the MP, respectively, and the MP starts at time $\tau$ such that $\tau \geq s$ and $\tau + \Delta \leq t$.

The value $\tau - \Delta$ serves as the deadline for the completion time of the jobs of the first group scheduled before the MP; therefore, due to similarity of problems $1|MP \in [s,t], \Delta|\sum C_j$ and $1|CMP(1), N - Res|\sum C_j$, the following statement can be deduced from Lemma 13.1 and Theorem 13.7.

**Lemma 14.6** *Under the non-resumable scenario, in an optimal schedule for problem $1|MP \in [s,t], \Delta|\sum C_j$, the jobs are sequenced in SPT order before and after the MP. Finding an optimal schedule is NP-hard in the ordinary sense.*

In this section, we present and analyze the following algorithm for solving problem $1|MP \in [s,t], \Delta|\sum C_j$. In the algorithm, we exclude from consideration the situation that $p(N) \leq t - \Delta$, since then it is optimal to schedule all jobs in the SPT sequence before the MP. By symmetry, we ignore the case that $\min\{p_j | j \in N\} > t - \Delta$, since then all jobs must be scheduled in the SPT order after the MP.

**Algorithm MP1TFT_Win**

**Step 1**. For problem $1|MP \in [s,t], \Delta|\sum C_j$, renumber the jobs in accordance with (14.1) and let $\pi_0 := (1, 2, \ldots, n)$.

**Step 2**. Considering the jobs in accordance with permutation $\pi_0$, find the job $\ell$ such that

$$\sum_{j=1}^{\ell-1} p_j \le t - \Delta, \quad \sum_{j=1}^{\ell} p_j > t - \Delta.$$

**Step 3**. Compute $A := \sum_{j=1}^{\ell-1} p_j$ and $\tau := \max\{A, s\}$.

**Step 4**. Output schedule $S_{\mathrm{SPT}}$ in which the jobs of the first group $N^{[1]} := \{1, \dots, \ell - 1\}$ and processed as a block starting from time zero, are followed by the MP that starts at time $\tau$, which is followed by the block of jobs of the second group $N^{[2]} := \{\ell, \dots, n\}$ that starts at time $\tau + \Delta$.

The running time of Algorithm MP1TFT_Win is $O(n \log n)$. Below we analyze its worst-case performance.

**Theorem 14.10** *For problem* $1|MP \in [s, t], \Delta| \sum C_j$, *let* $S^*$ *be an optimal schedule and* $S_{\mathrm{SPT}}$ *be a schedule found by Algorithm MP1TFT_Win. The bound*

$$\frac{F(S_{\mathrm{SPT}})}{F(S^*)} \le \frac{9}{7}, \tag{14.15}$$

*and this bound is tight.*

*Proof* For schedule $S_{\mathrm{SPT}}$, define $n^{[1]} := \ell - 1 = \left|N^{[1]}\right|$ and $n^{[2]} := \left|N^{[2]}\right|$. We know that in schedule $S_{\mathrm{SPT}}$, both groups $N^{[1]}$ and $N^{[2]}$ are non-empty, i.e., $n^{[1]} \ge 1$ and $n^{[2]} \ge 1$. Also, define the sequences $\pi^{[1]} = \left(1, \dots, n^{[1]}\right)$, $\pi^{[2]} = \left(n^{[1]} + 1, \dots, n^{[2]}\right)$ and $\pi_{\mathrm{SPT}} = \left(\pi^{[1]}, \pi^{[2]}\right)$.

Suppose that in an optimal schedule $S^*$, the jobs are processed in accordance with a sequence $\sigma = (\sigma(1), \sigma(2), \dots, \sigma(n))$. Introduce two sets $H^{[1]}$ and $H^{[2]}$, and the corresponding sequences $\sigma^{[1]}$ and $\sigma^{[2]}$, such that

**(i)** set $H^{[1]}$ contains $n^{[1]}$ jobs that are processed from time zero and follow the sequence $\sigma^{[1]} = \left(\sigma^{[1]}(1), \sigma^{[1]}(2), \dots, \sigma^{[1]}\left(n^{[1]}\right)\right)$;

**(ii)** set $H^{[2]}$ contains $n^{[2]}$ jobs that are processed in accordance with the sequence $\sigma^{[2]} = \left(\sigma^{[2]}(1), \sigma^{[2]}(2), \dots, \sigma^{[2]}\left(n^{[2]}\right)\right)$.

Notice that the sets $H^{[1]}$ and $H^{[2]}$ need not form the groups that are processed before and after the MP in schedule $S^*$.

For schedule $S_{\mathrm{SPT}}$, define $\delta := \tau - A$. This is the length of a possible idle time between the completion of the jobs of group $N^{[1]}$ and the start of the MP. Similarly, let $\delta^*$ denote a possible idle time in an optimal schedule $S^*$. Clearly, $\delta^*$ is no larger than the idle time in any other feasible schedule, so that

$$\delta \ge \delta^*.$$

By the SPT ordering, set $N^{[1]}$ contains $n^{[1]}$ shortest jobs of the instance, i.e.,

$$\sum_{j\in N^{[1]}} p_j = \sum_{j=1}^{\ell-1} p_j \le \sum_{r=1}^{n^{[1]}} p_{\sigma^{[1]}(r)} = \sum_{j\in H^{[1]}} p_j$$

holds. Besides, if we add the processing time of the first job of the sets $N^{[2]}$ and $H^{[2]}$ to both sides of the above inequality, we get

$$\sum_{j=1}^{n^{[1]}} p_j + p_{n^{[1]}+1} \le \sum_{r=1}^{n^{[1]}} p_{\sigma^{[1]}(r)} + p_{\sigma^{[2]}(1)}. \tag{14.16}$$

The inequality (14.16) holds because its left-hand side is the sum of the $n^{[1]}+1$ shortest processing times. By construction,

$$\sum_{j=1}^{n^{[1]}} p_j + p_{n^{[1]}+1} = A + p_\ell > t - \Delta,$$

and it follows that in schedule $S^*$, the last job of set $H^{[1]}$ completes after the MP, and the block of jobs $H^{[2]}$ also starts after the MP.

Inequality (14.16) implies that

$$\sum_{j\in N^{[1]}} p_j + p_{n^{[1]}+1} + \delta + \delta^* + \Delta \le \sum_{j\in H^{[1]}} p_j + p_{\sigma^{[2]}(1)} + \delta + \delta^* + \Delta,$$

which is equivalent to

$$C_{\pi^{[2]}(1)}(S_{\text{SPT}}) + \delta^* \le C_{\sigma^{[2]}(1)}\left(S^*\right) + \delta,$$

i.e.,

$$C_{\pi^{[2]}(1)}(S_{\text{SPT}}) \le C_{\sigma^{[2]}(1)}\left(S^*\right) + \left(\delta - \delta^*\right).$$

The above inequality can be generalized to all subsequent jobs, i.e., to

$$C_{\pi^{[2]}(i)}(S_{\text{SPT}}) \le C_{\sigma^{[2]}(i)}\left(S^*\right) + \left(\delta - \delta^*\right),\ 1 \le i \le n^{[2]}. \tag{14.17}$$

Define $F(S)$ as the sum of the completion times for all jobs in schedule $S$, and define $F_V(S)$ as the sum of the completion times for all jobs in schedule $S$ that belong to a set $V \subseteq N$. It follows from (14.17) that

$$F_{N^{[2]}}(S_{\text{SPT}}) \le F_{H^{[2]}}\left(S^*\right) + n^{[2]}\left(\delta - \delta^*\right). \tag{14.18}$$

For schedule $S^*$, let $\sigma^{[1]}(u)$ be the last job completed before the MP, i.e., the MP starts at time $\tau^* = \sum_{r=1}^{u} p_{\sigma^{[1]}(r)} + \delta^*$.

If $\delta = 0$, then $\delta^* = 0$ and there is no idle time in both schedules $S_{\mathrm{SPT}}$ and $S^*$. For an arbitrary permutation $\pi$ of jobs, recall the definition of $F_0(\pi)$, given by (14.2). Using the representation (14.2), we may write

$$F(S_{\mathrm{SPT}}) = F_0(\pi_{\mathrm{SPT}}) + n^{[2]}\Delta;\;\; F\left(S^*\right) = F_0(\sigma) + (n-u)\Delta$$

and since $F_0(\pi_{\mathrm{SPT}}) \le F_0(\sigma)$ and $n^{[2]} = n - n^{[1]} \le n - u$, we deduce that $S_{\mathrm{SPT}}$ is an optimal schedule. In the remainder of this proof, we assume that $\delta > 0$.

If $\delta^* = 0$, then it can be immediately verified that

$$\begin{aligned}\sum_{j=1}^{n^{[1]}} p_j + \delta &= C_{\pi^{[1]}\left(n^{[1]}\right)}(S_{\mathrm{SPT}}) + \delta \\ &= s \le \sum_{r=1}^{u} p_{\sigma^{[1]}(r)} \le C_{\sigma^{[1]}\left(n^{[1]}\right)}\left(S^*\right) \le C_{\sigma^{[1]}\left(n^{[1]}\right)}\left(S^*\right) + \delta^*,\end{aligned}$$

while if $\delta^* > 0$, we have

$$C_{\pi^{[1]}\left(n^{[1]}\right)}(S_{\mathrm{SPT}}) + \delta = C_{\sigma^{[1]}(u)}\left(S^*\right) + \delta^* \le C_{\sigma^{[1]}\left(n^{[1]}\right)}\left(S^*\right) + \delta^*.$$

Thus, in any case for $\delta^* \ge 0$, we deduce

$$C_{\pi^{[1]}\left(n^{[1]}\right)}(S_{\mathrm{SPT}}) \le C_{\sigma^{[1]}\left(n^{[1]}\right)(S^*)} - \left(\delta - \delta^*\right).$$

Moreover, since all jobs in $\pi^{[1]}$ are in SPT order, it follows that

$$C_{\pi^{[1]}(i)}(S_{\mathrm{SPT}}) \le C_{\sigma^{[1]}(i)}\left(S^*\right),\;\; 1 \le i \le n^{[1]}.$$

From the last two inequalities, we derive that

$$F_{N^{[1]}}(S_{\mathrm{SPT}}) \le F_{H^{[1]}}\left(S^*\right) - \left(\delta - \delta^*\right). \tag{14.19}$$

Combining (14.18) and (14.19), we obtain $F_{N^{[1]}}(S_{\mathrm{SPT}}) + F_{N^{[2]}}(S_{\mathrm{SPT}}) \le F_{H^{[1]}}(S^*) - (\delta - \delta^*) + F_{H^{[2]}}(S^*) + n^{[2]}(\delta - \delta^*)$, which yields

$$F(S_{\mathrm{SPT}}) \le F\left(S^*\right) + \left(n^{[2]} - 1\right)\left(\delta - \delta^*\right). \tag{14.20}$$

Recall that if in schedule $S_{\mathrm{SPT}}$ there exists an idle time of length $\delta$ before the start of the MP, then all jobs of set $N^{[2]}$ are longer than the duration of the idle time, i.e., $p_j > \delta,\ j \in N^{[2]}$; otherwise, the algorithm would include them into set $N^{[1]}$. Using this observation, we can derive the following lower bound

$$\begin{aligned} F_{N^{[2]}}\left(S^*\right) &\geq \delta + 2\delta + \cdots + n^{[2]}\delta = n^{[2]}\left(\frac{n^{[2]}+1}{2}\right)\delta \\ &\geq \frac{n^{[2]}\left(n^{[2]}+1\right)}{2}\left(\delta - \delta^*\right). \end{aligned} \tag{14.21}$$

If schedule $S_{\text{SPT}}$ is not optimal, then sets $N^{[1]}$ and $H^{[1]}$ are different, which implies that there exists a job $i$ which is assigned to set $N^{[1]}$ in schedule $S_{\text{SPT}}$, but is assigned to set $H^{[2]}$ in schedule $S^*$. Thus, we have the following relation

$$F_{N^{[1]}}\left(S^*\right) \geq F_{\{i\}}\left(S^*\right) = C_i\left(S^*\right) \geq s \geq \delta \geq \delta - \delta^*. \tag{14.22}$$

Combining (14.21) and (14.22), we obtain

$$F_{N^{[1]}}\left(S^*\right) + F_{N^{[2]}}\left(S^*\right) = F\left(S^*\right) \geq \left(\frac{n^{[2]}\left(n^{[2]}+1\right)+2}{2}\right)\left(\delta - \delta^*\right). \tag{14.23}$$

Using the bounds (14.20) and (14.23), we have

$$\begin{aligned} \rho &\leq \frac{F(S)}{F(S^*)} \leq 1 + \frac{\left(n^{[2]}-1\right)(\delta - \delta^*)}{F(S^*)} \\ &\leq 1 + \frac{\left(n^{[2]}-1\right)(\delta - \delta^*)}{\left(\frac{n^{[2]}\left(n^{[2]}+1\right)+2}{2}\right)(\delta - \delta^*)} = 1 + \frac{2\left(n^{[2]}-1\right)}{n^{[2]}\left(n^{[2]}+1\right)+2}. \end{aligned} \tag{14.24}$$

Taking the derivative of the function

$$f(x) = \frac{2(x-1)}{x(x+1)+2},\ 1 \leq x \leq n-1,$$

with respect to $x$, we obtain

$$f'(x) = \frac{2(x-3)(x+1)}{(x(x+1)+2)^2}.$$

We see that function $f(x)$ achieves a maximum value of $2/7$ for $x = 3$, and the worst-case performance bound given by (14.24) can be updated as

$$\rho \leq 1 + \frac{2}{7} = \frac{9}{7}.$$

To see that the bound (14.15) is tight, consider the following instance of problem $1|MP \in [s,t], \Delta|\sum C_j$:

$$p_1 = 1,\, p_2 = p_3 = p_4 = W,\, s = W-1,\, t = W+1,\, \Delta = 1,$$

where $W \geq 2$ is a large given number.

Algorithm MP1TFT_Win will schedule the jobs in the sequence $\pi^{[1]} = (1)$ and $\pi^{[2]} = (2, 3, 4)$. The completion time of job $\pi^{[1]}(1)$ is given by $C_{\pi^{[1]}(1)} = 1$. The MP of length 1 unit starts at time $\tau = W - 1$, leaving the machine idle for $\delta = W - 2$ units. The completion times of the remaining jobs are given by $C_{\pi^{[2]}(1)} = 2W$, $C_{\pi^{[2]}(2)} = 3W$, and $C_{\pi^{[2]}(3)} = 4W$. The sum of completion times is equal to $9W + 1$.

On the other hand, in an optimal schedule, the jobs are processed in accordance with the sequence $(2, 1, 3, 4)$, so that $\sigma^{[1]} = (2)$ and $\sigma^{[2]} = (1, 3, 4)$. The completion time of job $\sigma^{[1]}(1)$ is given by $C_{\sigma^{[1]}(1)} = W$. An MP of length 1 starts at time $\tau = W$, leaving no machine idle time. The completion times of the remaining jobs are given by $C_{\sigma^{[2]}(1)} = W + 2$, $C_{\sigma^{[2]}(2)} = 2W + 2$ and $C_{\sigma^{[2]}(3)} = 3W + 2$. The sum of completion times is equal to $7W + 6$.

The performance ratio given by $(9W + 1)/(7W + 2)$ tends to $9/7$ as $W$ goes to infinity. $\square$

## 14.3 Bibliographic Notes

The models with maintenance periods whose duration depends on their start time have been introduced by Kubzin and Strusevich (2005, 2006), for two-machine shop scheduling problems such as flow shop, flow shop no-wait, and open shop.

The fact that problem $1|\tau \leq D_{MP}, \Delta(\tau)|C_{\max}$ can be solved by starting the MP at time zero, as well as an alternative proof of Lemma 14.2, is stated in Luo et al. (2015) (see also Graves and Lee (1999) for the case of a constant duration MP and extensions). Luo et al. (2015) also present an algorithm similar to Algorithm MP1TFT_DD, but implemented in $O(n^2)$ time.

For problem $1|\tau \leq D_{MP}, \Delta|\sum w_j C_j$, Lemma 14.3 is proved by Kellerer and Strusevich (2010). A direct proof of the NP-hardness of problem $1|\tau \leq D_{MP}, \Delta|\sum w_j C_j$ is given in Graves and Lee (1999).

As pointed out in Kellerer and Strusevich (2010), problem $1|\tau \leq D_{MP}, \Delta|\sum w_j C_j$ is closely related to one of the single machine scheduling problems with two competing agents, studied by Agnetis et al. (2004) among other two-agent scheduling problems. Suppose that two agents intend to use a single machine. Agent $A$ owns the $A$-jobs, while Agent $B$ owns the $B$-jobs. Agent $A$ wants to minimize the sum of the weighted completion times of the $A$-jobs, while Agent $B$ wants to have all the $B$-jobs completed by a given deadline $d$. It is easily verified that in any feasible schedule, the $B$-jobs can be processed as a block, without intermediate idle time, and this will not increase the objective function of Agent $A$. Thus, provided that the processing times and weights of the $A$-jobs are equal to $p_j$ and $w_j$, respectively, and the total processing time of the $B$-jobs is equal to $\Delta$, the two-agent problem is equivalent to problem $1|\tau \leq D_{MP}, \Delta|\sum w_j C_j$ with $D_{MP} = d$. Notice that the two-agent problem is proved NP-hard in the ordinary sense (see Agnetis et al. (2004)).

Algorithm MP1WTFT_DD and the full proof of Theorem 14.5 are given in Luo et al. (2010).

The proof of Theorem 14.6 is based on the work by Yang et al. (2002).

The study on problem $1|MP(period), \Delta|C_{\max}$ has been initiated by Chen (2006, 2008). In particular, Chen (2008) presents a version of Algorithm LS in which the jobs are organized in the LPT list (see Sect. 13.1.2 for a similar algorithm for problem $1|CMP(period), N-Res|C_{\max}$). The running time of Chen's algorithm is $O(n^2)$, although in the original paper Chen (2008), the running time is estimated as $O(n^2 \log n)$. Xu et al. (2009) show that Chen's LPT algorithm is in fact a 2-approximation for problem $1|MP(period), \Delta|C_{\max}$. Algorithm LS_Period and its analysis in Theorem 14.8 are due to Xu and Yin (2011); in fact, in the latter paper, the result of Theorem 14.8 is extended to the online version of the problem. An extension of problem $1|MP(period), \Delta|C_{\max}$ is considered in Low et al. (2010), where it is additionally required not to process more than $k$ jobs between any two consecutive MPs; for $k \geq 2$, a $(2-2/k)$-approximation algorithm is developed based on the LPT scheduling.

Analysis of Algorithm MP1TFT_Win is performed by Yang et al. (2002), who base their reasoning on the work by Lee and Liman (1992), where a similar SPT $\frac{9}{7}$-approximation algorithm applied to a less general problem $1|CMP(1), N-Res|\sum C_j$, is analyzed.

## References

Agnetis A, Mirchandani PB, Pacciarelli D, Pacifici A (2004) Scheduling problems with two competing agents. Oper Res 52:229–242

Chen JS (2006) Single-machine scheduling with flexible and periodic maintenance. J Oper Res Soc 57:703–710

Chen JS (2008) Scheduling of nonresumable jobs and flexible maintenance activities on a single machine to minimize makespan. Eur J Oper Res 190:90–102

Graves GH, Lee C-Y (1999) Scheduling maintenance and semiresumable jobs on a single machine. Naval Res Logist 48:845–863

Kellerer H, Strusevich VA (2010) Fully polynomial approximation schemes for a symmetric quadratic knapsack problem and its scheduling applications. Algorithmica 57:769–795

Kubzin MA, Strusevich VA (2005) Two-machine flow shop no-wait scheduling with machine maintenance. 4OR 3:303–313

Kubzin MA, Strusevich VA (2006) Planning machine maintenance in two-machine shop scheduling. Oper Res 54:789–800

Lee C-Y, Liman SD (1992) Single machine flow time scheduling with scheduled maintenance. Acta Inform 29:375–382

Low C, Ji M, Hsu C-J, Su C-T (2010) Minimizing the makespan in a single machine scheduling problem with flexible and periodic maintenance. Appl Math Model 34:334–342

Luo W, Chen L, Zhang G (2010) Approximation algorithms for scheduling with a variable machine maintenance. In: Algorithmic aspects of information management, Lecture notes in computer science, vol 6124, 209–219

Luo W, Cheng TCE, Ji M (2015) Single-machine scheduling with a variable maintenance activity. Comp Industr Eng 79:168–174

Mosheiov G, Sidney JB (2010) Scheduling a deteriorating maintenance activity on a single machine. J Oper Res Soc 61:882–887

Xu DH, Yin YQ, Li H (2009) A note on "scheduling of nonresumable jobs and flexible maintenance activities on a single machine to minimize makespan". Eur J Oper Res 197:825–827

Xu DH, Yin YQ (2011) On single-machine scheduling with flexible maintenance activities. Int J Adv Manufact Technol 56:1139–1145

Yang D-L, Hung C-L, Hsu C-J (2002) Minimizing the makespan in a single machine scheduling problem with a flexible maintenance. J Chin Inst Indust Eng 19(1):63–66

Yang S-L, Ying M, Xu D-L, Yang J-B (2011) Minimizing total completion time on a single machine with a flexible maintenance activity. Comput Oper Res 38:755–770

# Chapter 15
# Scheduling with Rate-Modifying Activities

In this chapter, we discuss scheduling problems on a single machine, provided that rate-modifying activities can be inserted into a schedule. We are given the jobs of set $N = \{1, 2, \ldots, n\}$ with known normal processing times. The actual processing time of the job is affected by its relative position with respect to the rate-modifying periods (RMPs) that might be introduced into a schedule.

As described in Sect. 12.3, the decision-maker is presented with a list (RMP$^{[1]}$, RMP$^{[2]}$, ..., RMP$^{[K]}$) of $K \geq 1$ possible rate-modifying activities, which can be either distinct or alike. The decision-maker may decide which of the listed RMPs to insert into a schedule and in which order.

The algorithms presented in this chapter for solving relevant problems are adaptations of the generic Procedure RMP1 from Sect. 12.4. As a rule, to solve scheduling problems under consideration, it is required to generate all outcomes of certain decisions to produce auxiliary problems with fixed parameters, such as the number of RMPs to be inserted, and/or the number of jobs in a group. To count the number of the related outcomes, we use various combinatorial configurations and identities listed in Sect. 5.3. As agreed earlier, in the estimations of the running times of the presented algorithms, we assume that the number $K$ of available RMPs is a constant.

Whenever possible, we discuss faster solution algorithms, not based on full enumeration of certain decisions.

As often is the case in this book, two sequences of the jobs are important. Recall that if the jobs are numbered in accordance with the LPT rule, then

$$p_1 \geq p_2 \geq \cdots \geq p_n, \tag{15.1}$$

while if they are numbered in accordance with the SPT rule, then

$$p_1 \leq p_2 \leq \cdots \leq p_n. \tag{15.2}$$

V.A. Strusevich and K. Rustogi, *Scheduling with Times-Changing Effects and Rate-Modifying Activities*, International Series in Operations Research & Management Science 243, DOI 10.1007/978-3-319-39574-6_15

We start with considering problems on a single machine to minimize the makespan and the total completion time, provided that at most one RMP can be inserted into a schedule (see Sect. 15.1). The problem of minimizing the makespan and the total completion time for the case of multiple RMPs is studied in Sect. 15.2. The problems of minimizing the total completion time on parallel machines are considered in Chap. 20.

## 15.1 Single Rate-Modifying Maintenance on a Single Machine

We start with a simple model, in which the jobs of set $N = \{1, 2, \dots, n\}$ have to be scheduled on a single machine, and a single RMP can be inserted into a schedule. All jobs are available at time zero, and no preemption is allowed. For a job $j \in N$, its (normal) processing time $p_j$ is given and remains constant throughout the planning period if job $j$ is processed before the RMP. On the other hand, if job $j$ is processed after the RMP, then the processing time is given by $\lambda_j p_j$, where $\lambda_j > 0$ is the rate-modifying factor.

In general, there are no restrictions imposed on the value of $\lambda_j$. If $\lambda_j < 1$ for some job $j \in N$, then the introduction of the RMP is beneficial for the job and its actual processing time becomes shorter than the normal time. If $\lambda_j < 1$ for all jobs $j \in N$, then the rate-modifying activity can be understood as maintenance. We do not exclude from consideration that the rate-modifying factors are smaller than 1 for some jobs and larger than 1 for other jobs.

In this section, we address the situation that the duration of the RMP depends linearly on its start time $\tau$. Adopting the formula (12.1) for a single RMP, the duration $\Delta(\tau)$ of the RMP is given by

$$\Delta(\tau) = \zeta\tau + \eta, \tag{15.3}$$

where $\zeta$ and $\eta$ are positive constants. If $\zeta = 0$, then the duration of the RMP is constant equal to $\eta$.

The objective functions considered in this section are the makespan and total completion time. For each of these objective functions, the respective problems are denoted by $1|RMP(1), \Delta(\tau)|C_{\max}$ and $1|RMP(1), \Delta(\tau)|\sum C_j$.

In the case of a single available RMP, outcomes (A1) and (A2) from Sect. 12.4 boil down to deciding whether to include the RMP into a schedule or not. Thus, in accordance with the reasoning presented in Sect. 12.4, a schedule $S^*$ that is optimal for problem $1|RMP(1), \Delta(\tau)|\Phi$ is the better of the two schedules $S^*(1)$ and $S^*(2)$, which are optimal for the corresponding auxiliary problems $1|RMP(k-1), \Delta(\tau)|\Phi$, where $k \in \{1, 2\}$.

If $k = 1$, i.e., $k - 1 = 0$ and no RMP is to be included, then in schedule $S^*(1)$ all jobs are scheduled as one group starting form time zero. If $\Phi = C_{\max}$, then the sequence of this jobs is arbitrary, while for $\Phi = \sum C_j$, the jobs must be sequenced in accordance with the SPT rule (15.2), as follows from Theorem 2.2.

In the consideration below, we discuss how to find an optimal schedule $S^*(2)$, in which the RMP is inserted.

Let $S$ be a schedule with a single inserted RMP, in which the $n$ jobs are partitioned in two groups, $N^{[1]}$ and $N^{[2]}$, with $\left|N^{[x]}\right| = n^{[x]}$, where $x \in \{1, 2\}$ and $n^{[1]} + n^{[2]} = n$. Further, the jobs of the $x$th group are sequenced in accordance with a permutation $\pi^{[x]} = \left(\pi^{[x]}(1), \pi^{[x]}(2), \ldots, \pi^{[x]}\left(n^{[x]}\right)\right)$; i.e., schedule $S$ is associated with a permutation of jobs $\pi = \left(\pi^{[1]}, \pi^{[2]}\right)$. In order to find an optimal schedule $S^*(2)$, we need to find the optimal value of $n^{[1]}$ (or $n^{[2]}$) and the job sequence $\pi$, so that the objective function $\Phi \in \left\{C_{\max}, \sum C_j\right\}$ is minimized.

### 15.1.1 Minimizing Makespan

For a schedule $S$ associated with a permutation $\pi$, the makespan can be written as

$$\begin{aligned} C_{\max}(S) &= \sum_{r=1}^{n^{[1]}} p_{\pi^{[1]}(r)} + \left(\zeta \sum_{r=1}^{n^{[1]}} p_{\pi^{[1]}(r)} + \eta\right) + \sum_{r=1}^{n^{[2]}} \lambda_{\pi^{[2]}(r)} p_{\pi^{[2]}(r)} \\ &= (1+\zeta) \sum_{r=1}^{n^{[1]}} p_{\pi^{[1]}(r)} + \sum_{r=1}^{n^{[2]}} \lambda_{\pi^{[2]}(r)} p_{\pi^{[2]}(r)} + \eta. \end{aligned}$$

It is fairly easy to find schedule $S^*(2)$ that minimizes this function and, therefore, to find schedule $S^*$ that is optimal for the overall problem $1|RMP(1), \Delta(\tau)|\Phi$.

**Algorithm RMP1Cmax**

```
Step 1. For each $j$ from 1 to $n$ do
    If $(1+\zeta) \le \lambda_j$, $j \in N$, then add job $j$ to group 1; otherwise, add job $j$ to group 2.
Step 2. Permute the jobs arbitrarily in each group and create a schedule $S^*(2)$
    associated with the obtained permutation.
Step 3. If $\sum_{r=1}^{n^{[2]}} \left((1+\zeta) p_{\pi^{[2]}(r)} - \lambda_{\pi^{[2]}(r)} p_{\pi^{[2]}(r)}\right) > \eta$, then output schedule $S^*(2)$ as
    an optimal schedule $S^*$; otherwise, output an optimal schedule $S^*(1)$ in which all
    jobs are arbitrary sequenced in one group and no RMP is involved.
```

**Theorem 15.1** *Algorithm RMP1Cmax returns an optimal solution for problem $1|RMP(1), \Delta(\tau)|C_{\max}$ in $O(n)$ time.*

*Proof* Step 1 compares the contribution of a job to the makespan, when it is scheduled before and after the RMP. If a job is found to contribute less before the RMP, it is scheduled in group 1; otherwise, it is sent to group 2. Notice that the sequence of jobs in a particular group has no impact on the makespan; hence, they can be scheduled arbitrarily. Steps 1 and 2 can be completed in $O(n)$ time. Step 3 gives a condition which allows us to choose the better of the two schedules $S^*(1)$ or $S^*(2)$. Step 3 can also be completed in $O(n)$ time. $\square$

If the RMP's duration is assumed to be a constant, i.e., $\zeta = 0$, then the resulting problem $1|RMP(1), \Delta|C_{\max}$ will still be solvable by Algorithm RMP1Cmax that requires $O(n)$ time.

The following statement immediately follows for simpler versions of problem $1|RMP(1), \Delta(\tau)|C_{\max}$.

**Corollary 15.1** *For problem $1|RMP(1), \Delta(\tau)|C_{\max}$, if the RMP is a period of maintenance that improves the running times of all jobs, i.e., $\lambda_j \le 1$, $j \in N$, or if the factor $\lambda_j$ is job-independent, i.e., $\lambda_j = \lambda$, $j \in N$, then an optimal schedule is the better of the two schedules: $S^*(1)$ in which no RMP is run and the jobs are processed from time zero; or $S^*(2)$, in which the RMP is run from time zero and all jobs are sequenced after the RMP.*

## 15.1.2 Minimizing Total Completion Time

For problem $1|RMP(1), \Delta(\tau)|\sum C_j$, we show how to adapt Procedure RMP1 to its solution. We only consider the situation that the RMP is included into a schedule. Consider an arbitrary outcome of Decision (B1), i.e., fix the number of jobs in the second group to be equal to $\ell$, $1 \le \ell \le n$. Then, $n^{[1]} = n - \ell$. For a schedule $S_{\mathrm{B1}}(2)$, associated with a permutation of jobs $\pi = \left(\pi^{[1]}, \pi^{[2]}\right)$, the completion time of a job $j = \pi^{[1]}(r)$ sequenced in position $r$, $1 \le r \le n^{[1]}$, of the first group can be written as

$$C_{\pi^{[1]}(r)} = \sum_{i=1}^{r} p_{\pi^{[1]}(i)},$$

while the completion time of a job $j = \pi^{[2]}(r)$ sequenced in position $r$, $1 \le r \le n^{[2]}$, of the second group can be written as

$$C_{\pi^{[2]}(r)} = (1+\zeta)\sum_{r=1}^{n^{[1]}} p_{\pi^{[1]}(r)} + \sum_{i=1}^{r} \lambda_{\pi^{[2]}(i)} p_{\pi^{[2]}(i)} + \eta.$$

Extending formula (2.9), the sum of completion times for a schedule $S_{\mathrm{B1}}(2)$ can be written by

$$F(S_{\mathrm{B1}}(2)) = \sum_{r=1}^{n^{[1]}} \left(n - r + 1 + \zeta n^{[2]}\right) p_{\pi^{[1]}(r)} + \sum_{r=1}^{n^{[2]}} \left(n^{[2]} - r + 1\right) \lambda_{\pi^{[2]}(r)} p_{\pi^{[2]}(r)} + n^{[2]}\eta.$$

The above expression can be represented as

$$F(S_{\mathrm{B1}}(2)) = \sum_{x=1}^{2}\sum_{r=1}^{n^{[x]}} W_{\pi^{[x]}(r)}^{[x]}(r)p_{\pi^{[x]}(r)} + n^{[2]}\eta, \tag{15.4}$$

where

$$W_{\pi^{[x]}(r)}^{[x]}(r) = \begin{cases} n-r+1+\zeta n^{[2]}, & 1 \le r \le n^{[x]},\ x=1, \\ \left(n^{[2]}-r+1\right)\lambda_{\pi^{[x]}(r)}, & 1 \le r \le n^{[x]},\ x=2, \end{cases} \tag{15.5}$$

is a job-dependent positional weight, such that for a job $j = \pi^{[x]}(r)$ scheduled in position $r$, $1 \le r \le n^{[x]}$, of group $x$, $1 \le x \le 2$, the product $W_{\pi^{[x]}(r)}^{[x]}(r)p_{\pi^{[x]}(r)}$ represents the contribution of the job to the objective function (15.4).

Thus, the objective function (15.4) admits a generic representation (12.3), so that Procedure RMP1 is applicable.

Notice that to compute the weights $W_{\pi^{[x]}(r)}^{[x]}(r)$, $1 \le r \le n^{[x]},\ 1 \le x \le 2,\ j \in N$, given by (15.5), we require knowledge of the number of jobs $n^{[x]}$ assigned to group $x$, and these values are determined by Decision (B1), i.e., by the choice $n^{[2]} = \ell$. The total completion time of all jobs in schedule $S_{\mathrm{B1}}(2)$ associated with a permutation $\pi = \left(\pi^{[1]}, \pi^{[2]}\right)$ with $n^{[2]} = \ell$ can be written as a function of $\ell$, $1 \le \ell \le n$, given by

$$F(S_{\mathrm{B1}}(2)) = \sum_{x=1}^{2}\sum_{r=1}^{n^{[x]}} c_{\pi^{[x]}(r),(x,r)} + \ell\eta, \tag{15.6}$$

where the costs $c_{\pi^{[x]}(r),(x,r)}$ are defined by

$$c_{j,(x,r)} := \begin{cases} \left(n-r+1+\zeta n^{[2]}\right)p_j, & 1 \le r \le n^{[1]} = n-\ell,\ x=1, \\ (\ell-r+1)\lambda_j p_j, & 1 \le r \le n^{[2]} = \ell,\ x=2. \end{cases} \tag{15.7}$$

As discussed in Sect. 12.4, a schedule $S_{\mathrm{B1}}^{*}(2)$ such that $F\left(S_{\mathrm{B1}}^{*}(2)\right) \le F(S_{\mathrm{B1}}(2))$ holds for all schedules $S_{\mathrm{B1}}(2)$ with the same number $\ell$ of jobs in the second group can be found by solving the linear assignment problem (LAP) (12.5) with an $n \times n$ cost matrix $\mathbf{C} = \left(c_{j,(x,r)}\right)$ defined by (15.7). Such an LAP can be solved in $O(n^3)$ time, as follows from Theorem 4.1. For the found solution, $z_{j,(x,r)} = 1$ implies that in schedule $F\left(S_{\mathrm{B1}}^{*}(2)\right)$ job $j$ is assigned to the $r$th position of group $x$.

Trying all values of $\ell$, we will find schedule $S^{*}(2)$ which is optimal among all schedules with a single inserted RMP, i.e., $F(S^{*}(2)) \le F\left(S_{\mathrm{B1}}^{*}(2)\right)$ holds for all outcomes of Decision (B1). An overall optimal schedule $S^{*}$ is the best of the schedules $S^{*}(1)$, with no RMP, and $S^{*}(2)$, so that the following statement holds.

**Theorem 15.2** *Problem* $1|RMP(1), \Delta(\tau)|\sum C_j$ *reduces to* $O(n)$ *square linear assignment problems and can be solved in* $O\left(n^4\right)$ *time.*

If the RMP's duration is assumed to be a constant, i.e., $\zeta = 0$, then solving the resulting problem $1|RMP(1), \Delta|\sum C_j$ still requires $O\left(n^4\right)$ time, due to a similar reduction in $O(n)$ linear assignment problems.

Now, let us consider a special case in which the rate-modifying multiplier is job-independent, i.e., $\lambda_j = \lambda$, for all $j \in N$. Under this assumption, the formula (15.4) for the sum of the completion times for a schedule $S_{\mathrm{B1}}(2)$ can be written as

$$F(S_{\mathrm{B1}}(2)) = \sum_{x=1}^{2} \sum_{r=1}^{n^{[x]}} W^{[x]}(r) p_{\pi^{[x]}(r)} + n^{[2]}\eta, \tag{15.8}$$

where

$$W^{[x]}(r) = \begin{cases} n - r + 1 + \zeta n^{[2]}, & 1 \le r \le n^{[x]},\ x = 1, \\ \left(n^{[2]} - r + 1\right)\lambda, & 1 \le r \le n^{[x]},\ x = 2, \end{cases} \tag{15.9}$$

is a job-independent positional weight, such that for a job $j = \pi^{[x]}(r)$ scheduled in position $r$, $1 \le r \le n^{[x]}$, of group $x$, $1 \le x \le 2$, the product $W^{[x]}(r) p_{\pi^{[x]}(r)}$ represents the contribution of the job to the objective function (15.8).

Thus, the objective function (15.8) admits a generic representation (12.3), so that Procedure RMP1 is applicable.

Notice that to compute the weights $W^{[x]}(r)$, $1 \le r \le n^{[x]}$, $1 \le x \le 2$, given by (15.9), we still require knowledge of the values $n^{[x]}$, which are determined by the choice $n^{[2]} = \ell$. The total completion time of all jobs in schedule $S_{\mathrm{B1}}(2)$ associated with a permutation $\pi = \left(\pi^{[1]}, \pi^{[2]}\right)$ with $n^{[2]} = \ell$ can be written similar to (15.6), where the cost function is given by $c_{j,(x,r)} = W^{[x]}(r) p_j$, $1 \le r \le n^{[x]}$, $1 \le x \le 2$. As discussed in Sect. 12.4, a schedule $S^*_{\mathrm{B1}}(2)$ such that $F\left(S^*_{\mathrm{B1}}(2)\right) \le F(S_{\mathrm{B1}}(2))$ holds for all schedules $S_{\mathrm{B1}}(2)$ with the same number $\ell$ of jobs in the second group can be found by solving an LAP with a cost matrix $\mathbf{C} = \left(c_{j,(x,r)}\right)$. Notice that matrix $\mathbf{C}$ with the elements $c_{j,(x,r)} = W^{[x]}(r) p_j$, $1 \le r \le n^{[x]}$, $1 \le x \le 2$, may be represented as a product matrix, which makes the LAP solvable using Algorithm Match (see Sect. 4.1 for details). For a given value of $\ell$, Algorithm Match requires $O(n)$ time, since within a group, the positional weights $W^{[x]}(r)$, $1 \le r \le n^{[x]}$, $1 \le x \le 2$, are monotone, and hence, they can be sorted in $O(n)$ time.

Trying all values of $\ell$, we will find schedule $S^*(2)$ which is optimal among all schedules with a single inserted RMP, i.e., $F(S^*(2)) \le F\left(S^*_{\mathrm{B1}}(2)\right)$ holds for all outcomes of Decision (B1). An overall optimal schedule $S^*$ is the best of the schedules $S^*(1)$, with no RMP, and $S^*(2)$, so that the following statement holds.

**Theorem 15.3** *Problem $1|RMP(1), \Delta(\tau)|\sum C_j$ with job-independent rate-modifying multipliers, i.e., $\lambda_j = \lambda$, for all $j \in N$, reduces to $O(n)$ linear assignment problems with a product matrix and can be solved in $O\left(n^2\right)$ time.*

If the RMP's duration is assumed to be a constant, i.e., $\zeta = 0$, then solving the resulting problem $1|RMP(1), \Delta|\sum C_j$ will still require $O\left(n^2\right)$ time, due to a similar reduction to a series of linear assignment problems with a product matrix.

## 15.2 Multiple Rate-Modifying Maintenance Periods on a Single Machine

The model with a single RMP can be extended to a general situation, in which the decision-maker is presented with a list ($\mathrm{RMP}^{[1]}$, $\mathrm{RMP}^{[2]}$, …, $\mathrm{RMP}^{[K]}$) of $K \geq 1$ possible rate-modifying activities. Each RMP can have a different effect on the machine conditions, so that inserting $\mathrm{RMP}^{[y]}$ will modify the processing time of a job $j$ scheduled after it by a factor of $\lambda_j^{[y]}$, $1 \leq y \leq K$. Notice that the factor $\lambda_j^{[y]}$ applies not only to all jobs that are scheduled in the group that immediately follows that RMP, but also to all jobs in all subsequent groups.

*Example 15.1* For illustration, take a list of $K \geq 4$ RMPs and suppose that only two RMPs, namely $\mathrm{RMP}^{[2]}$ and $\mathrm{RMP}^{[4]}$, are chosen to be inserted into a schedule and performed in this order. Then, all jobs will be split into three groups. The processing times of the jobs scheduled in the first group, before the first scheduled RMP, i.e., $\mathrm{RMP}^{[2]}$, will be unaffected. For each job $j$ scheduled in the second group, i.e., between $\mathrm{RMP}^{[2]}$ and $\mathrm{RMP}^{[4]}$, the actual processing time becomes $\lambda_j^{[2]} p_j$. For each job $j$ scheduled in the third group, i.e., after $\mathrm{RMP}^{[4]}$, the actual processing time becomes $\lambda_j^{[2]} \lambda_j^{[4]} p_j$.

If $k-1$ RMPs are chosen from the available $K$ options, then the jobs are divided into $k$, $1 \leq k \leq K+1$ groups. Depending on which RMPs are chosen and the order in which they are performed, the actual processing time of a job $j \in N$, scheduled in position $r$ of the $x$th group, can be given by

$$p_j^{[x]}(r) = \mu_j^{[x]} p_j,\ 1 \leq r \leq n^{[x]},\ 1 \leq x \leq k, \tag{15.10}$$

where $\mu_j^{[x]}$ represents a group-dependent rate-modifying multiplier for job $j$, which depends on the previously scheduled RMPs and can be written as

$$\mu_j^{[x]} = \prod_{v=1}^{x-1} \lambda_j^{[v]},\ 1 \leq x \leq k. \tag{15.11}$$

For illustration, referring to Example 15.1 above, if RMP Decisions 1–3 determine that $\mathrm{RMP}^{[2]}$ and $\mathrm{RMP}^{[4]}$ are chosen to be inserted into a schedule in this order, then we have $\mu_j^{[1]} = 1$, $\mu_j^{[2]} = \lambda_j^{[2]}$, and $\mu_j^{[3]} = \lambda_j^{[2]} \lambda_j^{[4]}$, for all $j \in N$, where for the $\lambda$-values we maintain the numbering as in the initial list. On the other hand, if the chosen RMPs are inserted in the opposite order, then $\mu_j^{[1]} = 1$, $\mu_j^{[2]} = \lambda_j^{[4]}$, and $\mu_j^{[3]} = \lambda_j^{[2]} \lambda_j^{[4]}$.

As described in Sect. 12.3, the duration $\bar{\Delta}^{[x]}$ of the $x$th RMP is given by

$$\bar{\Delta}^{[x]} = \sum_{j \in N^{[x]}} \zeta_j^{[x]} p_j^{[x]}(r) + \eta^{[x]}, \tag{15.12}$$

where $\zeta_j^{[x]}$, $j \in N$, and $\eta^{[x]}$ are known parameters. It is assumed that in a schedule jobs that belong to set $N^{[x]}$ are scheduled in group $x$, i.e., before the $x$th RMP, and the contribution of a job $j \in N^{[x]}$ toward the actual duration $\bar{\Delta}^{[x]}$ of the $x$th RMP is equal to $\zeta_j^{[x]} p_j^{[x]}(r)$. Here, $p_j^{[x]}(r)$ is the actual processing time of job $j$ as defined in (15.10).

The problem of minimizing the makespan and the total completion time, respectively, under the settings defined by (15.10) and (15.12) is denoted by $1|RMP(K), \bar{\Delta}^{[x]}|\Phi$, for $\Phi \in \{C_{\max}, \sum C_j\}$, where the first term in the middle field represents that a list of $K$ RMPs is available for maintenance activities and the second term indicates that the durations of the RMPs follow a rule given by (15.12).

In order to solve problem $1\left|RMP(K), \bar{\Delta}^{[x]}\right|\Phi$ for $\Phi \in \{C_{\max}, \sum C_j\}$, we adapt Procedure RMP1 outlined in Sect. 12.4. Fix outcomes (A1) and (A2), and for a particular outcome of Decision (B1), introduce a schedule $S_{\text{B1}}(k)$ for an auxiliary problem $1\left|RMP(k-1), \bar{\Delta}^{[x]}\right|\Phi$, associated with certain outcomes of Decisions (B2) and (B3). In schedule $S_{\text{B1}}(k)$, the jobs are organized in groups $N^{[x]}$, $1 \le x \le k$, each group $N^{[x]}$ contains $n^{[x]}$ jobs, where $\sum_{x=1}^{k} n^{[x]} = n$. The jobs in $N^{[x]}$ are sequenced in accordance with a permutation $\pi^{[x]} = \left(\pi^{[x]}(1), \pi^{[x]}(2), \dots, \pi^{[x]}\left(n^{[x]}\right)\right)$, $1 \le x \le k$.

We now derive an expression for the total time it takes to process all jobs in a group $x$, $1 \le x \le k$, in a schedule $S_{\text{B1}}(k)$. It follows from (15.10) that the total processing time of the jobs assigned to group $x$ can be given by

$$E_x = \sum_{r=1}^{n^{[x]}} \mu_{\pi^{[x]}(r)}^{[x]} p_{\pi^{[x]}(r)}, \ 1 \le x \le k. \tag{15.13}$$

In accordance with (15.12), the duration $\bar{\Delta}^{[x]}$ of the RMP scheduled after the $x$th group is given by

$$T_x = \sum_{r=1}^{n^{[x]}} \zeta_{\pi^{[x]}(r)}^{[x]} \mu_{\pi^{[x]}(r)}^{[x]} p_{\pi^{[x]}(r)} + \eta^{[x]}, \ 1 \le x \le k-1. \tag{15.14}$$

### 15.2.1 Minimizing Makespan

For a schedule $S_{\text{B1}}(k)$, the makespan can be written as

$$C_{\max}(S_{\text{B1}}(k)) = E_1 + T_1 + E_2 + T_2 + \cdots + E_{x-1} + T_{x-1} + E_x.$$

It follows from (15.13) and (15.14) that

$$C_{\max}(S_{\mathrm{B}1}(k)) = \sum_{x=1}^{k-1}\sum_{r=1}^{n^{[x]}}(1+\zeta^{[x]}_{\pi^{[x]}(r)})\mu^{[x]}_{\pi^{[x]}(r)}p_{\pi^{[x]}(r)} + \sum_{r=1}^{n^{[k]}}\mu^{[k]}_{\pi^{[k]}(r)}p_{\pi^{[k]}(r)} + \sum_{x=1}^{k-1}\eta^{[x]}. \tag{15.15}$$

The above can be represented as

$$C_{\max}(S_{\mathrm{B}1}(k)) = \sum_{x=1}^{k}\sum_{r=1}^{n^{[x]}}W^{[x]}_{\pi^{[x]}(r)}p_{\pi^{[x]}(r)} + \sum_{x=1}^{k-1}\eta^{[x]}, \tag{15.16}$$

where

$$W^{[x]}_{\pi^{[x]}(r)} = \begin{cases}\left(1+\zeta^{[x]}_{\pi^{[x]}(r)}\right)\mu^{[x]}_{\pi^{[x]}(r)}, & \text{if } 1 \le r \le n^{[x]},\ 1 \le x \le k-1,\\ \mu^{[k]}_{\pi^{[k]}(r)}, & \text{if } 1 \le r \le n^{[x]},\ x = k,\end{cases} \tag{15.17}$$

is a job-dependent positional weight, such that the product $W^{[x]}_{\pi^{[x]}(r)}p_{\pi^{[x]}(r)}$ represents the contribution of job $j = \pi^{[x]}(r)$ scheduled in position $r$, $1 \le r \le n^{[x]}$, of group $x$, $1 \le x \le k$, to the objective function (15.15).

The function (15.16) admits a generic representation (12.3), so that Procedure RMP1 is in principle applicable. However, in this case, a faster approach can be employed, which neither requires full enumeration of possible outcomes of Decision (B1), nor requires the solution of a linear assignment problem in Step 1(b).

Notice that $W^{[x]}_{\pi^{[x]}(r)}$ depends on the group only, not on the position of a job within the group. Thus, for each job $j$, we can compute all weights without any knowledge of the number of jobs in a group as

$$W^{[x]}_{j} = \begin{cases}\left(1+\zeta^{[x]}_{j}\right)\mu^{[x]}_{j}, & \text{if } 1 \le x \le k-1,\\ \mu^{[k]}_{j}, & \text{if } x = k.\end{cases} \tag{15.18}$$

Let $S^*(k)$ denote a schedule that is optimal for problem $1\left|RMP(k-1), \bar{\Delta}^{[x]}\right| C_{\max}$, for all possible outcomes of Decision (B1). Schedule $S^*(k)$ can be found by the following algorithm.

**Algorithm RMPkCmax**

**Step 1**. For each job $j \in N$, compute the multipliers $\mu^{[x]}_j$ by (15.11) and the weights $W^{[x]}_j$ by (15.18).

**Step 2**. For $j$ from 1 to $n$ do

Identify a group index $x_j$ such that $W_j^{[x_j]} = \min\left\{W_j^{[x]} | 1 \le x \le k\right\}$. Assign job $j \in N$ to a group $N^{[x_j]}$.

**Step 3**. Permute the jobs arbitrarily in each group and output schedule $S^*(k)$ associated with the obtained permutation.

Algorithm RMPkCmax places each job into a group where it is matched to the smallest weight available for that job. The following statement obviously holds.

**Lemma 15.1** *Algorithm RMPkCmax returns schedule $S^*(k)$ that is optimal for problem $1\left|RMP(k-1), \bar{\Delta}^{[x]}\right|C_{\max}$ in $O(nk)$ time.*

Thus, in the case under consideration, we may simplify Procedure RMP1 by replacing Steps 1(b) and 1(c) by the following:

**Step 1(b)′**. Find schedule $S^*(k)$ by running Algorithm RMPkCmax.

In an optimal schedule $S^*(k)$, for an instance of problem $1\left|RMP(k-1), \bar{\Delta}^{[x]}\right| C_{\max}$, it is possible that some of the $k$ groups do not receive any jobs at all. Such a situation can occur if many RMPs, when performed back to back, are able to restore the machine to a better state, as compared to that achieved when a single RMP is performed. Moreover, the reduction in processing times of jobs scheduled after these RMPs is greater than the constant time it takes to perform the RMPs.

As proved in Lemma 12.1, the number of all possible outcomes (A1) and (A2) can be estimated as $\sum_{k=1}^{K+1} K^{k-1}$. Since Algorithm RMPkCmax requires $O(nk)$ time to run for a given $k$, $1 \le k \le K+1$, the total running time required to solve problem $1\left|RMP(K), \bar{\Delta}^{[x]}\right|C_{\max}$ can be estimated as $O\left(n \sum_{k=1}^{K+1} kK^{k-1}\right) = O(nK^{K+1})$, which is linear in $n$ for a constant $K$. Thus, the following statement holds.

**Theorem 15.4** *An optimal solution for problem $1\left|RMP(K), \bar{\Delta}^{[x]}\right|C_{\max}$ can be found in $O\left(nK^{K+1}\right)$ time by using Algorithm RMPkCmax as a subroutine.*

Notice that the running time of $O\left(nK^{K+1}\right)$ may be reduced if we have a situation in which some of the RMP Decisions do not need to be taken. For example, if all available RMPs are identical, then only RMP Decision 1 must be taken; i.e., we only need to find the optimal number of the RMPs in the schedule. In this case, the running time required to solve problem $1\left|RMP(K), \bar{\Delta}^{[x]}\right|C_{\max}$ can be estimated as $O\left(\sum_{k=1}^{K+1} nk\right) = O\left(nK^2\right)$. If $K = 1$, then the running time reduces to $O(n)$, which complies with Theorem 15.1.

Now, let us consider a special case in which the found positional weights are job-independent, i.e., when $\lambda_j^{[x]} = \lambda^{[x]}$, $1 \le x \le K+1$, and $\zeta_j^{[x]} = \zeta^{[x]}$, $1 \le x \le K$, for all $j \in N$. Such a case corresponds to a situation, in which each RMP has the same effect on every job, and every job has the same effect on the duration of each RMP. Under these circumstances, Step 2 of Algorithm RMPkCmax requires constant time only, instead of $O(nk)$ time, since it can be proved that each job will be scheduled in the last group. To see this, we adapt (15.18) and rewrite the found positional weights as

$$W^{[x]} = \begin{cases} \left(1+\zeta^{[x]}\right)\mu^{[x]}, & \text{if } 1 \le x \le k-1, \\ \mu^{[k]}, & \text{if } x = k. \end{cases} \tag{15.19}$$

It is clear that in an optimal schedule, only those RMP$^{[y]}$ for which $\lambda^{[y]} < 1$, $1 \le y \le K$, will be inserted, since they guarantee that the processing time of every job scheduled after an inserted RMP$^{[y]}$ is less than its normal processing time. Other RMPs can be ignored. If several such RMPs with $\lambda^{[y]} < 1$ are inserted in a schedule, it is easy to verify from (15.19) that the last group is associated with the smallest value of $W^{[x]}$, $1 \le x \le k$. As a result, all jobs will be scheduled in the last group, leaving the first $k-1$ groups empty.

Finally, to know which RMPs to schedule, we are required to enumerate all possibilities of outcomes (A1) and (A2), since each RMP is associated with different duration parameters and affects the subsequent processing times differently. Notice that the positional weights found by (15.19) are not affected by the order of the inserted RMPs, since only the positional weights associated with the last group are significant. Moreover, for given outcomes (A1) and (A2), the value of the makespan can be found in constant time, if the sum of all processing times is known in advance. The latter can be done in $O(n)$ time. As proved in Lemma 12.1, the number of all possible outcomes (A1) and (A2) can be estimated as $\sum_{k=1}^{K+1}\binom{K}{k-1} = 2^K$, since the order of jobs is not important.

Since Algorithm RMPkCmax requires $O(k)$ time to run for a given $k$, $1 \le k \le K+1$, the total running time required to solve problem $1\left|RMP(K), \Delta^{[x]}\right|C_{\max}$, can be estimated as $O\left(n + \sum_{k=1}^{K+1} k\binom{K}{k-1}\right) = O(n + 2^K K) = O(n)$. The following statement holds.

**Theorem 15.5** *An optimal solution for problem* $1\left|RMP(K), \Delta^{[x]}\right|C_{\max}$, *with* $\lambda_j^{[x]} = \lambda^{[x]}$, $1 \le x \le K+1$, *and* $\zeta_j^{[x]} = \zeta^{[x]}$, $1 \le x \le K$, *for all* $j \in N$, *can be found in* $O(n)$ *time by using Algorithm RMPkCmax as a subroutine.*

### *15.2.2 Minimizing Total Completion Time*

For a schedule $S_{B1}(k)$, the completion time of a job $j = \pi^{[x]}(r)$ scheduled in position $r$ of the $x$th group can be given by

$$C_{\pi^{[x]}(r)} = E_1 + T_1 + E_2 + T_2 + \cdots + E_{x-1} + T_{x-1} + \sum_{u=1}^{r} \mu^{[x]}_{\pi^{[x]}(u)} p_{\pi^{[x]}(u)}.$$

It follows from (15.13) and (15.14) that

$$C_{\pi^{[x]}(r)} = \sum_{v=1}^{x-1}\sum_{u=1}^{n^{[v]}}(1+\zeta^{[v]}_{\pi^{[v]}(u)})\mu^{[v]}_{\pi^{[v]}(u)}p_{\pi^{[v]}(u)} + \sum_{u=1}^{r}\mu^{[x]}_{\pi^{[x]}(u)}p_{\pi^{[x]}(u)} + \sum_{v=1}^{x-1}\eta^{[v]}.$$

Thus, the sum of the completion times $F(S_{B1}(k)) = \sum C_j(S_{B1}(k))$ can be written as

$$F(S_{B1}(k)) = \sum_{x=1}^{k}\sum_{r=1}^{n^{[x]}}\left[\sum_{v=1}^{x-1}\sum_{u=1}^{n^{[v]}}(1+\zeta^{[v]}_{\pi^{[v]}(u)})\mu^{[v]}_{\pi^{[v]}(u)}p_{\pi^{[v]}(u)} + \sum_{u=1}^{r}\mu^{[x]}_{\pi^{[x]}(u)}p_{\pi^{[x]}(u)} + \sum_{v=1}^{x-1}\eta^{[v]}\right]$$

Substitute the constant term

$$\Gamma(k) = \sum_{x=1}^{k}\sum_{r=1}^{n^{[x]}}\sum_{v=1}^{x-1}\eta^{[v]} \tag{15.20}$$

and rearrange the above expression to obtain

$$\begin{aligned}
F(S_{B1}(k)) &= \sum_{x=1}^{k}\left[n^{[x]}\sum_{v=1}^{x-1}\sum_{u=1}^{n^{[v]}}(1+\zeta^{[v]}_{\pi^{[v]}(u)})\mu^{[v]}_{\pi^{[v]}(u)}p_{\pi^{[v]}(u)} + \sum_{r=1}^{n^{[x]}}\sum_{u=1}^{r}\mu^{[x]}_{\pi^{[x]}(u)}p_{\pi^{[x]}(u)}\right] \\
&\quad +\Gamma(k) \\
&= \sum_{x=1}^{k}\left(\sum_{r=1}^{n^{[x]}}(1+\zeta^{[x]}_{\pi^{[x]}(r)})\mu^{[x]}_{\pi^{[x]}(r)}p_{\pi^{[x]}(r)}\right)\left(\sum_{v=x+1}^{k}n^{[v]}\right) \\
&\quad +\sum_{x=1}^{k}\sum_{r=1}^{n^{[x]}}\left(n^{[x]}-r+1\right)\mu^{[x]}_{\pi^{[x]}(r)}p_{\pi^{[x]}(r)} + \Gamma(k) \\
&= \sum_{x=1}^{k}\sum_{r=1}^{n^{[x]}}\left[\left(\sum_{v=x+1}^{k}n^{[v]}\right)(1+\zeta^{[x]}_{\pi^{[x]}(r)}) + \left(n^{[x]}-r+1\right)\right]\mu^{[x]}_{\pi^{[x]}(r)}p_{\pi^{[x]}(r)} \\
&\quad +\Gamma(k).
\end{aligned}$$

The above can be represented as

$$F(S_{\mathrm{B}1}(k)) = \sum_{x=1}^{k} \sum_{r=1}^{n^{[x]}} W_{\pi^{[x]}(r)}^{[x]}(r) p_{\pi^{[x]}(r)} + \Gamma(k), \tag{15.21}$$

where

$$W_{\pi^{[x]}(r)}^{[x]}(r) = \left[ \left( \sum_{v=x+1}^{k} n^{[v]} \right) (1 + \zeta_{\pi^{[x]}(r)}^{[x]}) + \left( n^{[x]} - r + 1 \right) \right] \mu_{\pi^{[x]}(r)}^{[x]},$$
$$1 \leq x \leq k, \ 1 \leq r \leq n^{[x]},$$

is a job-dependent positional weight, such that the product $W_{\pi^{[x]}(r)}^{[x]} p_{\pi^{[x]}(r)}$ represents the contribution of job $j = \pi^{[x]}(r)$ scheduled in position $r$, $1 \leq r \leq n^{[x]}$, of group $x$, $1 \leq x \leq k$, to the objective function $\sum C_j(S_{\mathrm{B}1}(k))$.

The function (15.21) admits a generic representation (12.3), so that Procedure RMP1 is in principle applicable. In particular, for each outcome of Decision (B1), i.e., for fixed values $n^{[x]}$, $1 \leq x \leq k$, of the numbers of jobs in each group, schedule $S_{\mathrm{B}1}^{*}(k)$ that corresponds to the smallest value of function (15.21) can be found by solving an LAP. Notice that the weights are job-dependent so that for each outcome of Decision (B1), the corresponding LAP will have an $n \times n$ cost matrix (see Sect. 12.4).

Thus, in order to solve problem $1\left|RMP(K), \bar{\Delta}^{[x]}\right|\sum C_j$, we apply Procedure RMP1, which involves solving a linear assignment problem (LAP) in the full form as a subroutine in Step 1(b). According to Lemma 12.1, the number of times an LAP will have to be solved is equal to $O\left(n^K\right)$. The running time required to solve an $n \times n$ LAP (see Sect. 4.1) is equal to $O\left(n^3\right)$. Thus, the following statement holds.

**Theorem 15.6** *An optimal solution for problem $1\left|RMP(K), \bar{\Delta}^{[x]}\right|\sum C_j$ can be found in $O\left(n^{K+3}\right)$ time applying Procedure RMP1 and solving a series of $n \times n$ linear assignment problems.*

Now, let us consider a special case in which the found positional weights are job-independent, i.e., when $\lambda_j^{[x]} = \lambda^{[x]}$, $1 \leq x \leq K+1$, and $\zeta_j^{[x]} = \zeta^{[x]}$, $1 \leq x \leq K$, for all $j \in N$, so that we have

$$W^{[x]}(r) = \left[ \left( \sum_{v=x+1}^{k} n^{[v]} \right) (1 + \zeta^{[x]}) + \left( n^{[x]} - r + 1 \right) \right] \mu^{[x]}, \ 1 \leq x \leq k, \ 1 \leq r \leq n^{[x]}.$$

If the positional weights are given as above, the function (15.21) admits a generic representation (12.3), so that Procedure RMP1 is in principle applicable. Since the weights are job-independent, Procedure RMP1 involves solving a linear assignment problem with a product matrix as a subroutine in Step 1(b). According to Lemma 12.1, the number of times an LAP will have to be solved is equal to $O\left(n^K\right)$. The running time required to solve an LAP with a product matrix is equal to $O(n \log n)$; see Sect. 12.4. Thus, the following statement holds.

**Theorem 15.7** *An optimal solution for problem* $1\left|RMP(K), \Delta^{[x]}\right|C_{\max}$, *with* $\lambda_j^{[x]} = \lambda^{[x]}$, $1 \le x \le K+1$, *and* $\zeta_j^{[x]} = \zeta^{[x]}$, $1 \le x \le K$, *for all* $j \in N$, *can be found in* $O\left(n^{K+1} \log n\right)$ *time applying Procedure RMP1 and solving a series of linear assignment problems with a product matrix.*

## 15.3 Bibliographic Notes

The concept of rate-modifying maintenance activities was first introduced by Lee and Leon (2001). It is described how this idea is relevant in the electronic manufacturing industry. They consider a range of problems for a model in which the processing times of jobs vary depending on whether the job is scheduled before or after the RMP. The main model uses a rate-modifying multiplier $\lambda_j$, which is job-dependent; however, simplified models with job-independent rates are also considered. The objectives considered in Lee and Leon (2001) are the makespan, the sum of the completion times, the weighted sum of the completion times, and the maximum lateness. Polynomial-time algorithms are provided for the first two objectives; however, pseudo polynomial-time algorithms are provided for the latter two. Mosheiov and Sidney (2010) extend that work and study a situation in which the duration of the RMP is not constant but is a non-decreasing linear function of its start time. They provide polynomial-time algorithms for the problems of minimizing the makespan, the sum of the completion times, the maximum lateness, the total earliness, tardiness, and due-date cost, and the number of tardy jobs. In particular, Theorem 15.1 is proved in Lee and Leon (2001) for the RMP of a constant duration, while Corollary 15.1 is proved in Mosheiov and Sidney (2010), provided that the duration of the RMP is linear and satisfies (15.3). Similarly, Theorem 15.2 is proved in Lee and Leon (2001) and in Mosheiov and Sidney (2010), for the RMP of a constant duration and of the duration given by (15.3), respectively.

Lee and Lin (2001) consider a situation in which the machine is allowed to break down if an MP is not inserted until a given instant. This instant is a random variable, with a known probability density function. If the machine is found to break down, then a repair activity must be performed, which takes considerably longer than an MP. Similar to Lee and Leon (2001), this paper also assumes that the processing times of jobs vary depending on whether the job is scheduled before or after the MP. The rate-modifying multiplier $\lambda_j$ is, however, assumed to be job-independent. The objectives considered by Lee and Lin (2001) are the makespan, the sum of the completion times, and the maximum lateness. Both resumable and non-resumable scenarios are considered.

## References

Lee C-Y, Leon VJ (2001) Machine scheduling with a rate-modifying activity. Eur J Oper Res 128:119–128

Lee C-Y, Lin C-S (2001) Single-machine scheduling with maintenance and repair rate-modifying activities. Eur J Oper Res 135:493–513

Mosheiov G, Sidney JB (2010) Scheduling a deteriorating maintenance activity on a single machine. J Oper Res Soc 61:882–887

# Chapter 16
# Scheduling with Maintenance and Positional Effects

In this chapter, we discuss single machine scheduling problems with positional effects and rate-modifying activities to minimize the makespan. Our main focus will be to explore models with both job-dependent and job-independent deterioration effects and maintenance activities. We provide a variety of solution approaches that efficiently solve different versions of the general problem. We provide a full account of the entire range of single machine problems that can be solved using the developed solution approaches.

In the problems considered in this chapter, the jobs of set $N = \{1, 2, \ldots, n\}$ are to be processed on a single machine. Each job $j \in N$ is associated with a normal processing time $p_j$. The actual processing time of the job is affected by its position in the processing sequence, as well as by its relative position with respect to the rate-modifying periods (RMPs) introduced in a schedule.

As described in Chap. 7, under a job-dependent positional effect, the actual processing time $p_j(r)$ of a job $j$ sequenced in position $r$ is given by

$$p_j(r) = p_j g_j(r),\ j \in N,\ 1 \le r \le n,$$

where $g_j(r), j \in N$, is a job-dependent positional factor. Recall that if the values $g_j(r)$, $1 \le r \le n$, form a non-decreasing sequence for each $j \in N$, we deal with a positional deterioration effect. Such a model represents a scenario in which each job wears out the machine in a different way; hence, each job $j \in N$ is associated with a unique set of positional factors. On the other hand, if the sequence is non-increasing for each $j \in N$, a learning effect is observed.

Under a job-independent positional effect, the actual processing time $p_j(r)$ of a job $j$ sequenced in position $r$ is given by

$$p_j(r) = p_j g(r),\ j \in N, 1 \le r \le n,$$

V.A. Strusevich and K. Rustogi, *Scheduling with Times-Changing Effects and Rate-Modifying Activities*, International Series in Operations Research & Management Science 243, DOI 10.1007/978-3-319-39574-6_16

where $g(r)$, $1 \le r \le n$, forms an array of positional factors that is common for all jobs. Similar to the above, if array $g(r)$, $1 \le r \le n$, is monotone non-decreasing (or non-increasing), then we have a situation of positional deterioration (or of positional learning, respectively). Assuming that the processing machines are continuously available, scheduling problems with positional effects and their generalizations are studied in Chap. 7.

In this chapter, we consider enhanced single machine scheduling models in which the processing times of the jobs are subject to positional effects and various RMPs can be introduced on the processing machine.

Consider a general situation, outlined in Sect. 12.4, in which the decision-maker is presented with a list $(\mathrm{RMP}^{[1]}, \mathrm{RMP}^{[2]}, \ldots, \mathrm{RMP}^{[K]})$ of $K \ge 1$ possible rate-modifying activities. The decision-maker may decide which of the listed RMPs to insert into a schedule and in which order. Chapter 15 presents the results on scheduling with RMPs, provided that the normal processing times of the jobs are only affected by the rate-modifying multipliers associated with the inserted RMPs.

If $k-1$ RMPs are chosen from the available $K$ options, then the jobs are divided into $k$, $1 \le k \le K+1$ groups. Depending on which RMPs are chosen and the order in which they are performed, the actual processing time of a job $j \in N$, scheduled in position $r$ of the $x$th group, can be given by

$$p_j^{[x]}(r) = p_j g_j^{[x]}(r),\ 1 \le r \le n,\ 1 \le x \le k, \tag{16.1}$$

where $g_j^{[x]}(r)$ is a job-dependent group-dependent positional factor. The presence of group-dependent positional factors implies that the actual processing time of a job is dependent on the position of the job in a group and also on the group that job is scheduled in. This is the most general positional factor known, since it allows a three-way dependency, namely on a job, a group, and a position.

If the positional effect is job-independent, the actual processing time of a job $j \in N$, scheduled in position $r$ of the $x$th group, can be given by

$$p_j^{[x]}(r) = p_j g^{[x]}(r),\ 1 \le r \le n,\ 1 \le x \le k, \tag{16.2}$$

where $g^{[x]}(r)$ is a group-dependent positional factor.

If a pure deterioration model is considered, then the positional factors within a group $x$ are in non-decreasing order

$$1 \le g_j^{[x]}(1) \le g_j^{[x]}(2) \le \cdots \le g_j^{[x]}(n),\ 1 \le x \le k,\ j \in N, \tag{16.3}$$

whereas if a pure learning model is considered, then the positional factors within a group $x$ are in non-increasing order

$$1 \ge g_j^{[x]}(1) \ge g_j^{[x]}(2) \ge \cdots \ge g_j^{[x]}(n),\ 1 \le x \le k,\ j \in N. \tag{16.4}$$

Notice that the group-dependent positional factors $g_j^{[x]}(r)$ reflect an influence that the RMPs may have on actual processing times, and as such, they include possible rate-modifying multipliers $\lambda_j^{[x]}$ that are used in the models studied in Chap. 15.

If the positional effect is job-independent, then the inequalities (16.3) and (16.4) hold without the subscript $j$, for a deterioration effect and a learning effect, respectively. As discussed in Chap. 7, it is also possible to have an arbitrary non-monotone positional effect, which could possibly arise due to a combination of deterioration and learning effects. We study such combined effects with different types of RMPs in Chap. 18.

In this chapter, we focus on the problems of minimizing the makespan under a pure deterioration effect, so that (16.3) holds, with and without job dependence. In this case, the RMPs can be understood as maintenance periods, which might be included into a schedule in order to handle the deteriorating machine conditions.

See Example 12.2 for an illustration and a description of a practical situation that corresponds to the problem with group-dependent positional effects.

Problems of minimizing the makespan with RMPs and learning effects are considered in Chap. 18 among the problems with more general scenarios. Chapter 18 also considers problems of minimizing the total completion time under positional and more general effects.

The algorithms presented in this chapter for solving relevant problems are adaptations of the generic Procedure RMP1 from Sect. 12.4.

To count the number of the related instances, we use various combinatorial configurations and identities listed in Sect. 5.3. In the estimations of the running times of the presented algorithms, we assume that the number $K$ of available RMPs is a constant, which is reasonable since usually the number of possible rate-modifying activities to be performed is fairly small.

As often is the case in this book, the LPT sequencing of the jobs is important. Recall that if the jobs are numbered in accordance with the LPT rule, then

$$p_1 \geq p_2 \geq \cdots \geq p_n. \tag{16.5}$$

## 16.1 Job-Dependent Deterioration Effects

In this section, we study job-dependent positional deterioration effects along with rate-modifying maintenance activities. The duration of an RMP included in a schedule is determined as a linear combination of the actual processing times of the jobs in the preceding group. It is assumed that in some schedule, jobs that are scheduled in group $x$, before the $x$th RMP, form set $N^{[x]}$, and the contribution of a job $j \in N^{[x]}$ toward the actual duration $\bar{\Delta}^{[x]}$ of the $x$th RMP is equal to $\zeta_j^{[x]} p_j^{[x]}(r)$. Here, $p_j^{[x]}(r)$ is the actual processing time of job $j$ scheduled in position $r$ within the $x$th group, defined as $p_j^{[x]}(r) = p_j g_j^{[x]}(r)$, where the positional factors satisfy (16.3), i.e., a deterioration effect within a group is observed. Thus, the duration of the $x$th RMP is given by

$$\bar{\Delta}^{[x]} = \sum_{j \in N^{[x]}} \zeta_j^{[x]} p_j^{[x]}(r) + \eta^{[x]}, \tag{16.6}$$

as defined in (12.2).

The problem of minimizing the makespan, under the general settings defined by (16.1) and (16.6), is denoted by $1\left|p_j^{[x]}(r) = p_j g_j^{[x]}(r), RMP(K), \bar{\Delta}^{[x]}\right|C_{\max}$, where the first term in the middle field represents the presence of job-dependent positional factors which follow (16.3), the second term points out that the list of $K$ RMPs is available for maintenance activities, and the third term indicates that the durations of the RMPs follow a rule given by (16.6).

Similar to a less general problem $1\left|RMP(K), \bar{\Delta}^{[x]}\right|C_{\max}$ studied in Sect. 15.2, an optimal solution to problem $1\left|p_j^{[x]}(r) = p_j g_j^{[x]}(r), RMP(K), \bar{\Delta}^{[x]}\right|C_{\max}$ can be found by adapting Procedure RMP1 from Sect. 12.4. In what follows, we show that problem $1\left|p_j^{[x]}(r) = p_j g_j^{[x]}(r), RMP(K), \bar{\Delta}^{[x]}\right|C_{\max}$ reduces to a linear assignment problem in its full form and can be solved in polynomial time.

### *16.1.1 Computing Positional Weights*

For problem $1\left|p_j^{[x]}(r) = p_j g_j^{[x]}(r), RMP(K), \bar{\Delta}^{[x]}\right|C_{\max}$, in accordance with Procedure RMP1 fix outcomes (A1) and (A2), and for a particular outcome of Decision (B1) introduce a schedule $S_{B1}(k)$ for an auxiliary problem $1\left|p_j^{[x]}(r) = p_j g_j^{[x]}(r), RMP(k-1), \bar{\Delta}^{[x]}\right|C_{\max}$ associated with certain outcomes of Decisions (B2) and (B3). In schedule $S_{B1}(k)$, the jobs are organized into groups $N^{[x]}$, $1 \leq x \leq k$, in which each group $N^{[x]}$ contains $n^{[x]}$ jobs, where $\sum_{x=1}^{k} n^{[x]} = n$. The jobs in $N^{[x]}$ are sequenced in accordance with a permutation $\pi^{[x]} = \left(\pi^{[x]}(1), \pi^{[x]}(2), \ldots, \pi^{[x]}\left(n^{[x]}\right)\right)$, $1 \leq x \leq k$.

It follows that in schedule $S_{B1}(k)$, the total processing time of the jobs assigned to group $x$ can be given by

$$E_x = \sum_{r=1}^{n^{[x]}} p_{\pi^{[x]}(r)} g_{\pi^{[x]}(r)}^{[x]}(r), \quad 1 \leq x \leq k.$$

In accordance with (16.6), the duration $\bar{\Delta}^{[x]}$ of the RMP scheduled after the $x$th group is given by

$$T_x = \sum_{r=1}^{n^{[x]}} \zeta_{\pi^{[x]}(r)}^{[x]} p_{\pi^{[x]}(r)} g_{\pi^{[x]}(r)}^{[x]}(r) + \eta^{[x]}, \quad 1 \leq x \leq k-1.$$

Thus, for a schedule $S_{\mathrm{B1}}(k)$, the makespan can be written as

$$\begin{aligned} C_{\max}(S_{\mathrm{B1}}(k)) &= E_1 + T_1 + E_2 + T_2 + \cdots + E_{x-1} + T_{x-1} + E_x \\ &= \sum_{x=1}^{k-1} \sum_{r=1}^{n^{[x]}} (1 + \zeta_{\pi^{[x]}(r)}^{[x]}) g_{\pi^{[x]}(r)}^{[x]}(r) p_{\pi^{[x]}(r)} \qquad (16.7) \\ &\quad + \sum_{r=1}^{n^{[k]}} g_{\pi^{[k]}(r)}^{[k]}(r) p_{\pi^{[k]}(r)} + \sum_{x=1}^{k-1} \eta^{[x]}. \end{aligned}$$

The above can be represented as

$$C_{\max}(S_{\mathrm{B1}}(k)) = \sum_{x=1}^{k} \sum_{r=1}^{n^{[x]}} W_{\pi^{[x]}(r)}^{[x]}(r) p_{\pi^{[x]}(r)} + \Gamma(k), \qquad (16.8)$$

where the constant term is defined as

$$\Gamma(k) = \sum_{x=1}^{k-1} \eta^{[x]}, \qquad (16.9)$$

and

$$W_{\pi^{[x]}(r)}^{[x]}(r) = \begin{cases} \left(1 + \zeta_{\pi^{[x]}(r)}^{[x]}\right) g_{\pi^{[x]}(r)}^{[x]}(r), & 1 \le r \le n^{[x]},\ 1 \le x \le k-1, \\ g_{\pi^{[k]}(r)}^{[k]}(r), & 1 \le r \le n^{[x]},\ x = k, \end{cases} \qquad (16.10)$$

is a job-dependent weight, such that the product $W_{\pi^{[x]}(r)}^{[x]} p_{\pi^{[x]}(r)}$ represents the contribution of job $j = \pi^{[x]}(r)$ scheduled in position $r$, $1 \le r \le n^{[x]}$, of group $x$, $1 \le x \le k$, to the objective function (16.7).

The function (16.8) admits a generic representation (12.3), and Procedure RMP1 is in principle applicable. In particular, for each outcome of Decision (B1), i.e., for fixed values $n^{[x]}$, $1 \le x \le k$, of the numbers of jobs in each group, schedule $S_{\mathrm{B1}}^{*}(k)$ that corresponds to the smallest value of function (16.7) can be found by solving a linear assignment problem (LAP). Below, we give implementation details of a less straightforward implementation of Procedure RMP1, which is based on the fact that the positional factors follow (16.3), i.e., are non-decreasing.

### 16.1.2 Reduction to Rectangular LAP

In this subsection, we describe a solution approach, which is able to solve problem $1\left|p_j^{[x]}(r) = p_j g_j^{[x]}(r), RMP(k-1), \bar{\Delta}^{[x]}\right|C_{\max}$ without prior knowledge of the number of jobs $n^{[x]}$ in each group, i.e., no specific outcome of Decision (B1) is needed. The only prerequisite condition is that the computed positional weights $W_j^{[x]}(r)$, $1 \le r \le n^{[x]}$, should be monotone within each group, i.e., should follow

$$W_j^{[x]}(1) \le W_j^{[x]}(2) \le \cdots \le W_j^{[x]}(n^{[x]}),\ 1 \le x \le k,\ j \in N. \tag{16.11}$$

It is easy to notice that because the positional factors $g_j^{[x]}(r)$ follow (16.3), the computed positional weights $W_j^{[x]}(r)$, $1 \le r \le n^{[x]}$, defined by (16.10), are non-decreasing within each group $x$, $1 \le x \le k$, for each job $j \in N$.

In the case under consideration, Step 1 of Procedure RMP1 can be simplified, and finding schedule $S^*(k)$ defined in Step 1(c) can be reduced to solving an LAP.

Define an LAP with a rectangular cost matrix that has $n$ rows, each corresponding to a job $j \in N$, and $m = nk$ columns. Number the columns by a string of the form $(x, r)$, where $x$ refers to a group, $1 \le x \le k$, and $r$, $1 \le r \le n$, indicates a position within the group. Create an $n \times m$ cost matrix $\mathbf{C} = \left(c_{j,(x,r)}\right)$ by setting $n^{[x]} = n$, $1 \le x \le k$, and computing all values of the costs $c_{j,(x,r)}$ by

$$c_{j,(x,r)} = W_j^{[x]}(r) p_j,\ 1 \le r \le n,\ 1 \le x \le k,\ j \in N, \tag{16.12}$$

where the values $W_j^{[x]}(r)$, $1 \le r \le n$, $1 \le x \le k$, come from (16.10). More precisely, the value of the element $c_{j,(x,r)}$ located at the intersection of the $j$th row and the $v$th column of matrix $\mathbf{C}$ for $v$, $1 \le v \le m$, such that $v = n(x-1) + r$, corresponds to the contribution of job $j$ to the objective function, if it is scheduled at the $r$th position of the $x$th group, where $1 \le x \le k$ and $1 \le r \le n$. Notice that matrix $\mathbf{C}$ represents a set of all possible values of $c_{j,(x,r)}$ and can be computed in $O\left(n^2 k\right)$ time.

As a result, the problem of minimizing the generic objective function (16.8) reduces to a rectangular assignment problem written as below:

$$\begin{aligned} \text{minimize} \quad & \sum_{j \in N} \sum_{x=1}^{k} \sum_{r=1}^{n} c_{j,(x,r)} z_{j,(x,r)} \\ \text{subject to} \quad & \sum_{j \in N} z_{j,(x,r)} \le 1,\ 1 \le x \le k,\ 1 \le r \le n; \\ & \sum_{x=1}^{k} \sum_{r=1}^{n} z_{j,(x,r)} = 1,\ j \in N; \\ & z_{j,(x,r)} \in \{0, 1\},\ j \in N,\ 1 \le x \le k,\ 1 \le r \le n. \end{aligned} \tag{16.13}$$

Recall that a rectangular assignment problem of the form (16.13) can be solved by Algorithm LAPD outlined in Sect. 4.1.1. The running time of this algorithm applied to a LAP with an $n \times m$ cost matrix, $m \geq n$, is $O(n^3 + nm)$. Thus, an optimal solution for problem (16.13) can be found in $O(n^3 + n^2k)$ time.

Suppose that for some $k$, $1 \leq k \leq K+1$, the solution of the assignment problem (16.13) related to an auxiliary problem $1\left|p_j^{[x]}(r) = p_j g_j^{[x]}(r), RMP(k-1), \bar{\Delta}^{[x]}\right| C_{\max}$ is found. Then, $z_{j,(x,r)} = 1$ implies that in the corresponding schedule $S^*(k)$, job $j$ is assigned to the $r$th position of group $x$. The conditions of (16.13) mean that each job will be assigned to a position and no position will be used more than once. The condition (16.11) guarantees that the found assignment admits a meaningful scheduling interpretation, because for each of the $k$ groups either several consecutive positions starting from the first are filled or the group is not used at all. In principle, the same solution approach remains valid even for the case with non-monotone positional factors, since setting $n^{[x]} = n$, $1 \leq x \leq k$, does indeed generate a set of all possible cost functions $c_{j,(x,r)}$ for the latter case as well. However, since the condition (16.11) does not hold for the case with arbitrary positional effects, it cannot be guaranteed that consecutive positions (starting from the first position) are filled in each group, thereby resulting in an infeasible solution. To ensure feasibility of the obtained solution, it is essential that the obtained positional weights should be monotonically ordered within a group.

Thus, in the case under consideration, we modify Procedure RMP1 by replacing Step 1 by the following:

**Step 1$'$.** Given problem $1\left|p_j^{[x]}(r) = p_j g_j^{[x]}(r), RMP(K), \bar{\Delta}^{[x]}\right| C_{\max}$, for each outcome (A1) and (A2), do

(a) Define $\Gamma(k)$ by (16.9) and compute positional weights $W_j^{[x]}(r), j \in N$, $1 \leq r \leq n$, by (16.10) applied with $n^{[x]} = n$, $1 \leq x \leq k$.
(b) Create the cost matrix $\mathbf{C} = \left(c_{j,(x,r)}\right)$ with the elements (16.12).
(c) Find schedule $S^*(k)$ by solving the linear assignment problem (16.13).

The following statement holds.

**Lemma 16.1** *An auxiliary problem* $1\left|p_j^{[x]}(r) = p_j g_j^{[x]}(r), RMP(k-1), \bar{\Delta}^{[x]}\right| C_{\max}$ *under a positional deterioration effect (16.3) can be solved in* $O(n^3 + n^2k)$ *time by reduction to a rectangular LAP of the form (16.13).*

In accordance with the modified Procedure RMP1, in order to obtain a schedule $S^*$ that is optimal for the original problem $1\left|p_j^{[x]}(r) = p_j g_j^{[x]}(r), RMP(K), \bar{\Delta}^{[x]}\right| C_{\max}$, we need to generate all possible outcomes (A1) and (A2), to find schedule $S^*(k)$ for each generated auxiliary problem $1\left|p_j^{[x]}(r) = p_j g_j^{[x]}(r), RMP(k-1), \bar{\Delta}^{[x]}\right| C_{\max}$ and identify the best of the found schedules.

As proved in Lemma 12.1, the number of all possible outcomes (A1) and (A2) can be estimated as $\sum_{k=1}^{K+1} K^{k-1}$. Since for a $k$, $1 \leq k \leq K+1$, solving the corresponding rectangular LAP requires $O\left(n^3 + n^2k\right)$ time, the total running time required to solve the general version of problem $1\left|p_j^{[x]}(r) = p_j g_j^{[x]}(r), RMP(K), \bar{\Delta}^{[x]}\right|C_{\max}$ by the modified Procedure RMP1 can be estimated as $O\left(n^3 K^K + n^2 \sum_{k=1}^{K+1} kK^{k-1}\right)$. Due to

$$\sum_{k=1}^{K+1} kK^{k-1} = \frac{1}{(K-1)^2} + \frac{1}{K} K^{K+2} \frac{K^2 - 2}{(K-1)^2} = O\left(K^{K+1}\right),$$

the overall running time can be written as $O(n^3 K^K + n^2 K^{K+1}) = O\left(n^2 K^K (n + K)\right)$, which is polynomial in $n$ for a constant $K$.

**Theorem 16.1** *Problem* $1\left|p_j^{[x]}(r) = p_j g_j^{[x]}(r), RMP(K), \bar{\Delta}^{[x]}\right|C_{\max}$ *under a positional deterioration effect (16.3) can be solved in* $O\left(n^2 K^K (n + K)\right)$ *time by reduction to* $O\left(K^K\right)$ *rectangular linear assignment problems.*

Notice that the running time stated in Theorem 16.1 may be reduced if we consider a less general situation, in which some of RMP Decisions do not need to be taken. Let us evaluate the number of different combinations for RMP Decisions 1–3 that are needed to solve less general versions of the problem. We may consider in total $2^3 = 8$ versions of problem $1\left|p_j^{[x]}(r) = p_j g_j^{[x]}(r), RMP(K), \bar{\Delta}^{[x]}\right|C_{\max}$, distinguishing them based on the three criteria:

(a) The RMPs are identical or distinct;
(b) Deterioration factors are group-independent or group-dependent, and
(c) Durations of the RMPs are constant or defined by (16.6).

Notice that the model with the RMPs of constant duration is a special case of (16.6) with all coefficients $\zeta_j^{[x]}$ equal to zero. More specifically, the duration of an RMP$^{[y]}$, $1 \le y \le K$, in the given list is equal to $\eta^{[y]}$.

For each version, an individual auxiliary problem $1\left|p_j^{[x]}(r) = p_j g_j^{[x]}(r), RMP(K), \bar{\Delta}^{[x]}\right|C_{\max}$ is solved by reduction to a rectangular LAP, as discussed above. Table 16.1 states the numbers of possible outcomes (A1) and (A2), i.e., the number of auxiliary problems $1\left|p_j^{[x]}(r) = p_j g_j^{[x]}(r), RMP(k-1), \bar{\Delta}^{[x]}\right|C_{\max}$ that are needed to be solved in order to solve a version of problem $1\left|p_j^{[x]}(r) = p_j g_j^{[x]}(r), RMP(K), \bar{\Delta}^{[x]}\right|C_{\max}$. In the rows of Table 16.1 and several other tables presented later in this chapter, we write either GI or GD, depending on whether the deterioration factors are group-independent or group-dependent, respectively.

Notice that if all available RMPs are identical, then only RMP Decision 1 must be taken, i.e., only an optimal number of the RMPs in the schedule has to be determined. Thus, problem $1\left|p_j^{[x]}(r) = p_j g_j^{[x]}(r), RMP(k-1), \bar{\Delta}^{[x]}\right|C_{\max}$ must be solved $K + 1$ times. In this case, the running time required to solve problem

**Table 16.1** Number of subproblems to solve for different versions of problem $1\left|p_j^{[x]}(r) = p_j g_j^{[x]}(r), RMP(K), \Delta_j^{[x]}(\tau)\right|C_{\max}$

| | Constant duration RMP | | Variable duration RMP | |
|---|---|---|---|---|
| | Identical | Distinct | Identical | Distinct |
| GI | $K+1$ | $K+1$ | $K+1$ | $2^K$ |
| GD | $K+1$ | $\sum_{k=1}^{K+1}\binom{K}{k-1}(k-1)!$ | $K+1$ | $\sum_{k=1}^{K+1}\binom{K}{k-1}(k-1)!$ |

$1\left|p_j^{[x]}(r) = p_j g_j^{[x]}(r), RMP(K), \bar{\Delta}^{[x]}\right|C_{\max}$ can be estimated as $O\left(n^3(K+1) + \sum_{k=1}^{K+1} n^2 k\right) = O\left(n^2 K(n+K)\right)$.

Now, consider the situation when the RMPs are distinct, but the deterioration effects are group-independent. This problem corresponds to a scenario, in which distinct RMPs are performed in the schedule, but they all restore the machine to the same state. Thus, the order of the RMPs is irrelevant (i.e., any outcome of RMP Decision 3 may be accepted), and the decision regarding the choice of the RMPs to be included into a schedule (RMP Decision 2) is made only on the basis of the durations of the RMPs.

If the durations of the RMPs are constant, then for each $k$, $1 \le k \le K+1$, we need to take $k-1$ RMPs with the smallest values of $\eta^{[y]}$, which can be found in $O(K \log K)$ time by sorting and renumbering the list of the RMPs in non-decreasing order of their durations, so that

$$\eta^{[1]} \le \eta^{[2]} \le \cdots \le \eta^{[K]}. \tag{16.14}$$

holds. Thus, if RMP Decision 1 is assumed to be taken, so that $k-1$ RMPs are to be included in the schedule, $1 \le k \le K+1$, then $\mathrm{RMP}^{[1]}, \ldots, \mathrm{RMP}^{[k-1]}$ are to be chosen and inserted into a schedule in the order of their numbering. Finally, $K+1$ options need to be evaluated in order to make RMP Decision 1.

On the other hand, if the durations are determined by (16.6), there is no easy way of selecting the best $k-1$ RMPs from the $K$ available RMPs. Thus, all possible selections need to be tried and this can be done in $\binom{K}{k-1}$ ways. Trying all possible values of $k$, $1 \le k \le K+1$ (i.e., taking RMP Decision 1), the total number of options can be estimated by $\sum_{k=1}^{K+1}\binom{K}{k-1} = 2^K$. Thus, the total running time required to solve this version of problem $1\left|p_j^{[x]}(r) = p_j g_j^{[x]}(r), RMP(K), \bar{\Delta}^{[x]}\right|C_{\max}$ can be estimated as $O\left(n^3 2^K + \sum_{k=1}^{K+1} n^2 k\binom{K}{k-1}\right) = O\left(n^2 2^K(n+K)\right)$.

The corresponding running times required to solve different versions of problem $1\left|p_j^{[x]}(r) = p_j g_j^{[x]}(r), RMP(K), \bar{\Delta}^{[x]}\right|C_{\max}$ are given in Table 16.2.

**Table 16.2** Running time required for different versions of problem $1\left|p_j^{[x]}(r) = p_j g_j^{[x]}(r), RMP(K), \Delta_j^{[x]}(\tau)\right|C_{\max}$

| | Constant duration RMP | | Variable duration RMP | |
|---|---|---|---|---|
| | Identical | Distinct | Identical | Distinct |
| GI | $O(n^2K(n+K))$ | $O(n^2K(n+K))$ | $O(n^2K(n+K))$ | $O(n^2 2^K(n+K))$ |
| GD | $O(n^2K(n+K))$ | $O(n^2K^K(n+K))$ | $O(n^2K(n+K))$ | $O(n^2K^K(n+K))$ |

## 16.2 Job-Independent Effects

In this section, we study job-independent positional deterioration effects along with rate-modifying maintenance activities. The duration of an RMP is given by

$$\Delta^{[y]}(\tau) = \zeta^{[y]}\tau + \eta^{[y]}, \tag{16.15}$$

as introduced in (12.1), where $\tau$ is the starttime of the RMP, measured either from time zero in the case of the first RMP or from the completion time of the previous RMP. Notice that the duration given by (16.15) is simply the job-independent version of (16.6).

The problem of minimizing the makespan, under the general settings defined by (16.2) and (16.15), can be denoted by $1\left|p_j^{[x]}(r) = p_j g^{[x]}(r), RMP(K), \Delta^{[x]}(\tau)\right|C_{\max}$, where the first term in the middle field represents the presence of job-independent positional factors which follow (16.3), the second term points out that $K$ RMPs are available for maintenance activities, and the third term points out that the duration of the RMPs follows a job-independent rule as given by (16.15).

An optimal solution to problem $1\left|p_j^{[x]}(r) = p_j g^{[x]}(r), RMP(K), \Delta^{[x]}(\tau)\right|C_{\max}$ can be found by adapting Procedure RMP1 from Sect. 12.4. In what follows, we show that problem $1\left|p_j^{[x]}(r) = p_j g^{[x]}(r), RMP(K)\Delta^{[x]}(\tau)\right|C_{\max}$ reduces to a series of linear assignment problems with a product matrix and can be solved in polynomial time.

### 16.2.1 Computing Positional Weights

For problem $1\left|p_j^{[x]}(r) = p_j g^{[x]}(r), RMP(K), \Delta^{[y]}(\tau)\right|C_{\max}$, in accordance with Procedure RMP1 fix outcomes (A1) and (A2), and for a particular outcome of Decision (B1), introduce a schedule $S_{\mathrm{B1}}(k)$ for an auxiliary problem $1\left|p_j^{[x]}(r) = p_j g^{[x]}(r), RMP(k-1), \Delta^{[y]}(\tau)\right|C_{\max}$ associated with certain outcomes of Decisions (B2) and (B3). In schedule $S_{\mathrm{B1}}(k)$, the jobs are organized into groups $N^{[x]}$,

$1 \le x \le k$, and each group $N^{[x]}$ contains $n^{[x]}$ jobs, where $\sum_{x=1}^{k} n^{[x]} = n$. The jobs in $N^{[x]}$ are sequenced in accordance with a permutation $\pi^{[x]} = \left(\pi^{[x]}(1), \pi^{[x]}(2), \ldots, \pi^{[x]}\left(n^{[x]}\right)\right)$, $1 \le x \le k$.

Computing the value of the makespan for schedule $S_{\mathrm{B1}}(k)$ is quite similar to that presented in Sect. 16.1.1. It follows that for a schedule $S_{\mathrm{B1}}(k)$, the makespan can be written as

$$C_{\max}(S_{\mathrm{B1}}(k)) = \sum_{x=1}^{k-1} \sum_{r=1}^{n^{[x]}} (1+\zeta^{[x]}) g^{[x]}(r) p_{\pi^{[x]}(r)} + \sum_{r=1}^{n^{[k]}} g^{[k]}(r) p_{\pi^{[k]}(r)} + \sum_{x=1}^{k-1} \eta^{[x]}. \tag{16.16}$$

The above can be represented as a generic objective function of the form

$$C_{\max}(S_{\mathrm{B1}}(k)) = \sum_{x=1}^{k} \sum_{r=1}^{n^{[x]}} W^{[x]}(r) p_{\pi^{[x]}(r)} + \Gamma(k), \tag{16.17}$$

where $\Gamma(k)$ is defined by (16.9) and

$$W^{[x]}(r) = \begin{cases} \left(1+\zeta^{[x]}\right) g^{[x]}(r), & 1 \le r \le n^{[x]},\ 1 \le x \le k-1, \\ g^{[k]}(r), & 1 \le r \le n^{[x]},\ x = k, \end{cases} \tag{16.18}$$

is a job-independent weight, such that the product $W^{[x]}_{\pi^{[x]}(r)} p_{\pi^{[x]}(r)}$ represents the contribution of job $j = \pi^{[x]}(r)$ scheduled in position $r$, $1 \le r \le n^{[x]}$, of group $x$, $1 \le x \le k$, to the objective function (16.16).

The function (16.8) admits a generic representation (12.3), and Procedure RMP1 is in principle applicable. Notice that the weights are job-independent so that for each outcome of Decision (B1), the corresponding linear assignment problem will have a product cost matrix(see Sect. 12.4).

Below, we consider eight versions of problem $1\left|p_j^{[x]}(r) = p_j g^{[x]}(r), RMP(K), \Delta^{[y]}(\tau)\right|C_{\max}$ distinguishing between them based on three criteria (a), (b), and (c) listed in Sect. 16.1.2. We present three solution approaches, which handle different versions of problem $1\left|p_j^{[x]}(r) = p_j g^{[x]}(r), RMP(K), \Delta^{[x]}(\tau)\right|C_{\max}$. All three approaches are able to take Decision (B1), i.e., to find the optimal values of $n^{[x]}$, $1 \le x \le k$, on the fly. The approaches differ in how outcomes (A1) and (A2) are generated. Based on these differences, we study different versions of the main problem in three separate subsections. The three solution approaches require different running times, but they all use a subroutine which implements the ideas of Algorithm Match, i.e., the corresponding optimal permutation of jobs is obtained by matching jobs with large normal processing times to small positional weights.

Table 16.3 lists the different versions of problem $1\left|p_j^{[x]}(r) = p_j g^{[x]}(r), RMP(K), \Delta^{[x]}(\tau)\right|C_{\max}$ considered in this chapter and gives references to the appropriate sections.

**Table 16.3** Different versions of problem $1\left|p_j^{[x]}(r) = p_j g^{[x]}(r), RMP(K), \Delta^{[x]}(\tau)\right|C_{\max}$

| | Constant duration RMP | | Variable duration RMP | |
|---|---|---|---|---|
| | Identical | Distinct | Identical | Distinct |
| GI | Sect. 16.2.4 | Sect. 16.2.4 | Sect. 16.2.3 | Sect. 16.2.2 |
| GD | Sect. 16.2.3 | Sect. 16.2.2 | Sect. 16.2.2 | Sect. 16.2.2 |

## 16.2.2 Reduction to LAP with a Product Matrix

Let us begin our consideration with the most general version of problem $1\left|p_j^{[x]}(r) = p_j g^{[x]}(r), RMP(K), \Delta^{[x]}(\tau)\right|C_{\max}$, with group-dependent deterioration factors and distinct RMPs of start-time-dependent durations. We describe a solution approach, which is able to solve the auxiliary problem $1\left|p_j^{[x]}(r) = p_j g^{[x]}(r), RMP(k-1), \Delta^{[x]}(\tau)\right|C_{\max}$ without prior knowledge of the number $n^{[x]}$ of jobs in each group. Similar to (16.11), we require a prerequisite condition that computed positional weights $W^{[x]}(r)$, $1 \le r \le n^{[x]}$, should follow

$$W^{[x]}(1) \le W^{[x]}(2) \le \cdots \le W^{[x]}(n^{[x]}),\ 1 \le x \le k. \tag{16.19}$$

It is easy to notice that because the positional factors $g^{[x]}(r)$ follow (16.3), the computed positional weights $W^{[x]}(r)$, $1 \le r \le n^{[x]}$, are non-decreasing within each group $x$, $1 \le x \le k$.

**Theorem 16.2** *Given an auxiliary problem* $1\left|p_j^{[x]}(r) = p_j g^{[x]}(r), RMP(k-1), \Delta^{[x]}(\tau)\right|C_{\max}$*, where* $1 \le k \le K+1$*, compute all possible positional weights* $W^{[x]}(r)$*,* $1 \le r \le n$*,* $1 \le x \le k$*, using (16.18) with* $n^{[x]} = n$ *and choose the* $n$ *smallest among them. For a particular group, if the chosen elements occupy consecutive positions, then assigning the jobs with the largest normal processing times to the positions associated with the smallest positional weights will create a schedule* $S^*(k)$ *which minimizes the objective function (16.17).*

*Proof* The proof of the theorem is rather straightforward. Notice that at most $n$ positions can be used in each group. In a schedule with $k$ groups, we have a choice of at most $nk$ positions, in which $n$ jobs can be potentially scheduled. Each of these positions has a certain positional weight associated with them. Recall that the contribution of a job $j = \pi^{[x]}(r)$ to the objective function $C_{\max}(S(k))$ is given by $W^{[x]}(r)p_j$. Thus, following from Sect. 2.1, in order to ensure the smallest value of the objective function, we must choose $n$ positions that generate the smallest positional weights. Since all possible values of $W^{[x]}(r)$ are known, we can identify these positions. Moreover, due to (16.19), it can be ensured that for each group, the found positions form a block of consecutive positions starting from position 1. These positions may be used to create a feasible schedule. For a group $x$, the number of chosen positions gives us the value $n^{[x]}$, $1 \le x \le k$. □

We now apply Theorem 16.2 to finding an optimal solution $S^*(k)$ to problem $1\left|p_j^{[x]}(r) = p_j g^{[x]}(r), RMP(k-1), \Delta^{[x]}(\tau)\right|C_{\max}$. This can be done by the following algorithm. In the description of the algorithm, it is assumed that the jobs are numbered in the LPT order (16.5).

**Algorithm NSmallPosi**

INPUT: An instance of problem $1\left|p_j^{[x]}(r) = p_j g^{[x]}(r), RMP(k-1), \Delta^{[x]}(\tau)\right|C_{\max}$

OUTPUT: An optimal schedule $S^*(k)$ defined by the processing sequences $\pi^{[x]}$, $1 \le x \le k$

```
Step 1. For each group $x$, $1 \le x \le k$, define an empty processing sequence
    $\pi^{[x]} := (\varnothing)$ and the weight $W^{[x]} := W^{[x]}(1)$ computed as in (16.18). Create a non-
    decreasing list $\Omega$ of the values $W^{[x]}$, $1 \le x \le k$; to break ties, place the weight
    associated with a group with a smaller index $x$ earlier in the list.
Step 2. For each job $j$ from 1 to $n$, do
    (a) Take the first element $W^{[v]}$ in list $\Omega$, the smallest available positional weight.
    (b) Assign job $j$ to group $v$ and place it after the current permutation $\pi^{[v]}$,
        i.e., update $\pi^{[v]} := (\pi^{[v]}, j)$ and associate job $j$ with the positional weight
        $W^{[v]}$. Remove $W^{[v]}$ from list $\Omega$. Compute $r = \left|\pi^{[v]}\right|$. Use (16.18) to update
        $W^{[v]} := W^{[v]}(r+1)$, and insert the updated value $W^{[v]}$ into $\Omega$, maintaining
        list $\Omega$ non-decreasing.
Step 3. The found permutation $\pi^* = \left(\pi^{[1]}, \pi^{[2]}, \dots, \pi^{[k]}\right)$ defines an optimal
    schedule $S^*(k)$. Compute the optimal value of the objective function $C_{\max}(S^*(k))$
    by (16.17).
```

In Step 1 of Algorithm NSmallPosi, list $\Omega$ can be created in $O(k \log k)$ time. Each iteration of the loop in Step 2 requires $O(\log k)$ time, since the insertion of the updated weight into a sorted list can be done by binary search. Notice that the total number of the weights computed in Steps 1 and 2 does not exceed $n + k$. Since $n \ge k$, the following statement holds.

**Lemma 16.2** *For an auxiliary problem* $1\left|p_j^{[x]}(r) = p_j g^{[x]}(r), RMP(k-1), \Delta^{[x]}(\tau)\right|C_{\max}$ *under a positional job-independent deterioration effect, Algorithm NSmallPosi finds an optimal schedule* $S^*(k)$ *in* $O(n \log n)$ *time, or in* $O(n \log k)$ *time, provided that the LPT sequence of the jobs is known.*

Thus, in the case under consideration, we modify Procedure RMP1 by replacing Step 1(b) and (c) by the following:

**Step 1(b′)**. For each outcome of Decision B(1), do

Find schedule $S^*(k)$ by applying Algorithm NSmallPosi.

**Table 16.4** Running time required for diffrent versions of problem $1\left|p_j^{[x]}(r)=p_jg^{[x]}(r),\right.$ $RMP(K),\ \Delta^{[x]}(\tau)\left|C_{\max}\right.$

| | Constant duration RMP | | Variable duration RMP | |
|---|---|---|---|---|
| | Identical | Distinct | Identical | Distinct |
| GI | Sect. 16.2.4 | Sect. 16.2.4 | Sect. 16.2.3 | $O\left(n2^K\log K\right)$ |
| GD | Sect. 16.2.3 | $O\left(nK^K\log K\right)$ | $O(nK\log K)$ | $O\left(nK^K\log K\right)$ |

As proved in Lemma 12.1, the number of all possible outcomes (A1) and (A2) can be estimated as $\sum_{k=1}^{K+1}K^{k-1}$. Since Algorithm NSmallPosi requires $O(n\log k)$ time to run for a given $k$, $1\le k\le K+1$, the total running time required to solve the most general version of problem $1\left|p_j^{[x]}(r)=p_jg^{[x]}(r),\ RMP(K),\ \Delta^{[x]}(\tau)\right|C_{\max}$ can be estimated as $O\left(n\sum_{k=1}^{K+1}K^{k-1}\log k\right)=O\left(nK^K\log K\right)$, which is linear in $n$ for a constant $K$.

Now, let us consider other less general versions of problem $1\left|p_j^{[x]}(r)=p_jg^{[x]}(r),\right.$ $RMP(K),\ \Delta^{[x]}(\tau)\left|C_{\max}\right.$. Similar to Sect. 16.1.2, we consider 8 versions of problem $1\left|p_j^{[x]}(r)=p_jg^{[x]}(r),RMP(K),\ \Delta^{[x]}(\tau)\right|C_{\max}$. For each version, the auxiliary problem $1\left|p_j^{[x]}(r)=p_jg^{[x]}(r),RMP(K),\ \Delta^{[x]}(\tau)\right|C_{\max}$ is solved in $O(n\log k)$ time by applying Algorithm NSmallPosi. Again, similar to the job-dependent version of the problem, Table 16.1 states the number of times the auxiliary problem $1\left|p_j^{[x]}(r)=p_jg^{[x]}(r),RMP(k-1),\ \Delta^{[x]}(\tau)\right|C_{\max}$ must be solved in order to solve different versions of problem $1\left|p_j^{[x]}(r)=p_jg^{[x]}(r),RMP(K),\right.$ $\Delta^{[x]}(\tau)\left|C_{\max}\right.$.

The corresponding running times required to solve different versions of problem $1\left|p_j^{[x]}(r)=p_jg^{[x]}(r),RMP(K),\ \Delta^{[x]}(\tau)\right|C_{\max}$ are given in Table 16.4. Notice that although Algorithm NSmallPosi is able to solve all eight versions of problem $1\left|p_j^{[x]}(r)=p_jg^{[x]}(r),RMP(K),\ \Delta^{[x]}(\tau)\right|C_{\max}$, for some cases, it is possible to make RMP Decisions 1–3 on the fly by using another solution approach, which allows the optimal solution to be found faster. For such cases, a reference to the relevant section is made in Table 16.4.

Below, we present examples of situations in which the outlined approach based on Algorithm NSmallPosi is not applicable.

First, the approach cannot be used for the problem $1\left|p_j^{[x]}(r)=p_jg^{[x]}(r),\right.$ $RMP\,(K),\ \Delta^{[x]}(\tau)\left|C_{\max}\right.$ of minimizing the makespan with non-monotone positional factors. Although setting $n^{[x]}=n.\ 1\le x\le k$ does indeed generate a set of all possible positional weights $W^{[x]}(r)$, still the condition (16.19) does not necessarily hold, and it cannot be guaranteed that the $n$ smallest positional weights come from the consecutive positions (starting from the first position) of a group, thereby resulting in an infeasible solution.

Second, the approach cannot be extended to minimizing another objective function, i.e., the total completion time. For illustration, take problem $1\left|p(r) = p_j r^a,\right.$ $RMP(K),\ \Delta(\tau)\left|\sum C_j\right.$ with a group-independent polynomial deterioration effect, i.e., $g^{[x]}(r) = r^a$, $a > 0$, and $K$ identical maintenance periods that have start-time-dependent durations, i.e., $\zeta^{[x]} = \zeta$ and $\eta^{[x]} = \eta$, $1 \le x \le K$. It can be proved that for this problem, solving an auxiliary problem $1\left|p(r) = p_j r^a, RMP(k-1),\ \Delta(\tau)\right|$ $\sum C_j$ reduces to minimizing a generic function of the form (16.17) with the positional weights

$$W^{[x]}(r) = \begin{cases} \left[\left(n - \sum_{v=1}^{x} n^{[v]}\right)(1+\zeta) + \left(n^{[x]} - r + 1\right)\right] r^a, & 1 \le r \le n^{[x]},\ 1 \le x \le k-1, \\ \left(n^{[x]} - r + 1\right) r^a & 1 \le r \le n^{[x]},\ x = k. \end{cases} \tag{16.20}$$

Notice that for the positional weights defined above, it is not possible to generate a set of all possible values of $W^{[x]}(r)$, $1 \le r \le n^{[x]}$, $1 \le x \le k$, without prior knowledge of the number $n^{[x]}$ of jobs in each group. Thus, Theorem 16.2 does not hold, and as a result, Algorithm NSmallPosi cannot be used.

We revisit both situations in Chap. 18 and solve the corresponding problems and their generalizations. In particular, problem $1\left|p(r) = p_j r^a, RMP(k-1),\ \Delta(\tau)\right|$ $\sum C_j$ will be proved solvable in $O\left(n^k \log n\right)$ time by full enumeration of possible outcomes of Decision (B1), i.e., of all possible values of $n^{[x]}$, $1 \le x \le k$.

### *16.2.3 On the Fly Decision Making*

In this subsection, we present a solution approach that allows making RMP Decisions 1–3 on the fly, without enumerating all possible options. This gives the desired outputs (A1) and (A2) and helps us to considerably reduce the running time for solving variants of the original problem $1\left|p_j^{[x]}(r) = p_j g^{[x]}(r), RMP(K),\ \Delta^{[x]}(\tau)\right|$ $C_{\max}$ listed below.

**Problem Posi1:** This is a version of problem $1\left|p_j^{[x]}(r) = p_j g^{[x]}(r), RMP(K),\right.$ $\Delta^{[x]}(\tau)\left|C_{\max}\right.$ with group-dependent deterioration factors and distinct RMPs of constant durations. Assume that for each RMP$^{[y]}$ from a given list, $\zeta^{[y]} = 0$, $1 \le y \le K$, holds and the RMPs are numbered in non-decreasing order of their durations, i.e., (16.14) holds. The positional factors $g^{[1]}(r)$, $1 \le r \le n$, are associated with jobs of the first group that is created before the first RMP. If a certain RMP$^{[y]}$, $1 \le y \le K$, is included in the schedule, then the jobs of the group that follows that RMP are associated with the positional factors $g^{[y+1]}(r)$, $1 \le r \le n$. Under the numbering of the RMPs given by (16.14), the factors associated with a certain position in group form a non-decreasing sequence, i.e.,

$$g^{[1]}(r) \le g^{[2]}(r) \le \cdots \le g^{[K+1]}(r),\ 1 \le r \le n. \tag{16.21}$$

Problem Posi1 is a generalization of one of the cases found in Table 16.4, in which group-dependent deterioration factors are considered along with identical RMPs of constant duration, i.e., $\zeta^{[y]} = 0$, $\eta^{[y]} = \eta$, $1 \le y \le K$. For the latter problem, it is reasonable to assume that (16.21) and (16.14) hold simultaneously, based on the following argument. If identical RMPs of equal duration are performed on the machine, then after an RMP the condition of the machine can be no better than its condition after the previous RMP. In such a case, every position (including the first position) in a group after an RMP will have a worse deterioration factor than its counterpart in an earlier group.

**Problem Posi2:** This is a version of problem $1\left|p_j^{[x]}(r) = p_j g(r), RMP(K), \Delta^{[y]}(\tau)\right|C_{\max}$ with group-independent deterioration factors, i.e., $g^{[x]}(r) = g(r)$, $1 \le r \le n$, $1 \le x \le K+1$, and distinct RMPs with start-time-dependent durations with agreeable parameters, i.e., $\zeta^{[y_1]} \le \zeta^{[y_2]}$ implies $\eta^{[y_1]} \le \eta^{[y_2]}$ for every pair of the RMP indices $y_1$ and $y_2$. Assume that the given $K$ RMPs are numbered in such a way that

$$\zeta^{[1]} \le \zeta^{[2]} \le \cdots \le \zeta^{[K]} \tag{16.22}$$

and (16.14) hold simultaneously. This version is a generalization of one of the cases found in Table 16.4, in which group-independent positional factors are considered along with identical RMPs of start-time-dependent duration, i.e., $\zeta^{[x]} = \zeta$, $\eta^{[x]} = \eta$, $1 \le x \le K$. The latter problem corresponds to a scenario in which identical RMPs are performed in the schedule and they all restore the machine to the same state.

In order to solve both versions of problem $1\left|p_j^{[x]}(r) = p_j g^{[x]}(r), RMP(K), \Delta^{[x]}(\tau)\right|C_{\max}$ described above, an optimal choice for RMP Decisions 2 and 3 can be made easily. If RMP Decision 1 is assumed to be taken, so that $k-1$, $1 \le k \le K+1$, RMPs are to be included in the schedule, then for both problems, RMP$^{[1]}$, …, RMP$^{[k-1]}$ are to be chosen and inserted into a schedule in the order of their numbering. For a chosen $k$, this gives the best outputs (A1) and (A2) for Problem Posi1, since the chosen RMPs have the shortest durations, and moreover, the created groups will be associated with the smallest deterioration factors, since (16.21) and (16.14) hold simultaneously. This gives the best outputs (A1) and (A2) for Problem Posi2 as well, since the groups are identical from the point of view of the associated positional factors, and the RMPs with smaller indices have smaller values of the duration parameters, because (16.22) and (16.14) hold simultaneously.

For an assumed value of $k$, let RMP Decisions 2–3 be made, and let problem $1\left|p_j^{[x]}(r) = p_j g^{[x]}(r), RMP(k-1), \Delta^{[x]}(\tau)\right|C_{\max}$ be the resulting auxiliary problem. This problem can be solved by minimizing the generic objective function of the form (16.17). For Problem Posi1, obtain the required positional weights $W^{[x]}(r)$ by substituting $\zeta^{[x]} = 0$, $1 \le x \le k$, in (16.18) so that

$$W^{[x]}(r) = g^{[x]}(r),\ 1 \le r \le n^{[x]},\ 1 \le x \le k, \tag{16.23}$$

while for Problem Posi2, substitute $g^{[x]}(r) = g(r)$, $1 \le r \le n^{[x]}$, $1 \le x \le k$, so that

$$W^{[x]}(r) = \begin{cases} (1+\zeta^{[x]})g(r), & 1 \le r \le n^{[x]},\ 1 \le x \le k-1, \\ g(r) & 1 \le r \le n^{[x]},\ x = k. \end{cases} \tag{16.24}$$

Take $k = K + 1$ and define $n^{[x]} := n$, $1 \le x \le k$. Compute all positional weights $W^{[x]}(r)$, $1 \le r \le n$, $1 \le x \le K + 1$, for each problem by using the formulae above. Notice that the computed positional weights represent a list of all possible values of $W^{[x]}(r)$ across all possible groups.

**Definition 16.1** If for each auxiliary problem $1\left|p_j^{[x]}(r) = p_j g^{[x]}(r),\ RMP(k-1),\ \Delta^{[x]}(\tau)\right|C_{\max}$, $1 \le k \le K + 1$, associated with problem $1\left|p_j^{[x]}(r) = p_j g^{[x]}(r),\ RMP(K),\ \Delta^{[x]}(\tau)\right|C_{\max}$, for each possible position $r$ the positional weight in group $x$ does not exceed the positional weight in the same position in another group $y$, group $x$ is said to *dominate* group $y$, $1 \le x \le k$, $1 \le y \le k$, $x \ne y$. If all available groups can be linearly ordered with respect to the introduced dominance relation, then problem $1\left|p_j^{[x]}(r) = p_j g^{[x]}(r), RMP(K),\ \Delta^{[x]}(\tau)\right|C_{\max}$ is said to satisfy the "*K-domi*" *condition.*

Both Problems Posi1 and Posi2 satisfy the *K-domi* condition. For Problem Posi1, due to (16.21), the positional weights associated with a position $r$ are ordered so that for each $k$, $1 \le k \le K + 1$, we have

$$W^{[1]}(r) \le W^{[2]}(r) \le \cdots \le W^{[k]}(r),\ 1 \le r \le n,$$

while for Problem Posi2, due to (16.22), the positional weights are ordered so that for each $k$, $1 \le k \le K + 1$, we have

$$W^{[k]}(r) \le W^{[1]}(r) \le W^{[2]}(r) \le \cdots \le W^{[k-1]}(r), 1 \le r \le n.$$

These observations guarantee the required dominance for any pair of groups in any of these two problems.

For versions of problem $1\left|p_j^{[x]}(r) = p_j g^{[x]}(r), RMP(K),\ \Delta^{[x]}(\tau)\right|C_{\max}$ that simultaneously satisfy (16.14) and the *K-domi* condition, it is possible to take all RMP Decisions 1–3 and to compute the optimal values of $n^{[x]}$, $1 \le x \le k$, on the fly. Recall that for Problems Posi1 and Posi2, for a fixed outcome of RMP Decision 1, we know the optimal outcomes of RMP Decisions 2 and 3. Thus, in order to solve these problems, we apply the following methodology that is based on Theorem 16.2.

Similar to Algorithm NSmallPosi from Sect. 16.2.2, the method presented below also finds an optimal schedule $S^*(k)$ with $k$ groups, $1 \le k \le K + 1$, by searching for the $n$ smallest positional weights and assigns the jobs with the largest values of $p_j$ to the positions corresponding to the smallest positional weights. The main difference between the two algorithms lies in the way the list of the $n$ smallest positional weights is found. For each $k$, $1 \le k \le K+1$, Algorithm NSmallPosi searches for the $n$ smallest positional weights by comparing $nk$ positional weights across $k$ groups. On the other hand, for each $k$, $2 \le k \le K+1$, the method presented below searches for the $n$ smallest positional weights by comparing the $n$ positional weights used in schedule $S^*(k-1)$ and the $n$ positional weights that are obtained, if the $k$th group is opened. As a result, the problem reduces to finding the $n$ smallest elements from at most $2n$ candidates, as opposed to $nk$ candidates.

Formally, the approach outlined above is implemented by manipulating two lists, which we denote by $G(k-1)$ and $H(k)$. List $G(k-1)$ contains all the positional weights corresponding to the positions used in the previously found schedule $S^*(k-1)$, while list $H(k)$ contains the positional weights that will be introduced if the $k$th group is opened.

List $H(v)$, $1 \le v \le K+1$, is defined differently for Problems Posi1 and Posi2. For Problem Posi1, $H(v)$ contains the positional weights $W^{[v]}(r)$, $1 \le r \le n - v + 1$, for $n^{[x]} = n$, so that by (16.23), we have $H(v) := \left(g^{[v]}(1), g^{[v]}(2), \dots, g^{[v]}(n-v+1)\right)$, $1 \le v \le K+1$. For Problem Posi2, notice that the values of the positional weights given by (16.24) change dynamically as the value of $k$ is changed. Thus, we define $H(v)$ so that this effect is incorporated; initialize $H(1) := (g(1), g(2), \dots, g(n))$ and define $H(v) := \left(\left(1+\zeta^{[v-1]}\right)g(1), \left(1+\zeta^{[v-1]}\right)g(2), \dots, \left(1+\zeta^{[v-1]}\right)g(n-v+1\right)$, $2 \le v \le K+1$. Notice that for both problems, list $H(v)$ has at most $n-v+1$ elements sorted in a non-decreasing order, $1 \le v \le K+1$. It suffices to keep only $n-v+1$ elements in list $H(v)$, since the condition *K-domi* guarantees that each of the $v-1$ earlier groups will have at least one job scheduled in them.

For $k=1$, list $G(1)$ is defined essentially as a copy of list $H(1)$. For each $k$, $2 \le k \le K+1$, list $G(k)$ is obtained be merging the lists $G(k-1)$ and $H(k)$ and taking the $n$ smallest elements of this merger, keeping them in non-decreasing order.

Define $P(S^*(k))$ as the sum of actual durations of the jobs in an optimal schedule with $k$ groups. Let $\gamma_i(k)$ denote the $i$th element in the sorted list $G(k)$, so that $G(k) = (\gamma_1(k), \gamma_2(k), \dots, \gamma_n(k))$. This implies that

$$P(S^*(k)) = \sum_{j=1}^{n} p_j \gamma_j(k),$$

so that

$$C_{\max}\left(S^*(k)\right) = P(S^*(k)) + \Gamma(k) = \sum_{j=1}^{n} p_j \gamma_j(k) + \sum_{x=1}^{k-1} \eta^{[x]}. \tag{16.25}$$

where $\Gamma(k)$ is a constant term as defined in (16.9).

The following algorithm solves an instance of problem $1\left|p_j^{[x]}(r) = p_j g^{[x]}(r), RMP(K), \Delta^{[x]}(\tau)\right|C_{\max}$ and returns the optimal number of RMPs, $k^*-1$, to be included in the schedule (RMP Decision 1) along with the optimal schedule $S^*(k^*)$ with $k^*$ groups.

**Algorithm NSmallPosi2**

INPUT: An instance of either Problem Posi1 or Problem Posi2 with the jobs renumbered in the LPT order
OUTPUT: An optimal schedule $S^*(k^*)$ defined by the processing sequences $\pi^{[x]}$, $1 \leq x \leq k^*$

**Step 1.** For Problem Posi1, define $H(1) := \left(g^{[1]}(1), g^{[1]}(2), \ldots, g^{[1]}(n)\right)$, while for Problem Posi2, define $H(1) := (g(1), g(2), \ldots, g(n))$. For a given problem, set $G(1) := H(1)$. Compute $C_{\max}(S^*(1))$ by formula (16.25). Define $k' := K + 1$.

**Step 2.** For $k$ from 2 to $k'$, do

(a) Create the list $G(k) = (\gamma_1(k), \gamma_2(k), \ldots, \gamma_n(k))$ that contains the $n$ smallest elements in the merger of the lists $G(k-1)$ and $H(k)$.

(b) Compute $C_{\max}(S^*(k))$ by formula (16.25). If $P(S^*(k)) = P(S^*(k-1))$, then define $k' := k - 1$ and break the loop by moving to Step 3; otherwise, continue the loop with the next value of $k$.

**Step 3.** Find the value $k^*$, $1 \leq k^* \leq k'$, such that

$$C_{\max}\left(S^*(k^*)\right) = \min\left\{C_{\max}\left(S^*(k)\right) | 1 \leq k \leq k'\right\}.$$

**Step 4.** Run Algorithm NSmallPosi for the found value of $k^*$ to obtain the optimal processing sequence $\pi^* = \left(\pi^{[1]}, \pi^{[2]}, \ldots, \pi^{[k^*]}\right)$.

Our method is justified, because list $G(k-1)$ already contains the $n$ smallest positional weights coming from the first $k-1$ groups. Thus, to search for the $n$ smallest weights needed for schedule $S(k)$, there is no need to scan the first $k-1$ groups again. In other words, we utilize the fact that if a certain position in the first $k-1$ groups is not used in schedule $S^*(k-1)$, it will not be used in schedule $S^*(k)$ either.

In Step 2, for every $k$, both lists $G(k-1)$ and $H(k)$ have at most $n$ elements sorted in a non-decreasing order; therefore, each Step 2(a) and Step 2(b) can be completed in $O(n)$ time. If the loop in Step 2 is not broken throughout the run of Algorithm NSmallPosi2, the final value of $k'$ remains equal to $K + 1$, i.e., it is possible that all RMPs will be run and all groups will be opened in an optimal schedule. The loop in Step 2 may be stopped when in Step 2b the condition $P(S^*(k)) = P(S^*(k-1))$ is achieved. This condition implies that the opening of the $k$th group does not provide any positional weights smaller than those contained in the list $G(k-1)$. If this happens for the $k$th group, all groups that could be opened after this would provide even worse positional weights, because list $H(k+1)$ is dominated by list $H(k)$, $1 \leq k \leq K$. Thus, the makespan cannot be reduced by running more RMPs after the $k'$th group is opened, so that there is no need to examine further values of $k$. With the found value of $k'$, the overall optimal schedule will be found among the schedules $S^*(k)$, $1 \leq k \leq k'$.

**Table 16.5** Run of Algorithm NSmall2 for Example 16.1

| $j$ | $p_j$ | $G(1)$ | $p_j\gamma_j$ | $H(2)$ | $G(2)$ | $p_j\gamma_j$ | $H(3)$ | $G(3)$ | $p_j\gamma_j$ | $H(4)$ | $G(4)$ | $p_j\gamma_j$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 10 | 1 | 10 | 2 | 1 | 10 | 2 | 1 | 10 | 3 | 1 | 10 |
| 2 | 9 | 2 | 18 | 4 | 2 | 18 | 4 | 2 | 18 | 6 | 2 | 18 |
| 3 | 6 | 2 | 12 | 4 | 2 | 12 | 4 | 2 | 12 | 6 | 2 | 12 |
| 4 | 3 | 3 | 9 | 6 | 2 | 6 | 6 | 2 | 6 |  | 2 | 6 |
| 5 | 3 | 3 | 9 | 6 | 3 | 9 |  | 2 | 6 |  | 2 | 6 |
| 6 | 2 | 4 | 8 |  | 3 | 6 |  | 3 | 6 |  | 3 | 6 |
| $P(S^*(1))$ | | | 66 | $P(S^*(2))$ | | 61 | $P(S^*(3))$ | | 58 | $P(S^*(4))$ | | 58 |
| $\Gamma(1)$ | | | 0 | $\Gamma(2)$ | | 1 | $\Gamma(3)$ | | 3 | $\Gamma(4)$ | | 6 |
| $C_{\max}(S^*(1))$ | | | 66 | $C_{\max}(S^*(2))$ | | 62 | $C_{\max}(S^*(3))$ | | 61 | $C_{\max}(S^*(4))$ | | 64 |

**Theorem 16.3** *Algorithm NSmallPosi2 solves an instance either of Problem Posi1 or of Problem Posi2 in $O(nK)$ time, provided that the LPT order of the jobs is known.*

*Example 16.1* We illustrate Algorithm NSmallPosi2 by considering an instance of version Problem Posi2, i.e., a version of problem $1\left|p_j^{[x]}(r) = p_j g^{[x]}(r), RMP(K), \Delta^{[x]}(\tau)\right|C_{\max}$. Six jobs are to be scheduled which have the following normal processing times listed in an LPT order

$$p_1 = 10, p_2 = 9, p_3 = 6, p_4 = 3, p_5 = 3, p_6 = 2.$$

The decision-maker has a choice of $K = 5$ RMPs, with the following parameters

$$\begin{aligned}
\text{RMP}^{[1]} &: \zeta^{[1]} = 1,\ \eta^{[1]} = 1;\\
\text{RMP}^{[2]} &: \zeta^{[2]} = 1,\ \eta^{[2]} = 2;\\
\text{RMP}^{[3]} &: \zeta^{[3]} = 2,\ \eta^{[3]} = 3;\\
\text{RMP}^{[4]} &: \zeta^{[4]} = 2,\ \eta^{[4]} = 4;\\
\text{RMP}^{[5]} &: \zeta^{[5]} = 3,\ \eta^{[5]} = 4,
\end{aligned}$$

so that they obey (16.14) and (16.22) simultaneously. Each of the RMPs restores the machine to an “as good as new” state. The group-independent positional factors are as follows:

$$g(1) = 1,\ g(2) = 2,\ g(3) = 2,\ g(4) = 3,\ g(5) = 3,\ g(6) = 4.$$

Table 16.5 shows the details of the run of Algorithm NSmallPosi2 for the above instance. Since $\gamma_r(3) = \gamma_r(4)$, for each $r$, $1 \le r \le 6$, the algorithm stops after the iteration $k = 4$, so that $k' = 3$. The algorithm outputs the minimum value of the makespan from the set $\{C_{\max}(S^*(k))|1 \le k \le 3\}$, which is $C_{\max}(S^*(3))$. In an optimal schedule for $k^* = 3$, the sequence of jobs $\pi^{[1]} = (1, 2, 3, 6)$ is processed in the first group, the sequence of jobs $\pi^{[2]} = (4)$ is processed in the second group, and the sequence of jobs $\pi^{[3]} = (5)$ is processed in the third group. The makespan of the resulting schedule is 61.

Algorithm NSmallPosi2 can also be applied to solve problem $1\left|p_j^{[x]}(r) = p_j g^{[x]}(r), RMP(K), \Delta^{[x]}(\tau)\right|C_{\max}$ with group-independent deterioration factors and constant duration RMPs (both identical and distinct). This is possible since both conditions (16.14) and the *K-domi* condition can be satisfied simultaneously. The required running time is again $O(nK)$. We do not discuss the solution of this problem here, as it is possible to solve it faster using another solution approach, presented in Sect. 16.2.4.

Now, consider a version of problem $1\left|p_j^{[x]}(r) = p_j g^{[x]}(r), RMP(K), \Delta^{[x]}(\tau)\right| C_{\max}$ in which the conditions *K-domi* and (16.14) do not hold simultaneously. Algorithm NSmallPosi2 can still be used to obtain an optimal value for RMP Decision 1 and to find an optimal permutation of jobs, but to make RMP Decisions 2 and 3, full enumeration of options might be required. As a result, the overall running time to solve problem $1\left|p_j^{[x]}(r) = p_j g^{[x]}(r), RMP(K), \Delta^{[x]}(\tau)\right|C_{\max}$ turns out to be no smaller than that obtained by using the general solution approach presented in Sect. 16.2.2.

### 16.2.4 Binary Search in Convex Sequences

In this section, we deal with problems in which the computed positional weights are group-independent, i.e., of the form $W^{[x]}(r) = W(r)$, $1 \le x \le k$, and additionally, they are ordered in a way such that $W(1) \le W(2) \le \cdots \le W(n)$. Such a situation arises for versions of problem $1\left|p_j^{[x]}(r) = p_j g^{[x]}(r), RMP(K), \Delta^{[x]}(\tau)\right|C_{\max}$, in which the deterioration factors are group-independent, i.e., $g^{[x]}(r) = g(r)$, $1 \le r \le n$, $1 \le x \le K+1$, while the RMPs are of constant duration, i.e., $\zeta^{[y]} = 0$, $1 \le y \le K$. Below, we assume that the RMPs are distinct; the problem with identical RMPs is its special case, and no faster solution algorithm than that described below is known.

Formally, we denote the described problem by $1\left|p_j^{[x]}(r) = p_j g(r), RMP(K), \Delta^{[x]}\right|C_{\max}$. In the middle field, the first term is used to notify that the deterioration factors are group-independent, the second term $RMP(K)$ is used to notify that a total of $K$ RMPs are available to be included in the schedule, and the third term is used to notify that the RMPs are of constant duration, but their values are group-dependent.

As in Sect. 16.2.3, we assume that the RMPs are numbered in such a way that (16.14) holds. Thus, if $k-1$, $1 \le k \le K+1$, RMPs are to be included in the schedule, then $\mathrm{RMP}^{[1]}, \ldots, \mathrm{RMP}^{[k-1]}$ are to be chosen and inserted into a schedule in the order of their numbering. The resulting problem $1\left|p_j^{[x]}(r) = p_j g(r), RMP(k-1), \Delta^{[x]}\right|C_{\max}$ can be solved by minimizing the generic objective function (16.17). Obtain the required positional weights $W^{[x]}(r)$ by substituting $g^{[x]}(r) = g(r)$, $1 \le r \le n$, and $\zeta^{[x]} = 0$, $1 \le x \le k$, in (16.18), so that we have

$$W^{[x]}(r) = g(r),\ 1 \le r \le n^{[x]},\ 1 \le x \le k.$$

Below, we outline a approach to solving problem $1\left|p_j^{[x]}(r) = p_j g(r),\right.$ $RMP(k-1),\ \Delta^{[x]}\left|C_{\max}\right.$, which is again based on Theorem 16.2.

Notice that for a given position $r$, $1 \le r \le n$, the positional weights are the same for every group, and within each group, they are sorted in a non-decreasing order. This implies that unlike for the problems studied in Sect. 16.2.3, the $n$ smallest positional weights for this problem are known. The $k$ smallest positional weights are due to the first positions of each of the $k$ groups. The next $k$ smallest positional weights are due to the second positions of each of the $k$ groups and so on. Assuming that $n = \lambda k + \mu$, where $\lambda$ and $\mu$ are non-negative integers, $\mu \le k-1$, the optimal number of jobs in each group can be given by

$$n^{[x]} = \begin{cases} \left\lceil \frac{n}{k} \right\rceil = \lambda + 1, & 1 \le x \le \mu \\ \left\lfloor \frac{n}{k} \right\rfloor = \lambda, & \mu + 1 \le x \le k. \end{cases} \tag{16.26}$$

With known values of $W^{[x]}(r)$, $1 \le r \le n^{[x]}$, and $n^{[x]}$, $1 \le x \le k$, an optimal makespan $C_{\max}(S^*(k))$ for problem $1\left|p_j^{[x]}(r) = p_j g(r), RMP(k-1), \Delta^{[x]}\right|C_{\max}$ can be found by performing an appropriate matching. If the jobs are considered in the LPT order, the first $k$ jobs will be assigned to the first positions in each of the $k$ groups, the next $k$ jobs will be assigned to the second positions in each of the $k$ groups, and so on, until all jobs have been scheduled. The following statement holds.

**Theorem 16.4** *Problem* $1\left|p_j^{[x]}(r) = p_j g(r), RMP(k-1), \Delta^{[x]}\right|C_{\max}$ *in* $O(n)$ *time, provided that the LPT order of the jobs is known.*

To determine the optimal solution for problem $1\left|p_j^{[x]}(r) = p_j g(r), RMP(K),\right.$ $\Delta^{[x]}\left|C_{\max}\right.$, all options associated with RMP Decisions 1–3 must be enumerated. RMP Decisions 2 and 3 have already been taken optimally; thus, we only need to determine the optimal value of the number of RMPs to be included into a schedule. A straightforward approach would involve solving all auxiliary problems $1\left|p_j^{[x]}(r) = p_j g(r), RMP(k-1), \Delta^{[x]}\right|C_{\max}$, $1 \le k \le K+1$, and choosing the schedule $S^*(k)$ with the smallest makespan as an overall schedule $S^*$. This brute-force algorithm solves problem $1\left|p_j^{[x]}(r) = p_j g(r), RMP(K), \Delta^{[x]}\right|C_{\max}$ in $O(nK)$ time.

This running time can be improved, since we prove that the sequence $C_{\max}(S^*(k))$, $1 \le k \le K+1$, is in fact $V$-shaped, so that in order to search for the smallest value of $C_{\max}(S^*(k))$, we only need to evaluate $\lceil \log_2(K+1) \rceil$ options of $k$, $1 \le k \le K+1$.

Recall from Sect. 5.1 that a sequence $A(k)$ is called *V-shaped* if there exists a $k_0$, $1 \le k_0 \le K+1$, such that

$$A(1) \ge \cdots \ge A(k_0-1) \ge A(k_0) \le A(k_0+1) \le \cdots \le A(K+1).$$

For a schedule $S(k))$, let $P(S(k))$ denote the sum of the actual durations of the jobs, and $\Gamma(k)$ be the total duration of all $k-1$ RMPs defined by (16.9), so that $C_{\max}(S(k)) = P(S(k)) + \Gamma(k)$ holds. The following statement holds.

**Lemma 16.3** *For problem* $1\left|p_j^{[x]}(r) = p_j g(r), RMP(k-1), \Delta^{[x]}\right|C_{\max}$, *if the jobs be numbered in the LPT order, then the makespan of the optimal schedule can be written as*

$$C_{\max}(S^*(k)) = P(S^*(k)) + \Gamma(k) = \sum_{j=1}^{n} p_j g\left(\left\lceil \frac{j}{k} \right\rceil\right) + \sum_{x=1}^{k-1} \eta^{[x]},\ 1 \le k \le K+1. \tag{16.27}$$

*Proof* The value $C_{\max}(S(k))$ can be seen as $P(S(k)) + \Gamma(k)$, where $P(S(k))$ denotes the sum of the actual durations of the jobs in a schedule $S(k)$ and $\Gamma(k)$ is the total duration of all $k-1$ RMPs defined by (16.9). Recall that if the jobs are numbered in the LPT order, then to minimize the value $P(S(k))$, we need to assign the first $k$ jobs to the first positions in each of the $k$ groups, then the next $k$ jobs going to the second positions in each of the $k$ groups, and so on, until all jobs have been sequenced.

Formally, if $j = \lambda k$, then the predecessors of $j$ are placed into the first $\lambda$ positions of groups $1, 2, \ldots, k-1$ and take $\lambda - 1$ positions of group $k$, so that job $j$ is assigned to position $\lambda = \left\lceil \frac{j}{k} \right\rceil$ of group $k$. If $j = \lambda k + \mu$ for $1 \le \mu \le k-1$, then the predecessors of $j$ will take the first $\lambda$ positions in each group and additionally the $(\lambda+1)$th position in each of the groups $1, 2, \ldots, \mu - 1$, so that job $j$ gets position $\lambda + 1 = \left\lceil \frac{j}{k} \right\rceil$ in group $\mu$.

It follows that the actual processing time of a job $j \in N$ in an optimal schedule $S^*(k)$ is equal to $p_j g\left(\left\lceil \frac{j}{k} \right\rceil\right)$, and the total processing time for all jobs is equal to

$$P(S^*(k)) = \sum_{j=1}^{n} p_j g\left(\left\lceil \frac{j}{k} \right\rceil\right),\ 1 \le k \le K+1, \tag{16.28}$$

as required. □

**Theorem 16.5** *For problem* $1\left|p_j^{[x]}(r) = p_j g(r), RMP(K), \Delta^{[x]}\right|C_{\max}$, *the sequence* $C_{\max}(S^*(k)) = P(S^*(k)) + \Gamma(k)$, $1 \le k \le K+1$, *given by (16.27), is V-shaped.*

*Proof* The proof is based on the concept of a convex sequence, a discrete analog of a convex function. Recall from Chap. 5 that a sequence $\Gamma(k)$, $1 \le k \le K+1$, is called convex if

$$\Gamma(k) \le \frac{1}{2}(\Gamma(k-1) + \Gamma(k+1)),\ 2 \le k \le K.$$

Also recall Theorem 5.2, in which it is stated that a sequence

$$P(k) = \sum_{j=1}^{n} p_j g\left(\left\lceil \frac{j}{k} \right\rceil\right),\ 1 \le k \le n,$$

is convex, provided that the jobs are ordered in the LPT order and the sequence $g(r)$, $1 \le r \le n$, is non-decreasing. It immediately follows that the sequence $P(S^*(k))$, $1 \le k \le K+1$, given by (16.28) is convex.

The sequence $\Gamma(k) = \sum_{x=1}^{k-1} \eta^{[x]}$, $1 \le k \le K+1$, can also be proved to be convex. Indeed, (16.14) corresponds to the inequalities $\eta^{[x-1]} \le \eta^{[x]}$, $2 \le x \le K$, which can be rewritten as $\eta^{[x-1]} \le \frac{1}{2}\eta^{[x-1]} + \frac{1}{2}\eta^{[x]}$, $2 \le x \le K$. Taking a $k$, $2 \le k \le K$, and summing up, we deduce

$$\begin{aligned}
\sum_{x=2}^{k} \eta^{[x-1]} &\le \frac{1}{2}\left(\sum_{x=2}^{k} \eta^{[x-1]} + \sum_{x=2}^{k} \eta^{[x]}\right); \\
\sum_{x=1}^{k-1} \eta^{[x]} &\le \frac{1}{2}\left(\sum_{x=1}^{k-2} \eta^{[x]} + \sum_{x=1}^{k} \eta^{[x]}\right) + \frac{1}{2}\eta^{[k-1]} - \frac{1}{2}\eta^{[1]} \\
&\le \frac{1}{2}\left(\sum_{x=1}^{k-2} \eta^{[x]} + \sum_{x=1}^{k} \eta^{[x]}\right).
\end{aligned}$$

The last inequality is true since (16.14) holds, so that

$$\Gamma(k) \le \frac{1}{2}(\Gamma(k-1) + \Gamma(k+1)),\ 2 \le k \le K,$$

and the sequence $\Gamma(k)$, $1 \le k \le K+1$, is convex. Note that the latter inequalities become equalities if the RMPs have identical durations.

Since the sum of two convex sequences is convex, the sequence $C_{\max}(S^*(k)) = P(S^*(k)) + \Gamma(k)$, $1 \le k \le K+1$, is also convex and by Lemma 5.1 is $V$-shaped. □

Theorem 16.5 allows us to find the optimal number of groups $k^*$, $1 \le k^* \le K+1$, to be created by the following binary search algorithm.

**Algorithm BinSearchPosi**

INPUT: An instance of problem $1\left|p_j^{[x]}(r) = p_j g(r),\, RMP(K),\, \Delta^{[x]}\right|C_{\max}$

OUTPUT: The optimal number of RMPs to include in the schedule

**Step 0**. If required, renumber the jobs in an LPT order and renumber the RMPs in a non-decreasing order of their durations so that $\eta^{[1]} \le \eta^{[2]} \le \cdots \le \eta^{[K]}$.

**Step 1**. Define $\underline{k} := 1, \overline{k} := K+1$ and $\widetilde{k} := \lceil (K+1)/2 \rceil$. Compute $C_{\max}(S^*(k))$ by formula (16.27) for $k \in \left\{\underline{k}, \widetilde{k}, \overline{k}\right\}$.

**Step 2**. If $C_{\max}\left(S^*(\underline{k})\right) \le C_{\max}\left(S^*(\widetilde{k})\right)$, then go to Step 3; otherwise, go to Step 4

**Step 3**. Redefine $\overline{k} := \widetilde{k}$, $C_{\max}\left(S^*(\overline{k})\right) := C_{\max}\left(S^*(\widetilde{k})\right)$ and go to Step 5.

**Step 4**. Redefine $\underline{k} := \widetilde{k}$, $C_{\max}\left(S^*(\underline{k})\right) := C_{\max}\left(S^*(\widetilde{k})\right)$ and go to Step 5.

**Step 5**. Redefine $\widetilde{k} := \left\lceil \left(\underline{k} + \overline{k}\right)/2 \right\rceil$. If $\underline{k} = \widetilde{k} = \overline{k}$, then output $k^* = \underline{k}$, and stop; otherwise, compute $C_{\max}\left(S^*(\widetilde{k})\right)$ and go to Step 2.

It is clear that due to the $V$-shapeness of the sequence $C_{\max}(S^*(k)), 1 \le k \le n$, the inequality $C_{\max}\left(S^*(\underline{k})\right) \le C_{\max}\left(S^*(\widetilde{k})\right)$ implies that the subsequence $C_{\max}(S^*(k))$, $\widetilde{k} \le k \le n$, is monotone non-decreasing, so that the minimum should be sought between the values $\underline{k}$ and $\widetilde{k}$. Similarly, the inequality $C_{\max}\left(S^*(\underline{k})\right) > C_{\max}\left(S^*(\widetilde{k})\right)$ implies that the minimum should be sought between the values $\widetilde{k}$ and $\overline{k}$. All together, Algorithm BinSearchPosi explores at most $\lceil \log_2(K+1) \rceil$ values of $k$ and the following statement holds.

**Theorem 16.6** *Algorithm BinSearchPosi solves an instance of problem* $1\left|p_j^{[x]}(r) = p_j g(r), RMP(K),\, \Delta^{[x]}\right|C_{\max}$ *in* $O(n \log K)$ *time, provided that the LPT order of the jobs is known.*

Table 16.6 summarizes the corresponding running times required to solve different versions of problem $1\left|p_j^{[x]}(r) = p_j g^{[x]}(r), RMP(K),\, \Delta^{[x]}(\tau)\right|C_{\max}$, provided the jobs are available in the LPT order.

**Table 16.6** Running time required for different versions of problem $1\left|p_j^{[x]}(r) = p_j g^{[x]}(r),\right.$ $\left.RMP(K),\, \Delta^{[x]}(\tau)\right|C_{\max}$

| | Constant duration MPs | | Start-time-dependent MPs | |
|---|---|---|---|---|
| | Identical | Distinct | Identical | Distinct |
| Group-indep | $O(n \log K)$ | $O(n \log K)$ | $O(nK)$ | $O\left(n2^K \log K\right)$ |
| Group-dep | $O(nK)$ | $O\left(nK^K \log K\right)$ | $O(nK \log K)$ | $O\left(nK^K \log K\right)$ |

## 16.3 Bibliographic Notes

One of the first papers that study the problem of changing processing times with rate-modifying activities is due to Kuo and Yang (2008). They study problem $1\left|p_j^{[x]}(r) = p_j r^a, a > 0, RMP(n), \Delta\right|C_{\max}$, with a group-independent polynomial deterioration effect, i.e., $g^{[x]}(r) = r^a$, $a > 0$, and identical RMPs, whose duration is assumed to be constant. To solve the resulting auxiliary problem $1\left|p_j^{[x]}(r) = p_j r^a, a > 0, RMP(k-1), \Delta\right|C_{\max}$ with $k-1$ RMPs, they prove the so-called *group balance principle*, according to which in an optimal schedule with $k$ groups the difference between the numbers of jobs in any two groups is at most one. With a known number of jobs in each group, they apply a method similar to Algorithm NSmall-Posi to solve problem $1\left|p_j^{[x]}(r) = p_j r^a, a > 0, RMP(k-1), \Delta\right|C_{\max}$. Trying all possible values of $k$, $1 \le k \le n$, an optimal solution to problem $1\left|p_j^{[x]}(r) = p_j r^a, a > 0, RMP(n), \Delta\right|C_{\max}$ is obtained in $O(n^2)$ time. It should be noted that Kuo and Yang (2008) claim that their algorithm requires $O(n \log n)$ time. In fact, they do not take into account the linear time that is needed to compute the value of the makespan for each $k$, $1 \le k \le n$. In Sect. 16.2.4, we prove that the running time for solving problem $1\left|p_j^{[x]}(r) = p_j r^a, a > 0, RMP(n), \Delta\right|C_{\max}$ can indeed be reduced to $O(n \log n)$, since Algorithm BinSearchPosi is applicable.

Zhao and Tang (2010) study a problem similar to that in Kuo and Yang (2008), but with job-dependent polynomial effects. Due to the group balance principle, the auxiliary problem $1\left|p_j^{[x]}(r) = p_j r^{a_j}, a_j > 0, RMP(k-1), \Delta\right|C_{\max}$ is reduced to an LAP, which is solvable in $O(n^3)$ time. Trying all possible values of $k$, $1 \le k \le n$, a solution to problem $1\left|p_j^{[x]}(r) = p_j r^{a_j}, a_j > 0, RMP(n), \Delta\right|C_{\max}$ is found in $O(n^4)$ time. Yang and Yang (2010) study a problem similar to that in Zhao and Tang (2010), but the durations of the RMPs are given as a linear function of their start time, so that the resulting problem is denoted as $1\left|p_j^{[x]}(r) = p_j r^{a_j}, a_j > 0, RMP(n), \Delta(\tau)\right|C_{\max}$. Yang and Yang (2010) solve the auxiliary problem $1\left|p_j^{[x]}(r) = p_j r^{a_j}, a_j > 0, RMP(k-1), \Delta(\tau)\right|C_{\max}$ by proving the group balance principle for the first $k-1$ groups, which allows them to guess the number of jobs in those groups. However, for the last group, they are not able to guess the number of jobs, so they resort to enumerating all possible options for the number of jobs in that group. As a result, the running time needed to solve this problem is $n$ times greater than that required by Zhao and Tang (2010), thereby making problem $1\left|p_j^{[x]}(r) = p_j r^{a_j}, a_j > 0, RMP(n), \Delta(\tau)\right|C_{\max}$ solvable in $O(n^5)$ time.

Rustogi and Strusevich (2012) further generalize the problems considered by Kuo and Yang (2008), Zhao and Tang (2010), and Yang and Yang (2010), by introducing general group-dependent positional effects, so that the actual processing time of a job is dependent on both the position of a job in a group and the position of the group in a schedule. This is the first paper in which the RMPs are allowed to be distinct; however, the choice and order of the RMPs are fixed in advance. For the most general problem $1\left|p_j^{[x]}(r) = p_j g_j^{[x]}(r), RMP(n), \Delta^{[x]}(\tau)\right|C_{\max}$ with job-dependent deterioration effects, the auxiliary problem $1\left|p_j^{[x]}(r) = p_j g_j^{[x]}(r), RMP(k-1),\right.$ $\left.\Delta^{[x]}(\tau)\right|C_{\max}$ is solved by reduction to a rectangular linear assignment problem. The resulting problem is solved by Algorithm LAPBL (see Chap. 4 for details) which requires $O(n^3 k)$ time. Trying all possible values of $k$, $1 \le k \le n$, a solution to problem $1\left|p_j^{[x]}(r) = p_j g_j^{[x]}(r), RMP(n), \Delta^{[x]}(\tau)\right|C_{\max}$ is found in $O(n^5)$ time. Rustogi and Strusevich (2012) also provide a faster solution approach that requires $O(n^4)$ time for two special cases of problem $1\left|p_j^{[x]}(r) = p_j g_j^{[x]}(r), RMP(n), \Delta^{[x]}(\tau)\right|C_{\max}$, i.e., when (i) duration of RMPs is constant and known in advance and (ii) duration of RMPs is start-time-dependent, but positional factors are group-independent. For these special cases, it is proved that the rectangular assignment problem can be solved in $O(n^3)$ time due to a special structure observed in the cost matrix, thereby making the overall problem solvable in $O(n^4)$ time. In the same paper, problems with job-independent deterioration effects are also considered and a solution approach is presented that is similar to Algorithm NSmallPosi2.

Finke et al. (2016) generalize the problem considered by Rustogi and Strusevich (2012) by introducing RMPs with the durations of the form (16.6). Similar to Rustogi and Strusevich (2012), they prove that for their problem denoted by $1\left|p_j^{[x]}(r) = p_j g_j^{[x]}(r), RMP(n), \bar{\Delta}^{[x]}(\tau)\right|C_{\max}$, the resulting auxiliary problem denoted by $1\left|p_j^{[x]}(r) = p_j g_j^{[x]}(r), RMP(k-1), \bar{\Delta}^{[x]}(\tau)\right|C_{\max}$ can be solved by reduction to a rectangular linear assignment problem. However, instead of Algorithm LAPBL, as used by Rustogi and Strusevich (2012), they use a more efficient Algorithm LAPD (see Chap. 4 for details) which requires $O(n^3 + n^2 k)$ time. Trying all possible values of $k$, $1 \le k \le n$, a solution to the general problem $1\left|p_j^{[x]}(r) = p_j g_j^{[x]}(r),\right.$ $\left.RMP(n), \bar{\Delta}^{[x]}(\tau)\right|C_{\max}$ is found in $O(n^4)$ time.

## References

Finke G, Gara-Ali A, Espinouse ML, Jost V, Moncel J (2016) Unified matrix approach to solve production-maintenance problems on a single machine. Omega. doi:10.1016/j.omega.2016.02.005

Kuo W-H, Yang D-L (2008) Minimising the makespan in a single-machine scheduling problem with the cyclic process of an aging effect. J Oper Res Soc 59:416–420

Rustogi K, Strusevich VA (2012) Single machine scheduling with general positional deterioration and rate-modifying maintenance. Omega 40:791–804

Yang S-J, Yang D-L (2010) Minimising the makespan single-machine scheduling with aging effects and variable maintenance activities. Omega 38:528–533

Zhao C-L, Tang H-Y (2010) Single machine scheduling with general job-dependent aging effect and maintenance activities to minimise makespan. Appl Math Model 34:837–841

# Chapter 17
# Scheduling with Maintenance and Start-Time-Dependent Effects

In this chapter, we discuss single machine scheduling problems with time-dependent deterioration effects and rate-modifying maintenance activities. The structure of this chapter is similar to that of Chap. 16. We provide a full account of the entire range of single machine problems to minimize the makespan that can be solved using the developed solution approaches.

In the problems considered in this chapter, the jobs of set $N = \{1, 2, \ldots, n\}$ are to be processed on a single machine. Each job $j \in N$ is associated with a normal processing time $p_j$. As described in Chap. 8, in the case of time-dependent effects, one of the most studied models is given by a linear function of the start time of a job, so that the actual processing time $p_j(\tau)$ of a job $j \in N$ that starts at a time $\tau \geq 0$ is given by

$$p_j(\tau) = p_j + a_j\tau, \tag{17.1}$$

where $a_j$ is a constant, which is strictly positive in the case of deterioration and strictly negative in the case of learning. In this chapter, we mainly concentrate on effects that are derived from a job-independent version of (17.1) and are given by

$$p_j(\tau) = p_j + a\tau, \tag{17.2}$$

where in the case of deterioration the rate $a > 0$ is common for all jobs.

In this chapter, we consider enhanced single machine scheduling models, in which the processing times of the jobs are subject to time-dependent deterioration effects of the form (17.2) and various rate-modifying periods (RMP) can be introduced on the machine to restore the processing conditions, either fully or at least partly.

Consider a general situation, outlined in Sect. 12.4, in which the decision-maker is presented with a list ($\text{RMP}^{[1]}$, $\text{RMP}^{[2]}, \ldots,$ $\text{RMP}^{[K]}$) of $K \geq 1$ possible rate-modifying activities. The decision-maker may decide which of the listed RMPs to insert into a schedule and in which order. Each RMP may restore the machine to a different state, so that different deterioration rates are applied in different groups. For each $\text{RMP}^{[y]}$ in the list, we are given the deterioration rate $a^{[y+1]} > 0$, $1 \leq y \leq K$,

V.A. Strusevich and K. Rustogi, *Scheduling with Times-Changing Effects and Rate-Modifying Activities*, International Series in Operations Research & Management Science 243, DOI 10.1007/978-3-319-39574-6_17

that applies to the jobs sequenced in the group that follows $\mathrm{RMP}^{[y]}$, provided it is included into a schedule. As part of the input, we are also given the deterioration rate $a^{[1]} > 0$ that applies to the first group that starts at time zero, before the first scheduled RMP.

If $k - 1$ RMPs are chosen from the available $K$ options, then the jobs are divided into $k$, $1 \le k \le K + 1$, groups. Depending on which RMPs are chosen and the order in which they are performed, the actual processing time $p_j^{[x]}(\tau)$ of a job $j \in N$ that is located in the $x$th group and starts at time $\tau$ measured from the beginning of the group is given by

$$p_j^{[x]}(\tau) = p_j + a^{[x]}\tau,\ \ \tau \ge 0,\ \ 1 \le x \le k, \tag{17.3}$$

where $a^{[x]} > 0$ is the deterioration rate. For a particular schedule, the inserted RMPs are renumbered in the order of their appearance, so that a group sequenced after the $x$th RMP is associated with a deterioration rate $a^{[x+1]} > 0$, $1 \le x \le k - 1$, and it is assumed that the time $\tau$ is reset after every RMP. Notice that these group-dependent deterioration rates $a^{[x]} > 0$ are analogous to the group-dependent positional factors studied in Chap. 16 and imply that the actual processing time of a job is dependent on the starting time of a job in a group and also on the group that job is scheduled in.

Recall that during each RMP no job processing takes place. The duration of an $\mathrm{RMP}^{[y]}$, $1 \le y \le K$, is given by

$$\Delta^{[y]}(\tau) = \zeta^{[y]}\tau + \eta^{[y]}, \tag{17.4}$$

as introduced in (12.1), where $\tau$ is the start time of the RMP, measured either from time zero (in the case of the first RMP) or from the completion time of the previous RMP.

In this chapter, we mainly focus on the problems of minimizing the makespan under the general settings defined by (17.3) and (17.4). The most general problem of this range is denoted by $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, RMP(K), \Delta^{[x]}(\tau)\right|C_{\max}$, where the first term in the middle field indicates that the actual processing times are start time dependent as given by (17.3), the second term points out that the list of $K$ RMPs is available for maintenance activities, and the third term indicates that the durations of the RMPs follow the rule (17.4).

Similarly to problem $1\left|p_j^{[x]}(r) = p_j g^{[x]}(r), RMP(K), \Delta^{[x]}(\tau)\right|C_{\max}$ studied in Sect. 16.2, an optimal solution to problem $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, RMP(K),\right.$ $\left.\Delta^{[x]}(\tau)\right|C_{\max}$ can be found by adapting Procedure RMP1 from Sect. 12.4. Recall that in particular Procedure RMP1 requires taking the RMP Decisions 1–3.

The example below illustrates a situation where the problem defined above may be applicable.

*Example 17.1* Six jobs should be processed by a human operator who has equipment with multiple cutting tools. The processing times (in appropriate time units) of these jobs under perfect conditions of both the operator and the tools are given by

$$p_1 = 10, p_2 = 9, p_3 = 6, p_4 = 3, p_5 = 3, p_6 = 2.$$

During the processing, the operator gets tired and the tool loses its sharpness, which leads to extending the actual processing time, i.e., to a deterioration effect. During the planning period, up to $K = 5$ types of maintenance activities can be performed. Some can be seen as rest periods for the operator (leaving the tools' conditions unchanged), and some of them are related to improving the cutting capabilities of one or several tools (with the operator having rest as well). In any case, the duration of $\mathrm{RMP}^{[y]}$, $1 \le y \le 5$, is defined in accordance with (17.4) and may include a constant term $\eta^{[y]}$, which can be seen as the duration of mandatory standard tests to be run for the maintenance activity of this type, as well as the start-time-dependent term $\zeta^{[y]}\tau$, which is understood as the duration of repair activities that depends on the conditions of the tool(s). The rates $\zeta^{[y]}$ are different for different types of maintenance activities, since they involve work on different tools. Any RMP improves the processing conditions, but not to an initial (perfect) state, so that a different deterioration rate might be in effect after each RMP. As a result, the jobs are split into several groups, one before the first RMP and one after each scheduled RMP. The actual processing times of the jobs follow the rule (17.3). For a group $x$, the job deterioration rate $a^{[x]}$ is set by the preceding $(x-1)$th RMP. The default deterioration rate is given as $a^{[1]} = 0.1$. The other numerical parameters are given by

$$\begin{array}{llll}
\mathrm{RMP}^{[1]}: & \zeta^{[1]} = 0.05, & \eta^{[1]} = 10, & a^{[2]} = 0.15; \\
\mathrm{RMP}^{[2]}: & \zeta^{[2]} = 0.10, & \eta^{[2]} = 8, & a^{[3]} = 0.20; \\
\mathrm{RMP}^{[3]}: & \zeta^{[3]} = 0.025, & \eta^{[3]} = 6, & a^{[4]} = 0.25; \\
\mathrm{RMP}^{[4]}: & \zeta^{[4]} = 0.15, & \eta^{[4]} = 2, & a^{[5]} = 0.20; \\
\mathrm{RMP}^{[5]}: & \zeta^{[5]} = 0.2, & \eta^{[5]} = 0, & a^{[6]} = 0.15.
\end{array}$$

In the setting of this example, RMP Decision 1 is to choose $k$ types of maintenance activities to be included into the schedule, e.g., 3 out of 5. Then RMP Decision 2 selects particular types of maintenance, e.g., $\mathrm{RMP}^{[1]}$, $\mathrm{RMP}^{[3]}$, and $\mathrm{RMP}^{[4]}$. RMP Decision 3 determines the sequence in which the three chosen RMPs are included into the schedule, e.g., $\mathrm{RMP}^{[3]}$, $\mathrm{RMP}^{[4]}$, and $\mathrm{RMP}^{[1]}$. In the resulting schedule, there will be 4 groups, with the job deterioration rates equal to 0.1, 0.25, 0.20 and 0.15, respectively.

In what follows, we show that problem $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, RMP(K), \Delta^{[x]}(\tau)\right|C_{\max}$ reduces to a series of linear assignment problems with a product matrix and can be solved in polynomial time. As a rule, to solve scheduling problems under consideration, it is required to generate several instances of auxiliary problems with fixed parameters, such as the number of RMPs to be inserted, and/or the number of jobs in a group. To count the number of the related instances, we use various combinatorial configurations and identities listed in Sect. 5.3. In the estimations of the running times of the presented algorithms, we assume that the number $K$ of available RMPs is a constant, which is reasonable since usually the number of possible rate-modifying activities to be performed is fairly small.

As often is the case in this book, the LPT sequencing of the jobs is important. Recall that if the jobs are numbered in accordance with the LPT rule, then

$$p_1 \geq p_2 \geq \cdots \geq p_n. \tag{17.5}$$

## 17.1 Computing Positional Weights

For problem $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, RMP(K), \Delta^{[x]}(\tau)\right|C_{\max}$, in accordance with Procedure RMP1 fix outcomes (A1) and (A2), and for a particular outcome of Decision (B1), introduce a schedule $S_{\mathrm{B1}}(k)$ for an auxiliary problem $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, RMP(k-1), \Delta^{[x]}(\tau)\right|C_{\max}$ associated with certain outcomes of Decisions (B2) and (B3). In schedule $S_{\mathrm{B1}}(k)$, the jobs are organized into groups $N^{[x]}$, $1 \leq x \leq k$, and each group $N^{[x]}$ contains $n^{[x]}$ jobs, where $\sum_{x=1}^{k} n^{[x]} = n$. The jobs in $N^{[x]}$ are sequenced in accordance with a permutation $\pi^{[x]} = \left(\pi^{[x]}(1), \pi^{[x]}(2), \ldots, \pi^{[x]}\left(n^{[x]}\right)\right)$, $1 \leq x \leq k$.

For schedule $S_{\mathrm{B1}}(k)$, denote the total processing time of the jobs assigned to group $x$ by $F_x$. In accordance with (17.4), the duration $\Delta^{[x]}(\tau)$ of the RMP scheduled after the $x$th group is given by

$$T_x = \zeta^{[x]} F_x + \eta^{[x]}, \; 1 \leq x \leq k-1. \tag{17.6}$$

Let us denote the total duration of the first $r$ jobs in a group $x$ by $F_{(x,r)}$, $1 \leq x \leq k$, $1 \leq r \leq n^{[x]}$. It follows from (17.3) that $F_{(x,1)} = p_{\pi^{[x]}(1)}$. The actual processing time of the second job in the group is given by $p_{\pi^{[x]}(2)} + a^{[x]} F_{(x,1)}$, which implies that

$$F_{(x,2)} = F_{(x,1)} + \left(p_{\pi^{[x]}(2)} + a^{[x]} F_{(x,1)}\right) = \left(1 + a^{[x]}\right) p_{\pi^{[x]}(1)} + p_{\pi^{[x]}(2)}.$$

Similarly, for the third job in group $x$, the actual processing time is given by $p_{\pi^{[x]}(3)} + a^{[x]} F_{(x,2)}$, so that

$$\begin{aligned} F_{(x,3)} &= F_{(x,2)} + \left(p_{\pi^{[x]}(3)} + a^{[x]} F_{(x,2)}\right) = \left(1 + a^{[x]}\right)\left(\left(1 + a^{[x]}\right) p_{\pi^{[x]}(1)} + p_{\pi^{[x]}(2)}\right) + p_{\pi^{[x]}(3)} \\ &= \left(1 + a^{[x]}\right)^2 p_{\pi^{[x]}(1)} + \left(1 + a^{[x]}\right) p_{\pi^{[x]}(2)} + p_{\pi^{[x]}(3)}. \end{aligned}$$

Extending, we deduce that for the jobs in the first $r$ positions in group $x$, the formula

$$F_{(x,r)} = \sum_{u=1}^{r} \left(1 + a^{[x]}\right)^{r-u} p_{\pi^{[x]}(u)}, \; 1 \leq r \leq n^{[x]}, 1 \leq x \leq k,$$

holds.

Thus, the total time it takes to process all jobs in a group $x$ can be given by

$$F_x = F_{\left(x,n^{[x]}\right)} = \sum_{r=1}^{n^{[x]}} \left(1 + a^{[x]}\right)^{n^{[x]}-r} p_{\pi^{[x]}(r)},\ 1 \le x \le k. \tag{17.7}$$

The makespan of a schedule $S_{\mathrm{B1}}(k)$ is the sum of the durations of all scheduled groups and the RMPs and is given by

$$C_{\max}(S_{\mathrm{B1}}(k)) = F_1 + T_1 + F_2 + T_2 + \cdots + F_{k-1} + T_{k-1} + F_k,$$

where $T_x$ is the duration of the RMP scheduled after the $x$th group. Substituting the value of $T_x$ from (17.6) into the above equation, we get

$$C_{\max}(S_{\mathrm{B1}}(k)) = \sum_{x=1}^{k-1}(1 + \zeta^{[x]})F_x + F_k + \sum_{x=1}^{k-1} \eta^{[x]}.$$

Now substituting the value of $F_x$ from (17.7), we get

$$\begin{aligned} C_{\max}(S_{\mathrm{B1}}(k)) = &\sum_{x=1}^{k-1}\sum_{r=1}^{n^{[x]}}(1 + \zeta^{[x]})\left(1 + a^{[x]}\right)^{n^{[x]}-r} p_{\pi^{[x]}(r)} \\ &+ \sum_{r=1}^{n^{[k]}} \left(1 + a^{[k]}\right)^{n^{[k]}-r} p_{\pi^{[k]}(r)} + \sum_{x=1}^{k-1} \eta^{[x]}. \end{aligned} \tag{17.8}$$

We see that in (17.8) a job $j$ scheduled in position $r$ of group $x$, $1 \le x \le k-1$, $1 \le r \le n^{[x]}$, will contribute to its normal processing time $p_j = p_{\pi^{[x]}(r)}$ taken $(1 + \zeta^{[x]})\left(1 + a^{[x]}\right)^{n^{[x]}-r}$ times. If job $j$ is scheduled in the last group $x = k$, then its contribution is $\left(1 + a^{[k]}\right)^{n^{[k]}-r}$ times $p_j$. This implies that the objective function (17.8) can be written as a generic function of the form

$$C_{\max}(S_{\mathrm{B1}}(k)) = \sum_{x=1}^{k}\sum_{r=1}^{n^{[x]}} W^{[x]}(r) p_{\pi^{[x]}(r)} + \Gamma(k), \tag{17.9}$$

where the constant term is defined by

$$\Gamma(k) = \sum_{x=1}^{k-1} \eta^{[x]}, \tag{17.10}$$

and

$$W^{[x]}(r) = \begin{cases} (1+\zeta^{[x]})\left(1+a^{[x]}\right)^{n^{[x]}-r}, & 1 \le r \le n^{[x]},\ 1 \le x \le k-1, \\ \left(1+a^{[k]}\right)^{n^{[k]}-r}, & 1 \le r \le n^{[x]},\ x = k, \end{cases} \tag{17.11}$$

is a job-independent weight, such that the product $W^{[x]}p_{\pi^{[x]}(r)}$ represents the contribution of job $j = \pi^{[x]}(r)$ scheduled in position $r$, $1 \le r \le n^{[x]}$, of group $x$, $1 \le x \le k$, to the objective function (17.8).

The function (17.9) admits a generic representation (12.3), and Procedure RMP1 is applicable. Notice that the weights are job-independent, so that for each outcome of Decision (B1) the corresponding linear assignment problem will have a product cost matrix (see Sect. 12.4).

Similarly to Chap. 16, below we consider eight versions of problem $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, RMP(K), \Delta^{[x]}(\tau)\right|C_{\max}$ distinguishing between them based on three criteria:

**(a)** the RMPs are identical or distinct;
**(b)** deterioration factors are group-independent or group-dependent, and
**(c)** durations of the RMPs are constant or defined by (17.4).

We present three solution approaches, which handle different versions of problem $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, RMP(K), \Delta^{[x]}(\tau)\right|C_{\max}$. All three approaches are able to take Decision (B1), i.e., to find the optimal values of $n^{[x]}$, $1 \le x \le k$, on the fly. The approaches differ in how outcomes (A1) and (A2) are generated. Based on these differences, we study different versions of the main problem in three separate subsections. The three solution approaches require different running times, but they all use a subroutine which implements the ideas of Algorithm Match; i.e., the corresponding optimal permutation of jobs is obtained by matching jobs with large normal processing times to small positional weights.

Table 17.1 lists different versions of problem $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, RMP(K),\right.$ $\Delta^{[x]}(\tau)|C_{\max}$ considered in this chapter and gives references to the appropriate sections. In the rows of Table 17.1 and several other tables presented later in this chapter, we write either GI or GD, depending on whether the deterioration factors are group-independent or group-dependent, respectively.

**Table 17.1** Different versions of problem $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, RMP(K), \Delta^{[x]}\right|C_{\max}$

| | Constant duration RMP | | Variable duration RMP | |
|---|---|---|---|---|
| | Identical | Distinct | Identical | Distinct |
| GI | Sect. 17.4 | Sect. 17.4 | Sect. 17.3 | Sect. 17.2 |
| GD | Sect. 17.3 | Sect. 17.2 | Sect. 17.2 | Sect. 17.2 |

## 17.2 Reduction to LAP with a Product Matrix

In this section, we describe a solution approach which solves problem $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, RMP(K), \Delta^{[x]}(\tau)\right|C_{\max}$ in the most general setting, i.e., when the deterioration rates are group-dependent and the $K$ RMPs are known to be distinct, with start-time-dependent durations. We describe a solution approach, which is able to solve the auxiliary problem $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, RMP(k-1), \Delta^{[x]}(\tau)\right|C_{\max}$ without prior knowledge of the number $n^{[x]}$ of jobs in each group. The only condition is that the computed positional weights $W^{[x]}(r)$, $1 \le r \le n^{[x]}$, should follow

$$W^{[x]}(1) \ge W^{[x]}(2) \ge \cdots \ge W^{[x]}(n^{[x]}), \; 1 \le x \le k. \tag{17.12}$$

Set the value $n^{[x]} = n$, $1 \le x \le k$, and compute all possible positional weights $W^{[x]}(r)$, $1 \le r \le n$, $1 \le x \le k$, by (17.11), where for convenience we denote $U^{[x]} := \left(1 + \zeta^{[x]}\right)$, $1 \le x \le k-1$, and $U^{[k]} := 1$. The resulting $n \times k$ matrix is given by

$$\begin{pmatrix} U^{[1]}\left(1+a^{[1]}\right)^{n-1} & U^{[2]}\left(1+a^{[2]}\right)^{n-1} & \cdots & U^{[k]}\left(1+a^{[k]}\right)^{n-1} \\ U^{[1]}\left(1+a^{[1]}\right)^{n-2} & U^{[2]}\left(1+a^{[2]}\right)^{n-2} & \cdots & U^{[k]}\left(1+a^{[k]}\right)^{n-2} \\ \vdots & \vdots & \cdots & \vdots \\ U^{[1]}\left(1+a^{[1]}\right)^{2} & U^{[2]}\left(1+a^{[2]}\right)^{2} & \cdots & U^{[k]}\left(1+a^{[k]}\right)^{2} \\ U^{[1]}\left(1+a^{[1]}\right) & U^{[2]}\left(1+a^{[2]}\right) & \cdots & U^{[k]}\left(1+a^{[k]}\right) \\ U^{[1]} & U^{[2]} & \cdots & U^{[k]} \end{pmatrix}. \tag{17.13}$$

Each column of the above matrix represents all possible positional weights that can be associated with a particular group. The first element of column $x$ represents a weight associated with the first position in group $x$, while the last element of column $x$ represents a weight associated with the last, i.e., the first from the rear end position of group $x$, $1 \le x \le k$. We allow up to $n$ positions in each group. Notice that the elements of each column form a non-increasing sequence of the weights, so that (17.12) holds.

The following statement explains how problem $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau,\right.$ $\left.RMP\,(k-1), \Delta^{[x]}(\tau)\right|C_{\max}$ can be solved, without prior knowledge of the values $n^{[x]}$, $1 \le x \le k$.

**Theorem 17.1** *Given problem* $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, RMP(k-1), \Delta^{[x]}(\tau)\right|$ $C_{\max}$, *where* $1 \le k \le K+1$, *compute the matrix (17.13) of all possible positional weights* $W^{[x]}(r)$ *and choose the n smallest among them. If in each column the chosen elements occupy consecutive positions starting from the last row, then assigning the jobs with the largest normal processing times to the positions associated with the smallest positional weights will ensure that the objective function (17.9) is minimized.*

Notice that Theorem 17.1 is only applicable to solving those scheduling problems with changing processing times, for which all possible positional weights can be computed in advance, which is not the case, e.g., for problems of minimizing the total flow time or for problems with combined effects. We study these problems in Chap. 18.

The proof of Theorem 17.1 is straightforward and is similar to the proof of Theorem 16.2. According to Theorem 16.2, an optimal schedule can be found by choosing the $n$ smallest positional weights and assigning the jobs with the largest processing times to the positions corresponding to the smallest positional weights. Notice that the $n$ smallest positional weights are found in consecutive positions of the columns at the bottom of the matrix (17.13). The smallest positional weight in a group is associated with the last position in that group, irrespective of the number of jobs in that group, i.e., for group $x$, in accordance with (17.11) the weight $W^{[x]}\left(n^{[x]}\right) = U^{[x]}$. The next smallest positional weight in group $x$ is located immediately above in the same column, and so on.

The problem of finding the $n$ smallest positional weights and matching them to the appropriate jobs is structurally similar to that of scheduling jobs on uniform parallel machines to minimize the total flow time (see Sect. 2.3.1 for details). According to this method, the jobs are scanned in the LPT order and the machines are filled in the reversed order, from the last position to the first one.

Adapting this approach to our problem, consider the jobs in the LPT order. To assign the first job, compare the $k$ multipliers $U^{[x]}$, $1 \le x \le k$, and assign the job to the last position of the group associated with the smallest value of $U^{[x]}$, $1 \le x \le k$. The next positional weight that can be taken from this group is computed and replaces the previously used one. The process continues, and for the current job the smallest of the $k$ available positional weights determines the group and the position within the group, where the job should be assigned. This approach does not require any advance knowledge of the number of jobs $n^{[x]}$ in each group or, in fact, even an advance knowledge of the full matrix (17.13).

A formal description of the algorithm is given below.

**Algorithm NSmallTime**

Input: An instance of problem $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, RMP(k-1), \Delta^{[x]}(\tau)\right|C_{\max}$ with the jobs renumbered in the LPT order

Output: An optimal schedule $S^*(k)$ defined by the processing sequences $\pi^{[x]}$, $1 \le x \le k$

**Step 1**. For each group $x$, $1 \le x \le k$, define an empty processing sequence $\pi^{[x]} := (\varnothing)$ and the weight $W^{[x]} = U^{[x]}$. Create a non-decreasing list $\Omega$ of the values $W^{[x]}$, $1 \le x \le k$; to break ties, we place the weight associated with a group with a smaller index $x$ earlier in the list.

**Step 2**. For each job $j$ from 1 to $n$ do

**(a)** Take the first element $W^{[v]}$ in list $\Omega$, the smallest available positional weight.

**(b)** Assign job $j$ to group $v$ and place it in front of the current permutation $\pi^{[v]}$, i.e., update $\pi^{[v]} := (j, \pi^{[v]})$ and associate job $j$ with the positional weight

$W^{[v]}$. Remove $W^{[v]}$ from list $\Omega$. Update $W^{[v]} := W^{[v]}\left(1 + a^{[v]}\right)$ and insert the updated value $W^{[v]}$ into $\Omega$, while maintaining list $\Omega$ non-decreasing.

**Step 3**. With the found permutation $\pi^* = \left(\pi^{[1]}, \pi^{[2]}, \ldots, \pi^{[k]}\right)$, compute the optimal value of the objective function $C_{\max}(S^*(k))$ by substituting appropriate values in (17.9).

In Step 1 of Algorithm NSmallTime, list $\Omega$ can be created in $O(k \log k)$ time. Each iteration of the loop in Step 2 requires $O(\log k)$ time, since the insertion of the updated weight into a sorted list can be done by binary search. Since $n \geq k$, the following statement holds.

**Lemma 17.1** *For an auxiliary problem* $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, RMP(k-1), \Delta^{[x]}(\tau)\right|C_{\max}$ *under a time-dependent deterioration effect, Algorithm NSmallTime finds an optimal schedule* $S^*(k)$ *in* $O(n \log n)$ *time, or in* $O(n \log k)$ *time, provided that the LPT sequence of the jobs is known.*

Thus, in the case under consideration, we modify Procedure RMP1 by replacing Step 1(b) and (c) by the following:

**Step 1(b′)**. For each outcome of Decision B(1) do

Find schedule $S^*(k)$ by applying Algorithm NSmallTime.

*Example 17.2* We illustrate Algorithm NSmallTime by solving the instance described in Example 17.1, provided that three RMPs chosen from the original list are RMP$^{[3]}$, RMP$^{[4]}$, and RMP$^{[1]}$, and they are scheduled in this order. The parameters $\zeta^{[y]}$, $\eta^{[y]}$, and $a^{[y]}$ are renumbered according to the order in which the groups appear in the schedule. After renumbering, the parameters become

$$\begin{aligned} & & a^{[1]} &= 0.10;\\ \zeta^{[1]} = 0.025, & \quad \eta^{[1]} = 6, & a^{[2]} &= 0.25;\\ \zeta^{[2]} = 0.150, & \quad \eta^{[2]} = 2, & a^{[3]} &= 0.20;\\ \zeta^{[3]} = 0.050, & \quad \eta^{[3]} = 10, & a^{[3]} &= 0.15. \end{aligned}$$

The computation is shown in Table 17.2. For each job, the chosen positional weight is shown in a box in the previous row.

In the resulting schedule, the sequence of jobs (4, 2) forms Group 1, while Group 2 and Group 3 consist of job 5 and job 3, respectively, and the last Group 4 processes the sequence of jobs (6, 1). The makespan of this schedule can be computed by (17.9), where $\Gamma(3)$ is equal to $\eta^{[1]} + \eta^{[2]} + \eta^{[3]} = 18$, so that the resulting makespan is equal to $34.6575 + 18 = 52.6575$.

As suggested in Procedure RMP1, to determine an optimal solution for the general problem $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, RMP(K), \Delta^{[x]}(\tau)\right|C_{\max}$, all options associated with outcomes (A1) and (A2) must be enumerated and the solutions of

**Table 17.2** Run of Algorithm NSmallTime for Example 17.2 with the chosen sequence of MPs

| $j$ | $p_j$ | Group 1 | | Group 2 | | Group 3 | | Group 4 | | Contribution of job $j$ |
|---|---|---|---|---|---|---|---|---|---|---|
| | | $W^{[1]}$ | $\pi^{[1]}$ | $W^{[2]}$ | $\pi^{[2]}$ | $W^{[3]}$ | $\pi^{[2]}$ | $W^{[4]}$ | $\pi^{[4]}$ | |
| | | 1.025 | $\varnothing$ | 1.15 | $\varnothing$ | 1.05 | $\varnothing$ | $\boxed{1}$ | $\varnothing$ | |
| 1 | 10 | $\boxed{1.025}$ | $\varnothing$ | 1.15 | $\varnothing$ | 1.05 | $\varnothing$ | $1 * 1.15$ | (1) | $1 * 10$ |
| 2 | 9 | $1.025 * 1.1$ | (2) | 1.15 | $\varnothing$ | $\boxed{1.05}$ | $\varnothing$ | 1.15 | (1) | $1.025 * 9$ |
| 3 | 6 | $\boxed{1.1275}$ | (2) | 1.15 | $\varnothing$ | $1.05 * 1.2$ | (3) | 1.15 | (1) | $1.05 * 6$ |
| 4 | 3 | $1.1275 * 1.1$ | (4, 2) | $\boxed{1.15}$ | $\varnothing$ | 1.26 | (3) | 1.15 | (1) | $1.1275 * 3$ |
| 5 | 3 | 1.24025 | (4, 2) | $1.15 * 1.25$ | (5) | 1.26 | (3) | $\boxed{1.15}$ | (1) | $1.15 * 3$ |
| 6 | 2 | 1.24025 | (4, 2) | 1.4375 | (5) | 1.26 | (3) | $1.15 * 1.15$ | (6, 1) | $1.15 * 2$ |

the resulting auxiliary problems $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, RMP(k-1), \Delta^{[x]}(\tau)\right|C_{\max}$ be compared.

For a known $k$, $1 \le k \le K+1$, the number of ways to select $k-1$ RMPs from $K$ available RMPs (RMP Decision 2) is equal to $\binom{K}{k-1}$. Notice that the positional weights (17.11) that are associated with the first $k-1$ groups do not depend on the order of the RMPs. However, the $k$th group is differently structured from the others, since for this group $U^{[k]} = 1$, so it matters which RMP is to be scheduled last, i.e., at the $(k-1)$th position. Thus, the number of choices for RMP Decision 3 is equal to $k-1$. Trying all possible values of $k$ (RMP Decision 1), $1 \le k \le K+1$, the total number of options to be evaluated is given by $\sum_{k=1}^{K+1}\binom{K}{k-1}(k-1) = 2^{K-1}K$. Since Algorithm NSmallTime requires $O(n\log k)$ time to run for a given $k$, $1 \le k \le K+1$, the total running time required to solve the most general version of problem $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, RMP(K), \Delta^{[x]}(\tau)\right|C_{\max}$ can be estimated as $O\left(n\sum_{k=1}^{K+1}\log k\binom{K}{k-1}(k-1)\right) = O(n2^K K\log K)$, which is linear in $n$ for a constant $K$.

Now, let us consider other less general versions of problem $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, RMP(K), \Delta^{[x]}(\tau)\right|C_{\max}$. Similar to Chap. 16, we consider 8 versions of problem $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, RMP(K), \Delta^{[x]}(\tau)\right|C_{\max}$. For each version, the auxiliary problem $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, RMP(k-1), \Delta^{[x]}(\tau)\right|C_{\max}$ is solved in $O(n\log k)$ time by applying Algorithm NSmallTime. Table 17.3 states the number possible outcomes (A1) and (A2), i.e., the number of auxiliary problems $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, RMP(k-1), \Delta^{[x]}(\tau)\right|C_{\max}$ that are needed to be solved in order to solve a version of problem $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, RMP(K), \Delta^{[x]}(\tau)\right|C_{\max}$.

Notice that if all available RMPs are identical, then only RMP Decision 1 must be taken; i.e., only an optimal number of the RMPs in the schedule has to be determined. Thus, problem $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, RMP(k-1), \Delta^{[x]}(\tau)\right|C_{\max}$ must be solved $K+1$ times. In this case, the running time required to solve

**Table 17.3** Number of auxiliary problems to solve for different versions of problem $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, RMP(K), \Delta^{[x]}(\tau)\right|C_{\max}$

| | Constant duration RMP | | Variable duration RMP | |
|---|---|---|---|---|
| | Identical | Distinct | Identical | Distinct |
| GI | Sect. 17.4 | Sect. 17.4 | Sect. 17.3 | $2^K$ |
| GD | Sect. 17.3 | $2^K$ | $K+1$ | $2^K K$ |

problem $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, RMP(K), \Delta^{[x]}(\tau)\right|C_{\max}$ can be estimated as $O\left(n\sum_{k=1}^{K+1}\log k\right) = O(nK\log K)$.

Now consider the situation when the RMPs are distinct, but the deterioration rates are group-independent, i.e., $a^{[x]} = a$, $1 \le x \le k$. This problem corresponds to a scenario, in which distinct RMPs are performed in the schedule, but they all restore the machine to the same state. Thus, the order of the RMPs is irrelevant (i.e., RMP Decision 3 need not be taken), and RMP Decision 2 regarding the choice of the RMPs to be included into a schedule is made only on the basis of the durations of the RMPs. If the durations are determined by (17.4), there is no easy way of selecting the best $k-1$ RMPs from the $K$ available RMPs. Thus, all possible selections need to be tried and this can be done in $\binom{K}{k-1}$ ways. Trying all possible values of $k$, $1 \le k \le K+1$ (i.e., taking RMP Decision 1), the total number of options can be estimated by $\sum_{k=1}^{K+1}\binom{K}{k-1} = 2^K$. Thus, the total running time required to solve this version of problem $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, RMP(K), \Delta^{[x]}(\tau)\right|C_{\max}$ can be estimated as $O\left(n\sum_{k=1}^{K+1}\binom{K}{k-1}\log k\right) = O\left(n2^K\log K\right)$, a factor of $K$ less than in the general case.

For the problem with group-dependent deterioration rates and distinct RMPs of constant durations, i.e., $\zeta^{[x]} = 0$, $1 \le x \le k-1$, the computed positional weights can be written as

$$W^{[x]}(r) = \left(1 + a^{[x]}\right)^{n^{[x]}-r},\ 1 \le r \le n^{[x]},\ 1 \le x \le k,$$

by making appropriate substitutions in (17.11). Clearly, the found positional weights are dependent on the type of the RMP, but not on their order. Thus, as above the total number of options can be given by $\sum_{k=1}^{K+1}\binom{K}{k-1}$ and the total running time required to solve this version of problem $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, RMP(K), \Delta^{[x]}(\tau)\right|C_{\max}$ can be estimated as $O\left(n2^K\log K\right)$.

The corresponding running times required to solve different versions of problem $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, RMP(K), \Delta^{[x]}(\tau)\right|C_{\max}$ are given in Table 17.4. Notice that although Algorithm NSmallTime is able to solve all eight versions of problem $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, RMP(K), \Delta^{[x]}(\tau)\right|C_{\max}$, for some cases it is possible to make RMP Decisions 1–3 on the fly by using another solution approach, which allows the optimal solution to be found faster. For such cases, a reference to the relevant section is made in Table 17.4.

**Table 17.4** Running time required for different versions of problem $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau,\right.$ $RMP(K),\ \Delta^{[x]}(\tau)\left|C_{\max}\right.$

| | Constant duration RMP | | Variable duration RMP | |
|---|---|---|---|---|
| | Identical | Distinct | Identical | Distinct |
| GI | Sect. 17.4 | Sect. 17.4 | Sect. 17.3 | $O(n2^K \log K)$ |
| GD | Sect. 17.3 | $O(n2^K \log K)$ | $O(nK \log K)$ | $O(n2^K K \log K)$ |

## 17.3 On the Fly Decision Making

In this subsection, we present a solution approach that allows us to make RMP Decisions 1–3 on the fly, without enumerating all possible options. This gives the desired outputs (A1) and (A2) and helps us to considerably reduce the running time for solving variants of the original problem $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau,\ RMP(K),\ \Delta^{[x]}(\tau)\right|$ $C_{\max}$ listed below:

**Problem Time1:** This is a version of problem $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, RMP(K),\right.$ $\Delta^{[x]}(\tau)\left|C_{\max}\right.$ with group-dependent deterioration rates and distinct RMPs of constant durations. Assume that for each RMP$^{[y]}$ from a given list, $\zeta^{[x]} = 0$, $1 \le x \le K$, holds and the RMPs can be ordered such that

$$a^{[1]} \le a^{[2]} \le \cdots \le a^{[K+1]}, \tag{17.14}$$

and

$$\eta^{[1]} \le \eta^{[2]} \le \cdots \le \eta^{[K]}, \tag{17.15}$$

hold simultaneously. This version is a generalization of one of the cases found in Table 17.1, in which group-dependent deterioration rates are considered along with identical RMPs of constant duration, i.e., $\zeta^{[x]} = 0$, $\eta^{[x]} = \eta$, $1 \le x \le K$. For the latter problem, it is reasonable to assume that (17.14) and (17.15) hold simultaneously, based on the following argument. If identical RMPs of equal duration are performed on the machine, then after an RMP the condition of the machine can be no better than its condition after the previous RMP. In such a case, the deterioration rates do not decrease as more groups are created.

**Problem Time2:** This is a version of problem $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, RMP(K),\right.$ $\Delta^{[x]}(\tau)\left|C_{\max}\right.$ with group-independent deterioration rates, i.e., $a^{[x]} = a$, $1 \le x \le K + 1$, and distinct RMPs of start-time-dependent durations, subject to the condition that the duration parameters of the RMPs can be ordered such that

$$\zeta^{[1]} \le \zeta^{[2]} \le \cdots \le \zeta^{[K]}, \tag{17.16}$$

and (17.15) hold simultaneously. This version is a generalization of one of the cases found in Table 17.1, in which group-independent positional rates are considered along with identical RMPs of start-time-dependent duration, i.e., $\zeta^{[x]} = \zeta$, $\eta^{[x]} = \eta$, $1 \le x \le K$.

In order to solve both versions of problem $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, RMP(K), \Delta^{[x]}(\tau)\right|C_{\max}$ described above, the optimal choice for RMP Decisions 2 and 3 can be made easily. If RMP Decision 1 is assumed to be taken, so that $k-1$, $1 \le k \le K+1$, RMPs are to be included in the schedule, then for both problems, RMP$^{[1]}, \ldots,$ RMP$^{[k-1]}$ are to be chosen and inserted into a schedule in the order of their numbering. For a chosen $k$, this gives the best outputs (A1) and (A2) for Problem Time1, since the chosen RMPs have the shortest durations and, moreover, the created groups will be associated with the smallest deterioration factors, since (17.14) and (17.15) hold simultaneously. This gives the best outputs (A1) and (A2) for Problem Time2 as well, since the groups are identical from the point of view of the associated positional factors, and the RMPs with smaller indices have smaller values of the duration parameters, because (17.15) and (17.16) hold simultaneously.

For an assumed value of $k$, let RMP Decisions 2–3 be made, and let problem $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, RMP(k-1), \Delta^{[x]}(\tau)\right|C_{\max}$ be the resulting auxiliary problem. This problem can be solved by minimizing the generic objective function of the form (17.9). For Problem Time1, obtain the required positional weights $W^{[x]}(r)$ by substituting $\zeta^{[x]} = 0$, $1 \le x \le k$, in (16.18) so that

$$W^{[x]}(r) = \left(1 + a^{[x]}\right)^{n^{[x]}-r},\ 1 \le r \le n^{[x]},\ 1 \le x \le k, \tag{17.17}$$

while for Problem Time2, substitute $a^{[x]} = a$, $1 \le x \le k$, so that

$$W^{[x]}(r) = \begin{cases} (1+\zeta^{[x]})(1+a)^{n^{[x]}-r}, & 1 \le r \le n^{[x]},\ 1 \le x \le k-1, \\ (1+a)^{n^{[x]}-r}, & 1 \le r \le n^{[x]},\ x = k. \end{cases} \tag{17.18}$$

Set the value $n^{[x]} = n$, $1 \le x \le k$, and $k = K+1$, and compute all positional weights $W^{[x]}(r)$, $1 \le r \le n$, $1 \le x \le K+1$, for both problems by using the formulae above. Notice that the computed positional weights represent a set of all possible values of $W^{[x]}(r)$ across all possible groups.

Notice that both Problems Time1 and Time2 satisfy the *K-domi* condition, defined in Definition 16.1. Inequality (17.14) ensures that the positional weights associated with Problem Time1 are ordered so that for each $k$, $1 \le k \le K+1$, we have

$$W^{[1]}(r) \le W^{[2]}(r) \le \cdots \le W^{[k]}(r),\ 1 \le r \le n,$$

while because of (17.16), the positional weights for Problem Time2 are ordered so that for each $k$, $1 \le k \le K+1$, we have

$$W^{[k]}(r) \le W^{[1]}(r) \le W^{[2]}(r) \le \cdots \le W^{[k-1]}(r),\ 1 \le r \le n.$$

These observations guarantee the required dominance for any pair of groups in any of these two problems.

For instances of problem $1\left|p_j^{[x]}(\tau)=p_j+a^{[x]}\tau, RMP(K), \Delta^{[x]}(\tau)\right|C_{\max}$ that simultaneously satisfy (17.15) and the *K-domi* condition, it is possible to compute the optimal values of $n^{[x]}$, $1 \le x \le k$, and take all RMP Decisions 1–3 on the fly. Recall that for a fixed outcome of RMP Decision 1, the optimal outcomes for RMP Decisions 2 and 3 are already known. Thus, in order to solve these problems, we apply the following methodology that is based on similar ideas as Algorithm NSmallPosi2 outlined in Sect. 16.2.3.

Recall that Algorithm NSmallPosi2 is implemented by manipulating two lists, which we denote by $G(k-1)$ and $H(k)$. List $G(k-1)$ contains all the positional weights corresponding to the positions used in the previously found schedule $S^*(k-1)$, while list $H(k)$ contains the positional weights that will be introduced if the $k$th group is opened.

List $H(v)$, $1 \le v \le K+1$, is defined differently for Problems Time1 and Time2. For Problem Time1, $H(v)$ contains the positional weights $W^{[v]}(r)$, $v \le r \le n$, so that by (17.17) we have

$$H(v) := \begin{pmatrix} \left(1+a^{[v]}\right)^{n-v} \\ \left(1+a^{[v]}\right)^{n-(v+1)} \\ \vdots \\ \left(1+a^{[v]}\right)^{2} \\ \left(1+a^{[v]}\right) \\ 1 \end{pmatrix}, \ 1 \le v \le K+1.$$

For Problem Time2, notice that the values of the positional weights given by (17.18) change dynamically as the value of $k$ is changed. Thus, we define $H(v)$ so that this effect is incorporated: Initialize

$$H(1) := \begin{pmatrix} (1+a)^{n-1} \\ (1+a)^{n-2} \\ \vdots \\ (1+a)^{2} \\ (1+a) \\ 1 \end{pmatrix}$$

and define

$$H(v) := \begin{pmatrix} U^{[v-1]}(1+a)^{n-v} \\ U^{[v-1]}(1+a)^{n-(v+1)} \\ \vdots \\ U^{[v-1]}(1+a)^{2} \\ U^{[v-1]}(1+a) \\ U^{[v-1]} \end{pmatrix}, \ 2 \le v \le K+1,$$

where $U^{[v-1]} = \left(1+\zeta^{[v-1]}\right)$, $2 \le v \le K+1$.

Notice that for both problems, list $H(v)$ has at most $n - v + 1$, $1 \le v \le K + 1$, elements sorted in non-increasing order. It suffices to consider only $n - v + 1$ positions in list $H(v)$, since condition *K-domi* guarantees that each of the $v - 1$ earlier groups will have at least one job scheduled in it.

For $k = 1$, list $G(1)$ is defined essentially as a copy of list $H(1)$. For each $k$, $2 \le k \le K + 1$, list $G(k)$ is obtained be merging the lists $G(k - 1)$ and $H(k)$ and taking the $n$ smallest elements of this merger, keeping them in non-decreasing order.

Define $P(S^*(k))$ as the sum of actual durations of the jobs in an optimal schedule with $k$ groups. Let $\gamma_i(k)$ denote the $i$th element in the sorted list $G(k)$, so that $G(k) = (\gamma_1(k), \gamma_2(k), \dots, \gamma_n(k))$. This implies that

$$P(S^*(k)) = \sum_{j=1}^{n} p_j \gamma_j(k),$$

so that

$$C_{\max}\big(S^*(k)\big) = P(S^*(k)) + \Gamma(k) = \sum_{j=1}^{n} p_j \gamma_j(k) + \sum_{x=1}^{k-1} \eta^{[x]}. \tag{17.19}$$

where $\Gamma(k)$ is a constant term as defined in (17.10).

The following algorithm solves an instance of problem $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, RMP(K), \Delta^{[x]}(\tau)\right|C_{\max}$ and returns the optimal number of RMPs, $k^* - 1$, to be included in the schedule (RMP Decision 1) along with the optimal schedule $S^*(k^*)$ with $k^*$ groups.

**Algorithm NSmallTime2**

Input: An instance of either Problem Time1 or Problem Time2 with the jobs renumbered in the LPT order
Output: An optimal schedule $S^*(k^*)$ defined by the processing sequences $\pi^{[x]}$, $1 \le x \le k^*$

**Step 1**. For $k = 1$, find list $H(1)$ for the problem at hand. Define a sorted list $G(1)$ by reordering the elements of $H(1)$ in the opposite order. Compute $C_{\max}(S^*(1))$ by formula (17.19). Define $k' := K + 1$.
**Step 2**. For $k$ from 2 to $k'$ do

- **(a)** Create the list $G(k) = (\gamma_1(k), \gamma_2(k), \dots, \gamma_n(k))$ that contains $n$ smallest elements in the merger of the lists $G(k - 1)$ and $H(k)$.
- **(b)** Compute $C_{\max}(S^*(k))$ by formula (17.19). If $P(S^*(k)) = P(S^*(k - 1))$, then define $k' := k - 1$ and break the loop by moving to Step 3; otherwise, continue the loop with the next value of $k$.

**Step 3**. Find the value $k^*$, $1 \le k^* \le k'$, such that

$$C_{\max}\big(S^*(k^*)\big) = \min\big\{C_{\max}\big(S^*(k)\big) | 1 \le k \le k'\big\}.$$

**Step 4**. Run Algorithm NSmallTime for the found value of $k^*$ to obtain the optimal processing sequence $\pi^* = \left(\pi^{[1]}, \pi^{[2]}, \ldots, \pi^{[k^*]}\right)$.

Notice that similar to Algorithm NSmallTime, Step 2 of the above algorithm also follows Theorem 17.1 and searches for the $n$ smallest positional weights for a given $k$, $1 \le k \le K + 1$, and assigns the jobs with largest values of $p_j$ to the positions corresponding to the smallest positional weights. The main difference between the two algorithms lies in the way the list of $n$ smallest positional weights is found. For each $k$, $1 \le k \le K + 1$, Algorithm NSmallTime searches for the $n$ smallest positional weights by comparing the positional weights across all groups. On the other hand, Algorithm NSmallTime2 in each iteration compares the elements in only two lists, $G(k-1)$ and $H(k)$. Our method is justified, because list $G(k-1)$ already contains the $n$ smallest positional weights coming from the first $k-1$ groups. Thus, to search for the $n$ smallest weights needed for schedule $S^*(k)$, there is no need to scan the first $k-1$ groups again. In other words, we utilize the fact that if a certain position in the first $k-1$ groups is not used in schedule $S^*(k-1)$, then it will not be used in schedule $S^*(k)$ either.

In Step 2, for every $k$ each list $G(k-1)$ and $H(k)$ has at most $n$ elements sorted in a non-decreasing order; therefore, each Step 2(a) and Step 2(b) can be completed in $O(n)$ time. If the loop in Step 2 is not broken throughout the run of Algorithm NSmallTime2, the final value of $k'$ remains equal to $K + 1$; i.e., it is possible all RMPs will be run and all groups will be opened in an optimal schedule. The loop in Step 2 may be stopped when in Step 2b the condition $P(S^*(k)) = P(S^*(k-1))$ is achieved. This condition implies that the opening of the $k$th group does not provide any positional weights smaller than those contained in the list $G(k-1)$. If this happens for the $k$th group, all groups that could be opened after this would provide even worse positional weights, because list $H(k+1)$ is dominated by list $H(k)$, $1 \le k \le K$. Thus, the makespan cannot be reduced by running more RMPs after the $k'$th group is opened, so that there is no need to examine further values of $k$. With the found value of $k'$, the overall optimal schedule will be found among the schedules $S^*(k)$, $1 \le k \le k'$.

**Theorem 17.2** *Algorithm NSmallTime2 solves an instance either of Problem Time1 or of Problem Time2 in $O(nK)$ time, provided that the LPT order of the jobs is known.*

*Example 17.3* We illustrate the working of Algorithm NSmallTime2 by an instance of Problem Time2, obtained by modifying the generic instance of Example 17.1. For Problem Time2, the deterioration rate is group-independent and is known to be equal to $a = 0.1$. Such a situation arises when each RMP is able to restore the processing conditions to an "as good as new" state. We also modify the duration parameters of the RMPs to make them obey (17.15) and (17.16) simultaneously, so that they are now given as

$$\begin{aligned}
&\mathrm{RMP}^{[1]}\colon \zeta^{[1]} = 0.025, \quad \eta^{[1]} = 2;\\
&\mathrm{RMP}^{[2]}\colon \zeta^{[2]} = 0.05, \quad \eta^{[2]} = 4;\\
&\mathrm{RMP}^{[3]}\colon \zeta^{[3]} = 0.15, \quad \eta^{[3]} = 4;\\
&\mathrm{RMP}^{[4]}\colon \zeta^{[4]} = 0.25, \quad \eta^{[4]} = 6;\\
&\mathrm{RMP}^{[5]}\colon \zeta^{[5]} = 0.25, \quad \eta^{[5]} = 6.
\end{aligned}$$

The rest of the setting of this example remains similar to Example 17.1, with the normal processing times given by

$$p_1 = 10, p_2 = 9, p_3 = 6, p_4 = 3, p_5 = 3, p_6 = 2.$$

Table 17.5 shows the details of the run of Algorithm NSmallTime2 for the above instance. Since $\gamma_r(4) = \gamma_r(3)$ for each $r$, $1 \le r \le 6$, the algorithm stops after iteration $k = 4$, so that $k' = 3$. The algorithm outputs the minimum value of the makespan from the set $\{C_{\max}(S^*(k)) | 1 \le k \le 3\}$, which is $C_{\max}(S^*(2))$. Optimal sequences

**Table 17.5** Run of Algorithm NSmallTime2 for Example 17.3

| $k = 1$ | $r$ | $H(1)$ | $G(1)$ | $\gamma_r(1)p_r$ | |
|---|---|---|---|---|---|
| | 1 | 1.61051 | 1 | 10 | |
| | 2 | 1.4641 | 1.1 | 9.9 | |
| | 3 | 1.331 | 1.21 | 7.26 | |
| | 4 | 1.21 | 1.331 | 3.993 | |
| | 5 | 1.1 | 1.4641 | 4.3923 | |
| | 6 | 1 | 1.61051 | 3.22102 | |
| | $C_{\max}(S^*(1)) = \sum_{r=1}^{6} \gamma_r(1)p_r = 38.76632$ | | | | |
| $k = 2$ | $r$ | $G(1)$ | $H(2)$ | $G(2)$ | $\gamma_r(2)p_r$ |
| | 1 | 1 | 1.500703 | 1 | 10 |
| | 2 | 1.1 | 1.364275 | 1.025 | 9.225 |
| | 3 | 1.21 | 1.24025 | 1.1 | 6.6 |
| | 4 | 1.331 | 1.1275 | 1.1275 | 3.3825 |
| | 5 | 1.4641 | 1.025 | 1.21 | 3.63 |
| | 6 | 1.61051 | | 1.24025 | 2.4805 |
| | $C_{\max}(S^*(2)) = \sum_{r=1}^{6} \gamma_r(2)p_r + \beta^{[1]} = 35.318 + 2 = 37.318$ | | | | |
| $k = 3$ | $r$ | $G(2)$ | $H(3)$ | $G(3)$ | $\gamma_r(3)p_r$ |
| | 1 | 1 | 1.53065 | 1 | 10 |
| | 2 | 1.025 | 1.3915 | 1.025 | 9.225 |
| | 3 | 1.1 | 1.265 | 1.1 | 6.6 |
| | 4 | 1.1275 | 1.15 | 1.1275 | 3.3825 |
| | 5 | 1.21 | | 1.15 | 3.45 |
| | 6 | 1.24025 | | 1.21 | 2.42 |
| | $C_{\max}(S^*(3)) = \sum_{r=1}^{6} \gamma_r(3)p_r + \beta^{[1]} + \beta^{[2]} = 35.0775 + 2 + 4 = 41.0775$ | | | | |
| $k = 4$ | $r$ | $G(3)$ | $H(4)$ | $G(4)$ | $\gamma_r(4)p_r$ |
| | 1 | 1 | 1.5125 | 1 | 10 |
| | 2 | 1.025 | 1.375 | 1.025 | 9.225 |
| | 3 | 1.1 | 1.25 | 1.1 | 6.6 |
| | 4 | 1.1275 | | 1.1275 | 3.3825 |
| | 5 | 1.15 | | 1.15 | 3.45 |
| | 6 | 1.21 | | 1.21 | 2.42 |
| | $C_{\max}(S^*(4)) = \sum_{r=1}^{6} \gamma_r(4)p_r + \beta^{[1]} + \beta^{[2]} + \beta^{[3]} = 35.0775 + 2 + 4 + 4 = 45.0775$ | | | | |

of jobs in the two groups can be obtained by running Algorithm NSmallTime and are found to be $\pi^{[1]} = (5, 3, 1)$ and $\pi^{[2]} = (6, 4, 2)$. The makespan of the resulting schedule is 37.318.

Algorithm NSmallTime2 can also be applied to solve problem $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, RMP(K), \Delta^{[x]}(\tau)\right|C_{\max}$ with group-independent deterioration rates and constant duration RMPs (both identical and distinct). This is possible since both conditions $K$-*domi* and (17.15) can be satisfied simultaneously. The required running time is again $O(nK)$. We do not discuss the solution of this problem here, since it is possible to solve it even faster using another solution approach, discussed in Sect. 17.4.

Now consider a version of problem $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, RMP(K), \Delta^{[x]}(\tau)\right| C_{\max}$ in which the conditions $K$-*domi* and (17.15) do not hold simultaneously. Algorithm NSmallTime2 can still be used to obtain an optimal value for RMP Decision 1 and to find an optimal permutation of jobs, but to make RMP Decisions 2 and 3, a full enumeration of options might be required. As a result, the overall running time to solve problem $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, RMP(K), \Delta^{[x]}(\tau)\right|C_{\max}$ turns out to be no smaller than that obtained by using the general solution approach presented in Sect. 17.2.

We view the material of this subsection as a strong evidence of similarity between the problems with time-dependent effects and positional effects. These two models have been traditionally considered as different in nature.

## 17.4 Binary Search in Convex Sequences

In this subsection, we deal with problems in which the computed positional weights are group-independent, i.e., are of the form $W^{[x]}(r) = W(r)$, $1 \le x \le k$, and additionally, they are ordered in a way such that $W(1) \ge W(2) \ge \cdots \ge W(n)$. Such a situation arises for the versions of problem $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, RMP(K), \Delta^{[x]}(\tau)\right|C_{\max}$, in which the deterioration rates are group-independent, i.e., $a^{[x]} = a$, $1 \le x \le K + 1$, while RMPs are of constant duration, i.e., $\zeta^{[x]} = 0$, $1 \le x \le K$. Formally, we denote the described problem by $1\left|p_j^{[x]}(\tau) = p_j + a\tau, RMP(K), \Delta^{[x]}\right|C_{\max}$. We focus on the problem in which RMPs are distinct, since the problem with identical RMPs is not known to admit a faster solution algorithm.

Notice that for problem $1\left|p_j^{[x]}(\tau) = p_j + a\tau, RMP(K), \Delta^{[x]}\right|C_{\max}$, the optimal choice for RMP Decisions 2 and 3 can be made easily. Assume that for problem $1\left|p_j^{[x]}(\tau) = p_j + a\tau, RMP(K), \Delta^{[x]}\right|C_{\max}$, an optimal schedule includes $k - 1$ RMPs, so that the jobs are divided into $k$, $1 \le k \le K + 1$, groups. Since it is known that the RMPs create groups associated with the same deterioration rates, it follows that the order in which the RMPs are performed is immaterial. Further, it is obvious that in order to choose $k-1$ RMPs out of the available $K$ ones, the RMPs with smaller durations must be given priority. To ensure that the shortest $k-1$ RMPs are chosen in an optimal schedule, we renumber the $K$ available RMPs in a way that (17.15) holds and selects the ones with indices $1, 2, \ldots, k-1$, and include them into a schedule in the order of their numbering.

With RMP Decisions 1–3 having been made (for an assumed value of $k$), the auxiliary problem $1\left|p_j^{[x]}(\tau) = p_j + a\tau, RMP(k-1), \Delta^{[x]}\right|C_{\max}$ can be solved by minimizing the generic objective function (17.9). Obtain the required positional weights $W^{[x]}(r)$ by substituting $a^{[x]} = a$, $\zeta^{[x]} = 0$, $1 \le x \le k$, in (17.11) so that we have

$$W^{[x]}(r) = (1+a)^{n^{[x]}-r},\ 1 \le r \le n^{[x]},\ 1 \le x \le k.$$

Notice that, unlike for most previously considered models, here the positional weights in the last group are computed similarly to all other groups.

To solve an instance of problem $1\left|p_j^{[x]}(\tau) = p_j + a\tau, RMP(k-1), \Delta^{[x]}\right|C_{\max}$, below we outline a solution approach which is again based on Theorem 17.1.

First, set the value $n^{[x]} = n$, $1 \le x \le k$, and from the resulting set of positional weights

$$\begin{pmatrix} (1+a)^{n-1} & (1+a)^{n-1} & \cdots & (1+a)^{n-1} \\ (1+a)^{n-2} & (1+a)^{n-2} & \cdots & (1+a)^{n-2} \\ \vdots & \vdots & \cdots & \vdots \\ (1+a)^2 & (1+a)^2 & \cdots & (1+a)^2 \\ (1+a) & (1+a) & \cdots & (1+a) \\ 1 & 1 & \cdots & 1 \end{pmatrix}, \tag{17.20}$$

choose the $n$ smallest of them. Obviously, the $n$ smallest weights are found in consecutive positions at the bottom of the matrix. The smallest $k$ positional weights are due to the last positions of each of the $k$ groups. The next smallest $k$ positional weights are due to the second last positions of each of the $k$ groups, and so on. Assuming that $n = \lambda k + \mu$, where $\lambda$ and $\mu$ are non-negative integers, $\mu \le k-1$, the optimal number of jobs in each group can be given by

$$n^{[x]} = \begin{cases} \left\lceil \frac{n}{k} \right\rceil = \lambda + 1, & 1 \le x \le \mu \\ \left\lfloor \frac{n}{k} \right\rfloor = \lambda, & \mu + 1 \le x \le k. \end{cases}$$

With known values of $W^{[x]}(r)$, $1 \le r \le n^{[x]}$, and $n^{[x]}$, $1 \le x \le k$, an optimal makespan $C_{\max}(S^*(k))$ for problem $1\left|p_j^{[x]}(\tau) = p_j + a\tau, RMP(k-1), \Delta^{[x]}\right|$ $C_{\max}$ can be found in $O(n)$ time by running the matching algorithm.

To determine the optimal solution for problem $1\left|p_j^{[x]}(\tau) = p_j + a\tau, RMP(K),\right.$ $\left.\Delta^{[x]}\right|C_{\max}$, all options associated with RMP Decisions 1–3 must be enumerated. RMP Decisions 2 and 3 have already been taken optimally; thus, we only need to determine the optimal value of the number of RMPs. We can do this by solving problem $1\left|p_j^{[x]}(\tau) = p_j + a\tau, RMP(k-1), \Delta^{[x]}\right|C_{\max}$ for all values of $k$, $1 \le k \le K+1$, and choosing the instance that delivers the smallest value of $C_{\max}(S^*(k))$. Thus, problem $1\left|p_j^{[x]}(\tau) = p_j + a\tau, RMP(k-1), \Delta^{[x]}\right|C_{\max}$ can be solved in $O(nK)$ time. However, we prove that the sequence $C_{\max}(S^*(k))$, $1 \le k \le K+1$, is in fact $V$-shaped and thus, in order to search for the smallest value of $C_{\max}(S^*(k))$, we only need to evaluate $\lceil \log_2(K+1) \rceil$ options of $k$, $1 \le k \le K+1$. Recall that a sequence $A(k)$ is called *V-shaped* if there exists a $k_0$, $1 \le k_0 \le K+1$, such that

$$A(1) \ge \cdots \ge A(k_0 - 1) \ge A(k_0) \le A(k_0 + 1) \le \cdots \le A(K+1).$$

If we define

$$h(q) := (1+a)^{q-1}, \ 1 \le q \le n, \tag{17.21}$$

then a column of the matrix (17.20) of positional weights becomes

$$\begin{pmatrix} h(n-1) \\ h(n-2) \\ \vdots \\ h(2) \\ h(1) \end{pmatrix}.$$

For a schedule $S(k)$ with $k$ groups, let $P(S(k))$ denote the sum of the actual durations of the jobs, and $\Gamma(k)$ be the total duration of all $k-1$ RMPs defined by (17.10).

**Lemma 17.2** *For problem* $1\left|p_j^{[x]}(\tau) = p_j + a\tau, RMP(k-1), \Delta^{[x]}\right|C_{\max}$, *if the jobs are numbered in the LPT order, then the makespan of the optimal schedule can be written as*

$$C_{\max}(S^*(k)) = P(S^*(k)) + \Gamma(k) = \sum_{j=1}^{n} p_j h\left(\left\lceil \frac{j}{k} \right\rceil\right) + \sum_{x=1}^{k-1} \eta^{[x]}, \ 1 \le k \le K+1, \tag{17.22}$$

*where we denote* $h(q) := (1+a)^{q-1}, \ 1 \le q \le n.$

*Proof* It is clear that $C_{\max}(S(k)) = P(S(k)) + \Gamma(k)$. If the jobs are numbered in the LPT order, then to minimize the value $P(S(k))$ we need to assign the jobs one by one to an available position with the smallest positional weight. Irrespective of the number of jobs in each group, this can be done by distributing the first $k$ jobs to the last positions in each of the $k$ groups, then the next $k$ jobs to the second last positions in each of the $k$ groups, and so on, until all jobs have been sequenced.

If $j = \lambda k$, then the predecessors of $j$ are placed into the last $\lambda$ positions of groups $1, 2, \ldots, k-1$ and the last $\lambda - 1$ positions of group $k$, so that job $j$ is assigned to group $k$ and gets position $\lambda = \left\lceil \frac{j}{k} \right\rceil$ from the rear. If $j = \lambda k + \mu$ for $1 \le \mu \le k-1$, then the predecessors of $j$ take the last $\lambda$ positions in each group and additionally the $(\lambda+1)$th last position in each of the groups $1, 2, \dots, \mu - 1$, so that job $j$ gets position $\lambda + 1 = \left\lceil \frac{j}{k} \right\rceil$ from the rear in group $\mu$.

It follows that in an optimal schedule $S^*(k)$, the contribution of a job $j \in N$ to the objective function is equal to $p_j(1+a)^{\left\lceil \frac{j}{k} \right\rceil - 1}$, and the total processing time for all jobs is equal to

$$P(S^*(k)) = \sum_{j=1}^{n} p_j h\left(\left\lceil \frac{j}{k} \right\rceil\right), \tag{17.23}$$

as required. □

**Theorem 17.3** *For problem* $1\left|p_j^{[x]}(\tau) = p_j + a\tau, RMP(K), \Delta^{[x]}\right|C_{\max}$, *the sequence* $C_{\max}(S^*(k))$, $1 \le k \le K+1$, *given by (17.22), is* $V-$*shaped.*

Notice that the sequence $h(q) := (1+a)^{q-1}$, $1 \le q \le n$, is non-decreasing, so the proof of Theorem 17.3 is similar to the proof of Theorem 16.5.

Theorem 17.3 allows us to find an optimal schedule with an optimal number of groups $k^*$ by performing binary search in sequence $C_{\max}(S^*(k))$, $1 \le k \le K+1$, with respect to $k$. This can be done by an appropriate adaptation of Algorithm BinSearchPosi from Sect. 16.2.4. As a result, at most $\lceil \log_2(K+1) \rceil$ values of $k$ need to be explored, so that the optimal solution for problem $1\left|p_j^{[x]}(\tau) = p_j + a\tau, RMP(K),\right.$ $\left.\Delta^{[x]}\right|C_{\max}$ can be found in $O(n \log K)$ time.

As in Sect. 17.3, the approach we use for the problem with group-independent start-time-dependent effects is very similar to the one used in Sect. 16.2.4 for scheduling with group-independent positional effects. This is another evidence that the two types of models with changing processing times are closer than has been perceived (see Corollary 8.2 that establishes the equivalence of some single machine problems with additive start-time-dependent effects and with positional effects).

Table 17.6 summarizes all the problems considered in this paper along with the running times needed to solve them, provided that the jobs are taken in the LPT order of their normal processing times.

**Table 17.6** Computational complexities of different versions of problem $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau,\right.$ $\left.RMP(K), \Delta^{[x]}(\tau)\right|C_{\max}$

| | Constant duration RMP | | Variable duration RMP | |
|---|---|---|---|---|
| | Identical | Distinct | Identical | Distinct |
| GI | $O(n \log K)$ | $O(n \log K)$ | $O(nK)$ | $O\left(n2^K \log K\right)$ |
| GD | $O(nK)$ | $O\left(n2^K \log K\right)$ | $O(nK \log K)$ | $O\left(n2^K K \log K\right)$ |

## 17.5 Bibliographic Notes

Although many papers have considered the problem of time-dependent deterioration with maintenance periods, most of them, however, only see an RMP as a fixed non-availability period, which does not necessarily improve the machine conditions. Only a handful of studies have considered problems in which an RMP is actually used to restore the machine to its original state, so that the effects of time-dependent deterioration are repaired.

Lodree and Geiger (2010) study problem $1\left|p_j^{[x]}(\tau) = a_j\tau, RMP(1), \Delta\right|C_{\max}$, with a time-dependent effect of the form $p_j(\tau) = a_j\tau, a_j \geq 1$, and a single RMP in the schedule. They provide an optimal policy to determine the number of jobs to be included in each of the two created groups. According to this policy, if $n$ is even, both groups should contain $\left(\frac{n}{2}+1\right)$ jobs, whereas if $n$ is odd, one group should contain $\left(\frac{n+1}{2}\right)$ jobs and the other should contain $\left(\frac{n+3}{2}\right)$ jobs.

Rustogi and Strusevich (2015) solve the problem of minimizing the makespan for a linear time-dependent deterioration effect which is enhanced by including various maintenance activities in the schedule. They show that even in the presence of distinct maintenance activities and group dependence, the problem of minimizing the makespan can be solved by an efficient polynomial algorithm, which does not involve full enumeration. Rustogi and Strusevich (2015) study enhanced models that allow the deterioration rates to be group-dependent and the RMPs to have different duration parameters. Similar to the structure of this chapter, they propose three solution approaches that solve different versions of problem $1\left|p_j^{[x]}(\tau) = p_j + a^{[x]}\tau, RMP(K), \Delta^{[x]}\right|C_{\max}$. Most of the material presented in this chapter is based on Rustogi and Strusevich (2015).

Notice that to the best of our knowledge, apart from the problems studied by Lodree and Geiger (2010) and Rustogi and Strusevich (2015), all other problems related to time-dependent effects and rate-modifying activities require the prior knowledge of the number of jobs in each group, and therefore, the resulting running times contain at least a multiple $n^K$, since the optimal number of jobs in each group is often found by full enumeration. Yang and Yang (2010) study problem $1\left|p_j^{[x]}(\tau) = p_j + a\tau, RMP(K), \Delta\right|\sum C_j$, with a linear time-dependent deterioration effect with a known number, $K$, of RMPs and provide an optimal solution in $O\left(n^{K+1}\log n\right)$ time. The same method is extended by Yang (2010) and Yang (2012), to study different versions of problem $1\left|p_j^{[x]}(r,\tau) = \left(p_j + a\tau\right)r^b, RMP(K), \Delta\right|F$, $F \in \left\{C_{\max}, \sum C_j\right\}$, with a combined effect and a known number, $K$, of RMPs. For their respective models, both papers claim to solve the problem of minimizing the makespan and the problem of minimizing the total flow time in $O\left(n^{K+1}\log n\right)$ time each. Rustogi and Strusevich (2014) also show that the running times of $O\left(n^{K+1}\log n\right)$ are applied to a range of much more general problems with group-dependent combined effects (see Chap. 18).

## References

Lodree EJ Jr, Geiger CD (2010) A note on the optimal sequence position for a rate-modifying activity under simple linear deterioration. Eur J Oper Res 201:644–648

Rustogi K, Strusevich VA (2014) Combining time and position dependent effects on a single machine subject to rate-modifying activities. Omega 42:166–178

Rustogi K, Strusevich VA (2015) Single machine scheduling with time-dependent linear deterioration and rate-modifying maintenance. J Oper Res Soc 66:500–515

Yang SJ (2010) Single-machine scheduling problems with both start-time dependent learning and position dependent aging effects under deteriorating maintenance consideration. App Math Comput 217:3321–3329

Yang SJ (2012) Single-machine scheduling problems simultaneously with deterioration and learning effects under deteriorating multi-maintenance activities consideration. Comput Ind Eng 62:271–275

Yang SJ, Yang DL (2010) Minimizing the total completion time in single-machine scheduling with ageing/deteriorating effects and deteriorating maintenance activities. Comput Math Appl 60:2161–2169

# Chapter 18
# Scheduling with Rate-Modifying Activities and Enhanced Effects

In this chapter, we study enhanced models which combine various effects that determine the actual processing times of the jobs discussed so far, including

- Positional and time-dependent effects;
- Learning and deterioration effects;
- Rate-modifying activities.

We develop a general model and solve the problems of minimizing the makespan and the total completion time by reducing them to a series of linear assignment problems with a product matrix.

In all problems that we consider, we assume that the jobs of set $N = \{1, 2, \ldots, n\}$ have to be processed on a single machine. At time zero, all jobs are available and the machine is assumed to be in a perfect processing state, in which case the normal processing time of job $j \in N$ is equal to $p_j$. The processing conditions might change, with respect to both the elapsed time and the number of processed jobs. As outlined in Sect. 12.4, the decision-maker is presented with a list (RMP$^{[1]}$, RMP$^{[2]}$, ..., RMP$^{[K]}$) of $K \geq 1$ possible rate-modifying activities. The decision-maker may decide which of the listed RMPs to insert into a schedule and in which order.

Recall that in Sect. 8.1.3 we have reviewed models in which a linear time-dependent effect is combined with a positional effect, so that the actual processing time of a job $j \in N$ sequenced in position $r$ and starting at time $\tau \geq 0$ of a schedule is given by

$$p_j(\tau; r) = \big(p_j + a\tau\big)g(r), \tag{18.1}$$

where $a$ represents a rate that is common for all jobs $j \in N$ and is either positive (for a deterioration effect) or negative (for a learning effect). Array $g(r)$, $1 \leq r \leq n$, is a sequence of positional factors that defines an arbitrary positional effect. In general, the sequence $g(r)$, $1 \leq r \leq n$, need not be monotone; however, if it is non-decreasing (non-increasing), then it represents a positional deterioration (learning) effect.

V.A. Strusevich and K. Rustogi, *Scheduling with Times-Changing Effects and Rate-Modifying Activities*, International Series in Operations Research & Management Science 243, DOI 10.1007/978-3-319-39574-6_18

For the objective functions under consideration, it is proved in Theorem 8.11 that problem $1\left|p_j(\tau;r) = \left(p_j + a\tau\right)g(r)\right|\Phi$ with $\Phi \in \left\{C_{\max}, \sum C_j\right\}$ under no assumption on the sign of $a$ and on the monotonicity of array $g(r)$, $1 \le r \le n$, can be reduced to minimizing a generic objective function (8.22), so that the corresponding optimal schedule can be found by Algorithm Match in $O(n \log n)$ time.

In this chapter, we extend the results of Theorem 8.11 by studying the effect that combines the introduction of rate-modifying activities and the time-changing effects of the form (18.1). Such a model allows handling many quite general situations, in particular simultaneous learning and deterioration effects of both types (time-dependent and positional).

So far in this book, we have mainly considered problems in which an RMP is treated as a maintenance period and is used to compensate deterioration effects. However, in this chapter, an RMP is allowed to have any arbitrary effect on the machine conditions. It can still be seen as a maintenance period, which is aimed at restoring the machine conditions to a better state, so that the processing times become smaller. On the other hand, we now allow an RMP to be an activity in which, e.g., a machine/operator is replaced, so that all learning advantages are lost and the actual processing times of jobs become larger. Another form of an RMP may allow the learning rate of the machine to be further enhanced after running the RMP.

An illustration of a schedule with different types of RMPs is provided in Example 12.2, in which we combine learning and deterioration effects for a pure positional model. In the same example, we also consider a situation in which the positional factors $g^{[x]}(r)$ are found to be dependent on the number of jobs in previous groups. In this chapter, we extend such a model by combining it with a time-dependent effect. As a result, in our main model the actual processing time of a job is seen as affected by the following factors:

- the group of a job is assigned to;
- the position of the job within its group;
- the number of jobs scheduled in each of the previous groups;
- the time elapsed before processing the job within its group;
- the time elapsed in each of the previous groups.

## 18.1 Enhanced Model Description

In order to solve the described problems, we adapt Procedure RMP1 from Sect. 12.4. Assume that outcomes (A1) and (A2) are known and a certain outcome of Decision (B1) is determined. Consider a schedule $S_{B1}(k)$, in which $k-1$ RMPs are chosen from the available $K$ options, so that the jobs are divided into $k$, $1 \le k \le K+1$ groups. Assume that each group contains a total of $n^{[x]}$ jobs, so that $\pi^{[x]} = \left(\pi^{[x]}(1), \pi^{[x]}(2), \ldots, \pi^{[x]}\left(n^{[x]}\right)\right)$, $1 \le x \le k$, where $\sum_{x=1}^{k} n^{[x]} = n$. Depending on which RMPs are chosen and the order in which theyare performed, the actual processing time of a job $j = \pi^{[x]}(r)$, scheduled in position $r$, $1 \le r \le n^{[x]}$, of the $x$th group, $1 \le x \le k$, is given by

$$p_j^{[x]}(\tau; r) = \left(p_{\pi^{[x]}(r)} + a_1^{[x]} F_1 + a_2^{[x]} F_2 + \cdots + a_{x-1}^{[x]} F_{x-1} + a_x^{[x]} F_{(x,r-1)}\right) g^{[x]}(r),$$
$$1 \le r \le n^{[x]},\ 1 \le x \le k, \tag{18.2}$$

where $F_v$ denotes the total processing time of all jobs scheduled in a group $v$, $1 \le v \le x - 1$, while $F_{(x,r)}$ represents the total duration of the first $r$ jobs in a group $x$. For completeness, it is assumed that $F_0 = F_{(x,0)} := 0$. The terms $g^{[x]}(r)$ represent positive group-dependent job-independent positional factors, which do not have to be monotone within a particular group. The terms $a_1^{[x]}, a_2^{[x]}, \ldots, a_x^{[x]}$ are real numbers and represent the group-dependent rates associated with time-dependent effects. Notice that some of the rates $a_1^{[x]}, a_2^{[x]}, \ldots, a_x^{[x]}$ may be negative, which corresponds to a learning effect. Without going into technical details, in what follows we assume that the rates $a_1^{[x]}, a_2^{[x]}, \ldots, a_x^{[x]}$ are such that the processing times remain positive (see, e.g., Sect. 8.1.3), where the required conditions are explicitly written out for a less general combined effect.

For a group $x$, $1 \le x \le k$, a coefficient $a_v^{[x]}$ reflects the influence of the total duration $F_v$ of jobs in group $v$, $1 \le v \le x - 1$, on the actual processing time of any job in group $x$. Notice that such an influence can be different for different groups; i.e., for the same value of $v$ the rates, $a_v^{[x']}$ and $a_v^{[x'']}$ do not have to be equal for $x' \ne x''$. For a job $j$ scheduled in position $r$ of group $x$, the rates $a_1^{[x]}, a_2^{[x]}, \ldots, a_{x-1}^{[x]}$ determine how the length of each previous group affects the job's processing time, whereas $a_x^{[x]}$ determines how the processing time of the job is affected by the time elapsed since the opening of the $x$th group till job $j$ starts. The superscript $[x]$ associated with these rates stresses that the rates are group-dependent and can assume different values and signs depending on the group $x$, $1 \le x \le k$, a job is scheduled in. For example, to determine the actual processing time of a job scheduled in the third group, the required rates are $a_1^{[3]}, a_2^{[3]}$, and $a_3^{[3]}$, whereas if a job is placed in the fourth group, the required rates are $a_1^{[4]}, a_2^{[4]}, a_3^{[4]}$, and $a_4^{[4]}$.

If an RMP$^{[y]}$, $1 \le y \le K$, is inserted into a schedule, its duration, which we denote by $\Delta^{[y]}(\tau; \varpi)$, depends on time $\tau$ elapsed and the number of jobs $\varpi$ processed, relative to a certain reference point. The nature of such a reference point is content-dependent, and it will become clear from the explanations to follow. Provided that a reference point is identified, the general formula that defines the duration of RMP$^{[y]}$ is given by

$$\Delta^{[y]}(\tau; \varpi) = \left(\zeta^{[y]}\tau + \eta^{[y]}\right) h^{[y]}(\varpi), \tag{18.3}$$

where, as in (12.1), the coefficients $\zeta^{[y]}$ and $\eta^{[y]} > 0$ are known, while $h^{[y]}(\varpi)$ represents a positional factor.

Notice that (18.3) is a generalization of the definition of the RMP duration $\Delta^{[y]}(\tau)$ given in (12.1). First, the Formula (18.3) additionally allows a positional factor, so that the duration of the RMP is dependent not only on the start time of the RMP, but also on the position of the RMP in the processing sequence. Second, for $\Delta^{[y]}(\tau)$ defined by (12.1) $\tau$ is the duration of the group that immediately precedes RMP$^{[y]}$, so that the reference point for measuring $\tau$ is the start time of that group and it is not affected by any of the earlier scheduled groups; on the other hand, in (18.3) the reference point for computing $\Delta^{[y]}(\tau;\varpi)$ for RMP$^{[y]}$ may go back to the starting point of any earlier group, which can be located several groups away from RMP$^{[y]}$.

It is always assumed that during an RMP the system undergoes neither learning nor deterioration. Thus, the actual processing times of the jobs and the RMPs are independent of the duration of the previously scheduled RMPs.

For a schedule $S_{\mathrm{B}1}(k)$, we will explain how to compute $T_x$, the duration of an RMP after the $x$th group, by (18.3), so that $T_x$ is expressed as a function of the values $F_v$, $1 \le v \le x$.

Let us start this with a small-sized example. Consider an instance in which a machine is a device controlled by an operator. The machine undergoes deterioration and learning simultaneously. Two RMPs are to be included into the processing sequence, the first being used as a training period for the operator, while the second being a maintenance activity which repairs the device. As a result, three groups are created, with the number of jobs in each being $n^{[1]}$, $n^{[2]}$ and, $n^{[3]}$, and the duration of these groups being $F_1$, $F_2$, and $F_3$, respectively.

The first RMP starts when the first group is completed, so that time zero, the beginning of the first group, is taken as the reference point for applying (18.3). Thus, we have

$$T_1 = \left[\zeta^{[1]}F_1 + \eta^{[1]}\right]h^{[1]}\left(n^{[1]}\right) = \zeta^{[1]}h^{[1]}\left(n^{[1]}\right)F_1 + \eta^{[1]}h^{[1]}\left(n^{[1]}\right).$$

The second RMP starts when the second group is completed. The device undergoes continuous deterioration during the first two groups, since the first RMP does not affect the state of the device. In this case, again time zero, the beginning of the first group, is taken as the reference point for applying (18.3). The time elapsed until the second RMP starts is equal to $F_1 + F_2$, while the number of jobs scheduled until that point is equal to $n^{[1]} + n^{[2]}$. According have

$$\begin{aligned} T_2 &= \left[\zeta^{[2]}(F_1 + F_2) + \eta^{[2]}\right]h^{[2]}\left(n^{[1]} + n^{[2]}\right) \\ &= \zeta^{[2]}h^{[2]}\left(n^{[1]} + n^{[2]}\right)(F_1 + F_2) + \eta^{[2]}h^{[2]}\left(n^{[1]} + n^{[2]}\right). \end{aligned}$$

For the first RMP, define

$$\zeta_1^{[1]} := \zeta^{[1]}h^{[1]}\left(n^{[1]}\right),\ \widehat{\eta}^{[1]} := \eta^{[1]}h^{[1]}\left(n^{[1]}\right),$$

and for the second RMP, define

$$\zeta_1^{[2]} = \zeta_2^{[2]} := \zeta^{[2]}h^{[2]}\left(n^{[1]} + n^{[2]}\right),\ \widehat{\eta}^{[2]} := \eta^{[2]}h^{[2]}\left(n^{[1]} + n^{[2]}\right),$$

so that we can write

$$T_1 = \zeta_1^{[1]} F_1 + \widehat{\eta}^{[1]};$$
$$T_2 = \zeta_1^{[2]} F_1 + \zeta_2^{[2]} F_2 + \widehat{\eta}^{[2]}.$$

Extending this example, we can write out an expression for $T_x$, the duration of the $x$th RMP, $1 \le x \le k-1$, in schedule $S_{B1}(k)$, provided that the nature of the RMP delivers an appropriate reference point for applying (18.3). The suggested expression can generically be written as

$$T_x = \zeta_1^{[x]} F_1 + \zeta_2^{[x]} F_2 + \cdots + \zeta_x^{[x]} F_x + \widehat{\eta}^{[x]},\ 1 \le x \le k-1, \tag{18.4}$$

where the values $\zeta_1^{[x]}, \zeta_2^{[x]}, \dots, \zeta_x^{[x]}$ determine how the length of each previous group affects the duration of the RMP scheduled after the $x$th group, and $\widehat{\eta}^{[x]} > 0$ is a constant, $1 \le x \le k-1$. Typically, $\widehat{\eta}^{[x]}$ is function of $\eta^{[x]}$ and $h^{[x]}$, where the factor $h^{[x]}$ depends on all or some $n^{[v]}$, $1 \le v \le x$. The coefficients $\zeta_1^{[x]}, \zeta_2^{[x]}, \dots, \zeta_x^{[x]}$ can be seen as analogues of the rates $a_1^{[x]}, a_2^{[x]}, \dots, a_{x-1}^{[x]}$, defined in (18.2). Similarly to the rates, the values $\zeta_v^{[x']}$ and $\zeta_v^{[x'']}$ can be different for different groups $x'$ and $x''$. Notice that the values $\zeta_1^{[x]}, \zeta_2^{[x]}, \dots, \zeta_x^{[x]}$ can assume any sign as long as it is ensured that the duration of the RMPs is positive. They allow us to incorporate RMPs of a different nature in a schedule.

For an objective function $\Phi \in \left\{C_{\max}, \sum C_j\right\}$, we extend the standard three-field notation to denote the problem of single machine processing subject to the described effects by $1|\text{Effect (18.2)}, RMP(K), \Delta^{[x]}(\tau; \varpi)|\Phi$. Here, in the middle field we write "Effect (18.2)" to stress that the processing times are subject to a combined effect (18.2), we write "$RMP(K)$" to indicate that a total of $K$ rate-modifying activities are available, and we write "$\Delta^{[x]}(\tau; \varpi)$ " to indicate that the duration of RMP$^{[x]}$ is found in accordance with (18.3).

For problem $1|\text{Effect (18.2)}, RMP(K), \Delta^{[x]}(\tau; \varpi)|\Phi$, in accordance with Procedure RMP1 fix outcomes (A1) and (A2), determine a certain outcome of Decision (B1) and denote the resulting auxiliary problem as $1|\text{Effect (18.2)}, RMP(k-1), \Delta^{[x]}(\tau; \varpi)|\Phi$, where $\Phi \in \left\{C_{\max}, \sum C_j\right\}$. The required input for problem $1|\text{Effect (18.2)}, RMP(k-1), \Delta^{[x]}(\tau; \varpi)|\Phi$ includes:

- the processing times $p_1, p_2, \dots, p_n$, of jobs;
- the rates $a_1^{[x]}, a_2^{[x]}, \dots, a_x^{[x]}$, for each $x$, $1 \le x \le k$;
- the positional factors $g^{[x]}(r)$, $1 \le r \le n$, for each $x$, $1 \le x \le k$;
- the values $\zeta_1^{[x]}, \zeta_2^{[x]}, \dots, \zeta_x^{[x]}, \widehat{\eta}^{[2]}$ related to the duration of the $x$th RMP, for each $x$, $1 \le x \le k-1$.

The introduced model allows us to handle a wide range of practical problems, which have not been studied in the scheduling literature until very recently. Complicated as it looks, the model essentially incorporates and generalizes almost every job-independent effect known in scheduling with changing processing times. It is flexible enough to serve as a model of many plausible scenarios that may be found in practice, e.g., in the area manufacturing or shop floor production planning. Below we give an illustration of a possible application.

*Example 18.1* A human operator (Operator 1) uses a tool to process $n$ jobs. During the processing of the jobs, two RMPs will be included in the schedule. It is known that the first RMP, associated with the coefficients $\zeta'$ and $\eta'$, is actually a maintenance period which restores the machine to its original condition. However, the deterioration rate of the machine becomes greater after the maintenance period, since the original spare parts are not used. This RMP also provides the operator with sufficient rest, so that after the first RMP the operator is as fresh as he/she was at the beginning of the schedule. The duration of this RMP is dependent on its start time, so that for $x = 1$, we have $\zeta_1^{[1]} = \zeta'$ and $\widehat{\eta}^{[1]} = \eta'$. The second RMP takes a constant time $\eta''$, i.e., $\zeta_1^{[2]} = \zeta_2^{[2]} = 0$ and $\widehat{\eta}^{[2]} = \eta''$, but does not repair the machine at all. Instead, a new operator (Operator 2) is brought in. Below, we distinguish between the learning and deterioration parameters for the machine and those for the operator(s) by using the subscript "$m$" for the machine and "$w$" for the operator(s)/worker(s), respectively.

In a feasible schedule, the jobs will be split into $k = 3$ groups. The machine suffers from a linear time-dependent deterioration effect, the deterioration rate being equal to $d'_m > 0$ before the first RMP (the first group), and equal to $d''_m > d'_m > 0$ after the first RMP (for the second and the third groups). Additionally, the machine is also affected by a positional exponential deterioration effect of the form $\left(d'_m + 1\right)^{r-1}$ before the first RMP and of the form $\left(d''_m + 1\right)^{r-1}$ after that RMP. The operators are also subject to time-dependent effects; the deterioration and learning rates for Operator 1 are $d'_w > 0$ and $l'_w < 0$, respectively, and those for Operator 2 are $d''_w > 0$ and $l''_w < 0$, respectively. It is known that in addition to the skills gained while processing the jobs, Operator 2 also passively learns with a rate $l'''_w < 0$, while she observes Operator 1 processing the jobs in groups 1 and 2. Last, the performance of the workers is also affected by a polynomial positional effect and is quantified by $r^\delta$, where the appropriate value of $\delta$ is one of $d'_w$, $l'_w$, $d''_w$ and, $l''_w$, depending on the scenario. There is no positional effect associated with the passive learning effect of Operator 2.

For the described example, the parameters of our model (18.2) can be set as shown in Table 18.1.

Notice that our model allows us to assume that during an RMP, if the operator is not replaced, they do not lose their skills, which have been improved due to learning in the earlier groups of the schedule. Similarly, if during an RMP a machine is not fully repaired, our model is capable of handling the resulting situation, in which the deterioration effect from the group before the RMP must be carried forward to the next group. The same phenomenon is also observed in the passive learning effect of Operator 2, in which the skills gained during groups 1 and 2 are carried forward to group 3. The time-dependent effects can be captured by the group-dependent parameters $a_v^{[x]}$, $1 \le v \le x - 1$, $1 \le x \le k$. Thus, to model the situation in the given example, we define $a_1^{[2]} := l'_w$ (implying that during $F_1$ time units Operator 1 has gained an experience), $a_1^{[3]} := l'''_w$ (implying that Operator 2 has gained a passive learning experience during $F_1$ time units) and $a_2^{[3]} := d''_m + l'''_w$ (implying that during $F_2$ time units the machine has deteriorated and Operator 2 has gained a passive learning experience). The associated positional effects can be easily captured by

**Table 18.1** Parameters for Example 18.1

| Group | Parameter | Value |
|---|---|---|
| 1 | $a_1^{[1]}$ | $d'_m + d'_w + l'_w$ |
| | $g^{[1]}(r)$ | $\left(d'_m + 1\right)^{r-1} r^{d'_w + l'_w}, 1 \le r \le n^{[1]}$ |
| 2 | $a_1^{[2]}$ | $l'_w$ |
| | $a_2^{[2]}$ | $d''_m + d'_w + l'_w$ |
| | $g^{[2]}(r)$ | $\left(d''_m + 1\right)^{r-1} \left(n^{[1]} + r\right)^{l'_w} r^{d'_w}, 1 \le r \le n^{[2]}$ |
| 3 | $a_1^{[3]}$ | $l'''_w$ |
| | $a_2^{[3]}$ | $d''_m + l'''_w$ |
| | $a_3^{[3]}$ | $d''_m + d''_w + l''_w$ |
| | $g^{[3]}(r)$ | $\left(d''_m + 1\right)^{n^{[2]}+r-1} r^{d''_w + l''_w}, 1 \le r \le n^{[3]}$ |

adjusting the relative position of a job in the relevant group, as illustrated in Example 12.1.

As demonstrated in the above example, with an appropriate use of the parameters $a_v^{[x]}, \zeta_v^{[x]}, 1 \le v \le x, g^{[x]}(r), 1 \le r \le n$, and $\widehat{\eta}^{[x]}$, for each $x$, $1 \le x \le k$, our model as defined by (18.2) and (18.4) can be used to represent a wide range of practical situations.

In what follows, we show that problem $1|\text{Effect (18.2)}, RMP(K), \Delta^{[x]}(\tau; \varpi)|\Phi$, for $\Phi \in \left\{C_{\max}, \sum C_j\right\}$ reduces to a series of linear assignment problems with a product matrix and can be solved in polynomial time.

For all problems considered in this chapter, we are required to perform full enumeration to determine the number of jobs to schedule in a particular group, so that the resulting running times contain at least a multiple $n^K$.

## 18.2 Computing the Completion Times

For problem $1|\text{Effect (18.2)}, RMP(K), \Delta^{[x]}(\tau; \varpi)|\Phi$, in accordance with Procedure RMP1 fix outcomes (A1) and (A2), and for a particular outcome of Decision (B1), introduce a schedule $S_{B1}(k)$ for an auxiliary problem $1|\text{Effect (18.2)}, RMP(k-1), \Delta^{[x]}(\tau; \varpi)|\Phi$, associated with certain outcomes of Decisions (B2) and (B3). In schedule $S_{B1}(k)$, the jobs are organized into groups $N^{[x]}$, $1 \le x \le k$, and each group $N^{[x]}$ contains $n^{[x]}$ jobs, where $\sum_{x=1}^{k} n^{[x]} = n$. The jobs in $N^{[x]}$ are sequenced
in accordance with a permutation $\pi^{[x]} = \left(\pi^{[x]}(1), \pi^{[x]}(2), \ldots, \pi^{[x]}\left(n^{[x]}\right)\right)$, $1 \le x \le k$.

We now derive an expression for the total time it takes to process all jobs in a group $x$, $1 \le x \le k$, in a schedule $S_{B1}(k)$.

Recall that throughout this book, it is assumed that an empty product is equal to one, while an empty sum is equal to zero, i.e., $\prod_{i=j}^{r}(\cdot) = 1$ and $\sum_{i=j}^{r}(\cdot) = 0$, for $j > r$.

**Lemma 18.1** *Given a group $x$, $1 \le x \le k$, and job $j = \pi^{[x]}(r)$ sequenced in the $r$th position, $1 \le r \le n^{[x]}$, of the group, the completion time $F_{(x,r)}$ of the job with respect to the start time of the group is given by*

$$F_{(x,r)} = \sum_{u=1}^{r} \big(A^{[x]} + p_{\pi^{[x]}(u)}\big) B^{[x]}(u,r), \tag{18.5}$$

*where $A^{[x]} := \sum_{v=1}^{x-1} a_v^{[x]} F_v$, and*

$$B^{[x]}(u,r) := g^{[x]}(u) \prod_{i=u+1}^{r} \big(1 + a_x^{[x]} g^{[x]}(i)\big),\ 1 \le u \le r. \tag{18.6}$$

*Proof* We prove this lemma by induction. For job $j = \pi^{[x]}(r)$, the value $F_{(x,r)}$ for $r = 1$ is given in accordance with (18.2) by

$$\begin{aligned} p_j^{[x]}(1) &= \Big(p_{\pi^{[x]}(1)} + a_1^{[x]} F_1 + a_2^{[x]} F_2 + \cdots + a_{x-1}^{[x]} F_{x-1}\Big) g^{[x]}(1) \\ &= \big(p_{\pi^{[x]}(1)} + A^{[x]}\big) g^{[x]}(1) = \big(p_{\pi^{[x]}(1)} + A^{[x]}\big) B^{[x]}(1,1), \end{aligned}$$

which corresponds to the right-hand side of (18.5).

Assume that for $r$, $1 < r \le n^{[x]}$, the equality

$$F_{(x,r-1)} = \sum_{u=1}^{r-1} \big(A^{[x]} + p_{\pi^{[x]}(u)}\big) B^{[x]}(u,r-1), \tag{18.7}$$

holds. We know that

$$F_{(x,r)} = F_{(x,r-1)} + p_j^{[x]}(r).$$

Substituting (18.2), we get

$$\begin{aligned} F_{(x,r)} &= F_{(x,r-1)} + \big(p_{\pi^{[x]}(r)} + A^{[x]} + a_x^{[x]} F_{(x,r-1)}\big) g^{[x]}(r) \\ &= \big(1 + a_x^{[x]} g^{[x]}(r)\big) F_{(x,r-1)} + \big(p_{\pi^{[x]}(r)} + A^{[x]}\big) g^{[x]}(r). \end{aligned}$$

Substituting the value of $F_{(x,r-1)}$ from (18.7), we obtain

$$F_{(x,r)} = \big(1 + a_x^{[x]} g^{[x]}(r)\big) \sum_{u=1}^{r-1} \big(A^{[x]} + p_{\pi^{[x]}(u)}\big) B^{[x]}(u,r-1) + \big(A^{[x]} + p_{\pi^{[x]}(r)}\big) g^{[x]}(r).$$

It can be easily verified that $\left(1 + a_x^{[x]} g^{[x]}(r)\right) B^{[x]}(u, r-1) = B^{[x]}(u, r)$ and $g^{[x]}(r) = B^{[x]}(r, r)$, so that we obtain

$$\begin{aligned} F_{(x,r)} &= \sum_{u=1}^{r-1} \left(A^{[x]} + p_{\pi^{[x]}(u)}\right) B^{[x]}(u, r) + \left(A^{[x]} + p_{\pi^{[x]}(r)}\right) B^{[x]}(r, r) \\ &= \sum_{u=1}^{r} \left(A^{[x]} + p_{\pi^{[x]}(u)}\right) B^{[x]}(u, r), \end{aligned}$$

which proves the lemma. □

Due to Lemma 18.1, the total processing time of a group $x$, $1 \le x \le k$, can be written as

$$\begin{aligned} F_x = F_{\left(x, n^{[x]}\right)} &= \sum_{r=1}^{n^{[x]}} \left(A^{[x]} + p_{\pi^{[x]}(r)}\right) B^{[x]}\left(r, n^{[x]}\right) \\ &= \sum_{r=1}^{n^{[x]}} \left(\sum_{v=1}^{x-1} a_v^{[x]} F_v\right) B^{[x]}\left(r, n^{[x]}\right) + \sum_{r=1}^{n^{[x]}} p_{\pi^{[x]}(r)} B^{[x]}\left(r, n^{[x]}\right) \\ &= \sum_{v=1}^{x-1} \left(a_v^{[x]} \sum_{r=1}^{n^{[x]}} B^{[x]}\left(r, n^{[x]}\right)\right) F_v + \sum_{r=1}^{n^{[x]}} p_{\pi^{[x]}(r)} B^{[x]}\left(r, n^{[x]}\right). \end{aligned}$$

For convenience, denote

$$D^{[x]} := \sum_{r=1}^{n^{[x]}} p_{\pi^{[x]}(r)} B^{[x]}\left(r, n^{[x]}\right),\ 1 \le x \le k, \tag{18.8}$$

and

$$b_v^{[x]} := a_v^{[x]} \sum_{r=1}^{n^{[x]}} B^{[x]}\left(r, n^{[x]}\right),\ 1 \le v \le x-1,\ 1 \le x \le k. \tag{18.9}$$

Then $F_x$ can be rewritten as

$$F_x = b_1^{[x]} F_1 + b_2^{[x]} F_2 + \cdots + b_{x-1}^{[x]} F_{x-1} + D^{[x]}. \tag{18.10}$$

**Lemma 18.2** *The total processing time of a group $x$, $1 \le x \le k$, is given by*

$$F_x = \sum_{v=1}^{x} E^{[v,x]} D^{[v]}, \tag{18.11}$$

*where $E^{[x,x]} := 1$ and for $v \le x - 1$*

$$E^{[v,x]} := \sum_{w=1}^{x-v} \sum_{v=v_0<v_1<\cdots<v_w=x} b_{v_0}^{[v_1]} b_{v_1}^{[v_2]} b_{v_2}^{[v_3]} \cdots b_{v_{w-1}}^{[v_w]}, \tag{18.12}$$

*where in (18.12) for each* $w$*,* $1 \le w \le x - v$*, the second summation is taken over all increasing sequences of* $w + 1$ *distinct integers* $(v_0, v_1, \ldots, v_w)$ *with* $v_0 = v$ *and* $v_w = x$*.*

*Proof* Please notice the zigzag pattern of the subscripts and superscripts in the right-hand side of (18.12), as outlined below.

$$\begin{tikzcd}
 & {[v_1]} \arrow[dr] & & {[v_2]} \arrow[dr] & & \cdots & & {[v_{w-1}]} \arrow[dr] & & {[v_w]} \\
v_0 \arrow[ur] & & v_1 \arrow[ur] & & v_2 \arrow[ur] & & \cdots \arrow[ur] & & v_{w-1} \arrow[ur] &
\end{tikzcd}$$

The proof of the lemma is by induction. For $x = 1$, it follows from (18.10) that $F_1 = D^{[1]}$. On the other hand, for $x = 1$, (18.11) reduces to $F_1 = E^{[1,1]} D^{[1]}$, and since $E^{[1,1]} = 1$ by definition, we also obtain $F_1 = D^{[1]}$.

Assume now that the lemma holds for all groups $1, 2, \ldots, x - 1$, and prove that it holds for group $x \le k$. Starting with (18.10), we write

$$F_x = \sum_{v=1}^{x-1} b_v^{[x]} F_v + D^{[x]},$$

and use the induction assumption to substitute (18.11) into the above expression to obtain

$$\begin{aligned}
F_x &= \sum_{v=1}^{x-1} b_v^{[x]} \sum_{y=1}^{v} E^{[y,v]} D^{[y]} + D^{[x]} = \sum_{v=1}^{x-1} \sum_{y=v}^{x-1} b_y^{[x]} E^{[v,y]} D^{[v]} + D^{[x]} \\
&= \sum_{v=1}^{x-1} \left( b_v^{[x]} E^{[v,v]} + \sum_{y=v+1}^{x-1} b_y^{[x]} E^{[v,y]} \right) D^{[v]} + D^{[x]}.
\end{aligned}$$

Further, using the induction assumption, replace $E^{[v,v]}$ by 1 and substitute $E^{[v,y]}$, $v < y$, in accordance with (18.12). We deduce

$$\begin{aligned}
F_x &= \sum_{v=1}^{x-1} \left( b_v^{[x]} + \sum_{y=v+1}^{x-1} b_y^{[x]} \left( \sum_{w=1}^{y-v} \sum_{v=v_0<v_1<\cdots<v_w=y} b_{v_0}^{[v_1]} b_{v_1}^{[v_2]} b_{v_2}^{[v_3]} \cdots b_{v_{w-1}}^{[v_w]} \right) \right) D^{[v]} + D^{[x]} \\
&= \sum_{v=1}^{x-1} \left( b_v^{[x]} + \sum_{y=v+1}^{x-1} \sum_{w=1}^{y-v} \sum_{v=v_0<v_1<\cdots<v_w=y} b_{v_0}^{[v_1]} b_{v_1}^{[v_2]} b_{v_2}^{[v_3]} \cdots b_{v_{w-1}}^{[v_w]} b_y^{[x]} \right) D^{[v]} + D^{[x]} \\
&= \sum_{v=1}^{x-1} \left( b_v^{[x]} + \sum_{y=v+1}^{x-1} \sum_{w=1}^{y-v} \sum_{v=v_0<v_1<\cdots<v_w=y} b_{v_0}^{[v_1]} b_{v_1}^{[v_2]} b_{v_2}^{[v_3]} \cdots b_{v_{w-1}}^{[v_w]} b_{v_w}^{[x]} \right) D^{[v]} + D^{[x]}.
\end{aligned}$$

Observe that for a fixed $w$ the equality

$$\sum_{y=v+1}^{x-1} \sum_{v=v_0<v_1<\cdots<v_w=y} b_{v_0}^{[v_1]} b_{v_1}^{[v_2]} b_{v_2}^{[v_3]} \cdots b_{v_{w-1}}^{[v_w]} b_{v_w}^{[x]} = \sum_{v=v_0<v_1<\cdots<v_w\le x-1} b_{v_0}^{[v_1]} b_{v_1}^{[v_2]} b_{v_2}^{[v_3]} \cdots b_{v_{w-1}}^{[v_w]} b_{v_w}^{[x]}$$

holds, where the summation in the right-hand side is taken over all increasing sequences of $w+1$ distinct integers $(v_0, v_1, \ldots, v_w)$ with $v_0 = v$ and $v_w \le x-1$.

Notice that for $y = v+1$ and for $y = x-1$, we have that $w = 1$ and $w = x - v - 1$, respectively, i.e., $1 \le w \le x - v - 1$. This means that

$$F_x = \sum_{v=1}^{x-1} \left( b_v^{[x]} + \sum_{w=1}^{x-v-1} \sum_{v=v_0<v_1<\cdots<v_w\le x-1} b_{v_0}^{[v_1]} b_{v_1}^{[v_2]} b_{v_2}^{[v_3]} \cdots b_{v_{w-1}}^{[v_w]} b_{v_w}^{[x]} \right) D^{[v]} + D^{[x]}.$$

Replacing $w$ with $w-1$, rewrite

$$F_x = \sum_{v=1}^{x-1} \left( b_v^{[x]} + \sum_{w=2}^{x-v} \sum_{v=v_0<v_1<\cdots<v_{w-1}\le x-1} b_{v_0}^{[v_1]} b_{v_1}^{[v_2]} b_{v_2}^{[v_3]} \cdots b_{v_{w-2}}^{[v_{w-1}]} b_{v_{w-1}}^{[x]} \right) D^{[v]} + D^{[x]}.$$

It can be easily verified that

$$b_v^{[x]} = \sum_{w=1}^{1} \sum_{v=v_0<v_1<\cdots<v_{w-1}<v_w=x} b_{v_0}^{[v_1]} b_{v_1}^{[v_2]} \cdots b_{v_{w-2}}^{[v_{w-1}]} b_{v_{w-1}}^{[v_w]},\ 1 \le v \le x-1,$$

so that $F_x$, $1 \le x \le k$, can be rewritten as

$$\begin{aligned} F_x &= \sum_{v=1}^{x-1} \left( \sum_{w=1}^{x-v} \sum_{v=v_0<v_1<\cdots<v_{w-1}<v_w=x} b_{v_0}^{[v_1]} b_{v_1}^{[v_2]} b_{v_2}^{[v_3]} \cdots b_{v_{w-2}}^{[v_{w-1}]} b_{v_{w-1}}^{[v_w]} \right) D^{[v]} + D^{[x]} \\ &= \sum_{v=1}^{x-1} E^{[v,x]} D^{[v]} + E^{[x,x]} D^{[x]} = \sum_{v=1}^{x} E^{[v,x]} D^{[v]}, \end{aligned}$$

which proves the lemma. □

The expressions in Lemma 18.2 look heavy; below we illustrate them for small values of $x$, e.g., $x \le 4$. Recall that for $x = 1$, we have $F_1 = D^{[1]}$. For $x = 2$, the formula (18.10) gives $F_2 = b_1^{[2]} F_1 + D^{[2]} = b_1^{[2]} D^{[1]} + D^{[2]}$, which complies with (18.11) for $x = 2$, since $E^{[1,2]} = b_1^{[2]}$ and $E^{[2,2]} = 1$. Similarly, for $x = 3$, the formula (18.10) reduces to

$$\begin{aligned} F_3 &= b_1^{[3]} F_1 + b_2^{[3]} F_2 + D^{[3]} = b_1^{[3]} D^{[1]} + b_2^{[3]} \left( b_1^{[2]} D^{[1]} + D^{[2]} \right) + D^{[3]} \\ &= \left( b_1^{[3]} + b_1^{[2]} b_2^{[3]} \right) D^{[1]} + b_2^{[3]} D^{[2]} + D^{[3]}. \end{aligned}$$

It can be easily verified that this complies with (18.11) for $x = 3$. For $x = 4$, using (18.10) we obtain

$$\begin{aligned}
F_4 &= b_1^{[4]}F_1 + b_2^{[4]}F_2 + b_3^{[4]}F_3 + D^{[4]} \\
&= b_1^{[4]}D^{[1]} + b_2^{[4]}\left(b_1^{[2]}D^{[1]} + D^{[2]}\right) + b_3^{[4]}\left(\left(b_1^{[3]} + b_1^{[2]}b_2^{[3]}\right)D^{[1]} + b_2^{[3]}D^{[2]} + D^{[3]}\right) + D^{[4]} \\
&= \left(b_1^{[4]} + b_1^{[2]}b_2^{[4]} + b_1^{[3]}b_3^{[4]} + b_1^{[2]}b_2^{[3]}b_3^{[4]}\right)D^{[1]} + \left(b_2^{[4]} + b_2^{[3]}b_3^{[4]}\right)D^{[2]} + b_3^{[4]}D^{[3]} + D^{[4]}.
\end{aligned}$$

On the other hand, using (18.11) we obtain

$$\begin{aligned}
F_4 &= E^{[1,4]}D^{[1]} + E^{[2,4]}D^{[2]} + E^{[3,4]}D^{[3]} + E^{[4,4]}D^{[4]} \\
&= \left(\sum_{w=1}^{3}\sum_{1=v_0<v_1<\cdots<v_w=4} b_{v_0}^{[v_1]}b_{v_1}^{[v_2]}\cdots b_{v_{w-1}}^{[v_w]}\right)D^{[1]} \\
&\quad+\left(\sum_{w=1}^{2}\sum_{2=v_0<v_1<\cdots<v_w=4} b_{v_0}^{[v_1]}b_{v_1}^{[v_2]}\cdots b_{v_{w-1}}^{[v_w]}\right)D^{[2]} \\
&\quad+\left(\sum_{w=1}^{1}\sum_{3=v_0<v_1<\cdots<v_w=4} b_{v_0}^{[v_1]}b_{v_1}^{[v_2]}\cdots b_{v_{w-1}}^{[v_w]}\right)D^{[3]} + D^{[4]} \\
&= \left(\sum_{1=v_0<v_1=4} b_{v_0}^{[v_1]} + \sum_{1=v_0<v_1<v_2=4} b_{v_0}^{[v_1]}b_{v_1}^{[v_2]} + \sum_{1=v_0<v_1<v_2<v_3=4} b_{v_0}^{[v_1]}b_{v_1}^{[v_2]}b_{v_2}^{[v_3]}\right)D^{[1]} + \\
&\quad\left(\sum_{2=v_0<v_1=4} b_{v_0}^{[v_1]} + \sum_{2=v_0<v_1<v_2=4} b_{v_0}^{[v_1]}b_{v_1}^{[v_2]}\right)D^{[2]} + \left(\sum_{3=v_0<v_1=4} b_{v_0}^{[v_1]}\right)D^{[3]} + D^{[4]} \\
&= \left(b_1^{[4]} + b_1^{[2]}b_2^{[4]} + b_1^{[3]}b_3^{[4]} + b_1^{[2]}b_2^{[3]}b_3^{[4]}\right)D^{[1]} + \left(b_2^{[4]} + b_2^{[3]}b_3^{[4]}\right)D^{[2]} + b_3^{[4]}D^{[3]} + D^{[4]}.
\end{aligned}$$

Notice that the number of terms contained in the expression $\sum_{v=v_0<v_1<\cdots<v_w=x} b_{v_0}^{[v_1]}b_{v_1}^{[v_2]}b_{v_2}^{[v_3]}\cdots b_{v_{w-1}}^{[v_w]}$, involved in the right-hand side of (18.12), is $\binom{x-v-1}{w-1}$. Thus, to determine all values of $E^{[v,x]}$, $1 \le v \le x-1$, $2 \le x \le k$, the total number of products to be computed is $\sum_{x=2}^{k}\sum_{v=1}^{x-1}\sum_{w=1}^{x-v}\binom{x-v-1}{w-1} = 2^k - k - 1$.

Recall that the durations of the RMPs are given by (18.4). Including the time spent on rate-modifying activities, the completion time $C_{\pi^{[x]}(r)}$ of a job scheduled in position $r$, $1 \le r \le n^{[x]}$, of the $x$th group, $1 \le x \le k$, can be written as

$$\begin{aligned}
C_{\pi^{[x]}(r)} &= F_1 + T_1 + F_2 + T_2 + \cdots + F_{x-1} + T_{x-1} + F_{(x,r)} \\
&= \sum_{v=1}^{x-1}\left[1 + \sum_{w=v}^{x-1}\zeta_v^{[w]}\right]F_v + F_{(x,r)} + \sum_{v=1}^{x-1}\widehat{\eta}^{[v]}.
\end{aligned}$$

For convenience, denote

$$\xi^{[v,x-1]} := 1 + \sum_{w=v}^{x-1} \zeta_v^{[w]},\ 1 \le v \le x-1,\ \ 1 \le x \le k, \tag{18.13}$$

and rewrite

$$C_{\pi^{[x]}(r)} = \sum_{v=1}^{x-1} \xi^{[v,x-1]} F_v + F_{(x,r)} + \sum_{v=1}^{x-1} \widehat{\eta}^{[v]}. \tag{18.14}$$

Applying the results of Lemmas 18.1 and 18.2, the completion time can be written in terms of the original problem parameters.

## 18.3 Minimizing the Makespan

Let us now specifically consider an auxiliary problem $1|\text{Effect (18.2)}, RMP(k-1), \Delta^{[x]}(\tau; \varpi)\,|C_{\max}$. For a schedule $S_{\text{B1}}(k)$, denote the makespan by $C_{\max}(S_{\text{B1}}(k))$. If schedule $S_{\text{B1}}(k)$ is associated with a permutation $\pi = \left(\pi^{[1]}, \pi^{[2]}, \ldots, \pi^{[k]}\right)$, then $C_{\max}(S_{\text{B1}}(k))$ is equal to $C_{\pi^{[k]}\left(n^{[k]}\right)}$, the completion time of the last job in the last $k$th group. From (18.14), we have

$$C_{\max}(S_{\text{B1}}(k)) = C_{\pi^{[k]}\left(n^{[k]}\right)} = \sum_{x=1}^{k-1} \xi^{[x,k-1]} F_x + F_k + \Gamma(k),$$

where the constant term is defined as

$$\Gamma(k) = \sum_{x=1}^{k-1} \widehat{\eta}^{[x]}. \tag{18.15}$$

Using (18.11), rewrite

$$C_{\max}(S_{\text{B1}}(k)) = \sum_{x=1}^{k-1} \xi^{[x,k-1]} \left( \sum_{v=1}^{x} E^{[v,x]} D^{[v]} \right) + \sum_{v=1}^{k} E^{[v,k]} D^{[v]} + \Gamma(k).$$

Rearranging the terms, we get

$$C_{\max}(S_{\text{B1}}(k)) = \sum_{x=1}^{k} \left( \sum_{v=x}^{k-1} \xi^{[v,k-1]} E^{[x,v]} + E^{[x,k]} \right) D^{[x]} + \Gamma(k).$$

Substituting the value of $D^{[x]}$ from (18.8), we further derive

$$\begin{aligned} C_{\max}(S_{\mathrm{B1}}(k)) &= \sum_{x=1}^{k}\left(\sum_{v=x}^{k-1}\xi^{[v,k-1]}E^{[x,v]} + E^{[x,k]}\right)\sum_{r=1}^{n^{[x]}} p_{\pi^{[x]}(r)}B^{[x]}\left(r, n^{[x]}\right) + \Gamma(k) \\ &= \sum_{x=1}^{k}\sum_{r=1}^{n^{[x]}} p_{\pi^{[x]}(r)}B^{[x]}\left(r, n^{[x]}\right)\left(\sum_{v=x}^{k-1}\xi^{[v,k-1]}E^{[x,v]} + E^{[x,k]}\right) + \Gamma(k). \end{aligned}$$

The above can be represented as

$$C_{\max}(S_{\mathrm{B1}}(k)) = \sum_{x=1}^{k}\sum_{r=1}^{n^{[x]}} W^{[x]}(r)p_{\pi^{[x]}(r)} + \Gamma(k), \tag{18.16}$$

where

$$W^{[x]}(r) = B^{[x]}\left(r, n^{[x]}\right)\left(\sum_{v=x}^{k-1}\left(1 + \sum_{w=v}^{k-1}\zeta_v^{[w]}\right)E^{[x,v]} + E^{[x,k]}\right),\ 1 \le r \le n^{[x]},\ 1 \le x \le k, \tag{18.17}$$

is a job-independent weight, such that the product $W^{[x]}p_{\pi^{[x]}(r)}$ represents the contribution of job $j = \pi^{[x]}(r)$ scheduled in position $r$, $1 \le r \le n^{[x]}$, of group $x$, $1 \le x \le k$, to the objective function. The expression for computing the positional weights (18.17) can be written in terms of the original parameters by the use of (18.6), (18.9), and (18.12).

Let $T(W)$ denote the time required for computing the positional weights for an auxiliary problem $1|\text{Effect (18.2)}, RMP(k-1), \Delta^{[x]}(\tau; \varpi)|\Phi$ expressed as a function of $n$ and $k$.

Finding all values of $B^{[x]}\left(r, n^{[x]}\right)$, $1 \le r \le n^{[x]}$, $1 \le x \le k$, by (18.6) requires $O\left(\sum_{x=1}^{k} n^{[x]}\right) = O(n)$ time. After that, all values of $b_v^{[x]}$, $1 \le v \le x-1$, $1 \le x \le k$, can be found by (18.9) in $O\left(k^2\right)$ time. Following Sect. 18.2, computing all values $E^{[v,x]}$, $1 \le v \le x-1$, $1 \le x \le k$, requires $O\left(2^k\right)$ time. Finally, all $n$ positional weights $W^{[x]}(r)$, $1 \le r \le n^{[x]}$, $1 \le x \le k$, will be computed in $O(n)$ time, provided that all other quantities have been found. Thus, for an auxiliary problem $1|\text{Effect (18.2)}, RMP(k-1), \Delta^{[x]}(\tau; \varpi)|C_{\max}$ is $T(W) = O\left(2n + k^2 + 2^k\right) = O(n)$, provided that $k$ is a given constant.

The function (18.16) admits a generic representation (12.3), and Procedure RMP1 is in principle applicable. In particular, for each outcome of Decision (B1), i.e., for fixed values $n^{[x]}$, $1 \le x \le k$, of the numbers of jobs in each group, schedule $S_{\mathrm{B1}}^{*}(k)$ that corresponds to the smallest value of function (18.16) can be found by solving an LAP. Notice that the weights are job-independent so that for each outcome of Decision (B1) the corresponding LAP will have a product cost matrix (see Sect. 12.4).

Notice that the computed positional weights $W^{[x]}(r)$, $1 \le r \le n^{[x]}$, given by (18.17) may be non-monotonically ordered within each group $x$, $1 \le x \le k$. Additionally, the term $n^{[x]}$ appears in (18.17), and thus, it is not possible to generate a set of all possible values of $W^{[x]}(r)$, $1 \le r \le n$, $1 \le x \le k$, without prior knowledge of the number $n^{[x]}$ of jobs in each group. As a result, Theorem 16.2 does not hold and none of the solution approaches presented in Chaps. 16 and 17 can be applied.

Thus, in order to solve problem $1|\text{Effect (18.2)}, RMP(K), \Delta^{[x]}(\tau;\varpi)|C_{\max}$, we apply Procedure RMP1, which involves solving an LAP with a product matrix as a subroutine in Step 1(b), so that the following statement holds.

**Theorem 18.1** *Problem* $1|\text{Effect } (18.2), RMP(K), \Delta^{[x]}(\tau;\varpi)|C_{\max}$ *can be solved in* $O\left(n^{K+1}\log n\right)$ *time by applying Procedure RMP1 and using Algorithm Match as a subroutine.*

*Proof* According to Lemma 12.1, the number of times an LAP will have to be solved is equal to $O\left(n^K\right)$. It takes $T(W) = O(n)$ time to compute all positional weights given by (18.17). The running time required to run Algorithm Match used for solving an LAP with a product matrix is equal to $O(n \log n)$. Thus, the total running time required to solve problem $1|\text{Effect (18.2)}, RMP(K), \Delta^{[x]}(\tau;\varpi)|C_{\max}$ is given by $O\left((n + n\log n)n^K\right) = O\left(n^{K+1}\log n\right)$. □

At the end of Sect. 18.4, we present a numerical example, in which we solve the problem of minimizing the makespan for the situation discussed in Example 18.1.

## 18.4 Minimizing the Total Completion Time

We now address problem $1|\text{Effect (18.2)}, RMP(k-1), \Delta^{[x]}(\tau;\varpi)|\sum C_j$. For a schedule $S_{\mathrm{B1}}(k)$ with $k$ groups associated with a permutation $\pi = \left(\pi^{[1]}, \pi^{[2]}, \dots, \pi^{[k]}\right)$, the total completion time can be written as

$$F(S_{\mathrm{B1}}(k)) = \sum C_j(S_{\mathrm{B1}}(k)) = \sum_{x=1}^{k}\sum_{r=1}^{n^{[x]}} C_{\pi^{[x]}(r)}.$$

Substituting (18.5) into (18.14), we rewrite the expression for $C_{\pi^{[x]}(r)}$ as

$$\begin{aligned}
C_{\pi^{[x]}(r)} &= \sum_{v=1}^{x-1}\xi^{[v,x-1]}F_v + \sum_{u=1}^{r}\left(A^{[x]} + p_{\pi^{[x]}(u)}\right)B^{[x]}(u,r) + \sum_{v=1}^{x-1}\widehat{\eta}^{[v]} \\
&= \sum_{v=1}^{x-1}\xi^{[v,x-1]}F_v + A^{[x]}\sum_{u=1}^{r}B^{[x]}(u,r) + \sum_{u=1}^{r}p_{\pi^{[x]}(u)}B^{[x]}(u,r) + \sum_{v=1}^{x-1}\widehat{\eta}^{[v]}.
\end{aligned}$$

Substituting $A^{[x]} = \sum_{v=1}^{x-1}a_v^{[x]}F_v$, into the above equation, we obtain

$$C_{\pi^{[x]}(r)} = \sum_{v=1}^{x-1}\left(\xi^{[v,x-1]} + a_v^{[x]}\sum_{u=1}^{r}B^{[x]}(u,r)\right)F_v + \sum_{u=1}^{r}p_{\pi^{[x]}(u)}B^{[x]}(u,r) + \sum_{v=1}^{x-1}\widehat{\eta}^{[v]}.$$

Thus, the total completion time of schedule $S_{\mathrm{B1}}(k)$ can be written as

$$\begin{aligned} F(S_{\mathrm{B1}}(k)) &= \sum_{x=1}^{k}\sum_{r=1}^{n^{[x]}}\left[\sum_{v=1}^{x-1}\left(\xi^{[v,x-1]} + a_v^{[x]}\sum_{u=1}^{r}B^{[x]}(u,r)\right)F_v + \sum_{u=1}^{r}p_{\pi^{[x]}(u)}B^{[x]}(u,r)\right] + \Lambda(k) \\ &= \sum_{x=1}^{k}\left[\sum_{v=1}^{x-1}\left(n^{[x]}\xi^{[v,x-1]} + a_v^{[x]}\sum_{r=1}^{n^{[x]}}\sum_{u=1}^{r}B^{[x]}(u,r)\right)F_v + \sum_{r=1}^{n^{[x]}}\sum_{u=1}^{r}p_{\pi^{[x]}(u)}B^{[x]}(u,r)\right] \\ &\quad + \Lambda(k), \end{aligned}$$

where the constant term is given by

$$\Lambda(k) := \sum_{x=1}^{k}\sum_{r=1}^{n^{[x]}}\sum_{v=1}^{x-1}\widehat{\eta}^{[v]}. \tag{18.18}$$

It can be easily verified that the term $\sum_{r=1}^{n^{[x]}}\sum_{u=1}^{r}p_{\pi^{[x]}(u)}B^{[x]}(u,r)$ can be rewritten as $\sum_{r=1}^{n^{[x]}}p_{\pi^{[x]}(r)}\sum_{u=r}^{n^{[x]}}B^{[x]}(r,u)$, $1 \le x \le k$, so that we have

$$\begin{aligned} F(S_{\mathrm{B1}}(k)) &= \sum_{x=1}^{k}\left[\sum_{v=1}^{x-1}\left(n^{[x]}\xi^{[v,x-1]} + a_v^{[x]}\sum_{r=1}^{n^{[x]}}\sum_{u=r}^{n^{[x]}}B^{[x]}(r,u)\right)F_v + \sum_{r=1}^{n^{[x]}}p_{\pi^{[x]}(r)}\sum_{u=r}^{n^{[x]}}B^{[x]}(r,u)\right] \\ &\quad + \Lambda(k). \end{aligned}$$

For convenience, substitute the value $\xi^{[v,x-1]}$ from (18.13) and denote

$$G^{[v,x]} := n^{[x]}\left(1 + \sum_{w=v}^{x-1}\zeta_v^{[w]}\right) + a_v^{[x]}\sum_{r=1}^{n^{[x]}}\sum_{u=r}^{n^{[x]}}B^{[x]}(r,u),\ 1 \le v \le x-1,\ 1 \le x \le k, \tag{18.19}$$

so that we have

$$F(S_{\mathrm{B1}}(k)) = \sum_{x=1}^{k}\left(\sum_{v=1}^{x-1}G^{[v,x]}F_v + \sum_{r=1}^{n^{[x]}}p_{\pi^{[x]}(r)}\sum_{u=r}^{n^{[x]}}B^{[x]}(r,u)\right) + \Lambda(k).$$

Substituting the value of $F_v$ from (18.11) into the above equation, we deduce

$$\begin{aligned} F(S_{\mathrm{B1}}(k)) &= \sum_{x=1}^{k}\left(\sum_{v=1}^{x-1}G^{[v,x]}\sum_{w=1}^{v}E^{[w,v]}D^{[w]} + \sum_{r=1}^{n^{[x]}}p_{\pi^{[x]}(r)}\sum_{u=r}^{n^{[x]}}B^{[x]}(r,u)\right) + \Lambda(k) \\ &= \sum_{x=1}^{k}\left[\sum_{v=1}^{x-1}\left(\sum_{w=v}^{x-1}G^{[w,x]}E^{[v,w]}\right)D^{[v]} + \sum_{r=1}^{n^{[x]}}p_{\pi^{[x]}(r)}\sum_{u=r}^{n^{[x]}}B^{[x]}(r,u)\right] + \Lambda(k) \end{aligned}$$

$$= \sum_{x=1}^{k} \sum_{v=x+1}^{k} \left( \sum_{w=x}^{v-1} G^{[w,v]} E^{[x,w]} \right) D^{[x]} + \sum_{x=1}^{k} \sum_{r=1}^{n^{[x]}} p_{\pi^{[x]}(r)} \sum_{u=r}^{n^{[x]}} B^{[x]}(r,u) + \Lambda(k)$$

$$= \sum_{x=1}^{k} D^{[x]} \sum_{v=x+1}^{k} \sum_{w=x}^{v-1} G^{[w,v]} E^{[x,w]} + \sum_{x=1}^{k} \sum_{r=1}^{n^{[x]}} p_{\pi^{[x]}(r)} \sum_{u=r}^{n^{[x]}} B^{[x]}(r,u) + \Lambda(k).$$

Further, substituting the value of $D^{[x]}$ from (18.8) and rearranging terms, we obtain

$$F(S_{\mathrm{B1}}(k)) = \sum_{x=1}^{k} \left[ \left( \sum_{r=1}^{n^{[x]}} p_{\pi^{[x]}(r)} B^{[x]}\left(r, n^{[x]}\right) \right) \sum_{v=x+1}^{k} \sum_{w=x}^{v-1} G^{[w,v]} E^{[x,w]} \right.$$
$$\left. + \sum_{r=1}^{n^{[x]}} p_{\pi^{[x]}(r)} \sum_{u=r}^{n^{[x]}} B^{[x]}(r,u) \right] + \Lambda(k)$$
$$= \sum_{x=1}^{k} \sum_{r=1}^{n^{[x]}} p_{\pi^{[x]}(r)} \left( B^{[x]}\left(r, n^{[x]}\right) \sum_{v=x+1}^{k} \sum_{w=x}^{v-1} G^{[w,v]} E^{[x,w]} + \sum_{u=r}^{n^{[x]}} B^{[x]}(r,u) \right)$$
$$+ \Lambda(k).$$

Thus, the total completion time can be represented as

$$F(S_{\mathrm{B1}}(k)) = \sum_{x=1}^{k} \sum_{r=1}^{n^{[x]}} W^{[x]}(r) p_{\pi^{[x]}(r)} + \Lambda(k), \tag{18.20}$$

with the positional weights defined by

$$W^{[x]}(r) = B^{[x]}\left(r, n^{[x]}\right) \sum_{v=x+1}^{k} \sum_{w=x}^{v-1} G^{[w,v]} E^{[x,w]} + \sum_{u=r}^{n^{[x]}} B^{[x]}(r,u), \ 1 \le r \le n^{[x]}, \ 1 \le x \le k. \tag{18.21}$$

The expression for the positional weights (18.18) can be written in terms of the original parameters by the use of (18.6), (18.19), (18.12), and (18.9).

All values $B^{[x]}(u,r)$, $1 \le u \le r \le n^{[x]}$, $1 \le x \le k$, can be computed by (18.6) in $O\left(\sum_{x=1}^{k} \frac{n^{[x]}\left(n^{[x]}+1\right)}{2}\right) = O(n^2)$ time. After that, all values of $G^{[v,x]}$, $1 \le v \le x-1$, $1 \le x \le k$, can be found by (18.19) in $O\left(k^2\right)$ time. As discussed in Sect. 18.3, computing all values $E^{[v,x]}$, $1 \le v \le x-1, 1 \le x \le k$, requires $O\left(2^k\right)$ time. Finally, all $n$ positional weights $W^{[x]}(r)$, $1 \le r \le n^{[x]}$, $1 \le x \le k$, will be computed in $O(n)$ time, provided that all other quantities have been found. Thus, the overall running time needed to compute the positional weights for an auxiliary problem $1|\text{Effect (18.2)}, RMP(k-1), \Delta^{[x]}(\tau; \varpi)|\sum C_j$ is $T(W) = O\left(n + n^2 + k^2 + 2^k\right) = O\left(n^2\right)$, provided that $k$ is a given constant.

Similar to Sect. 18.3, the function (18.20) admits a generic representation (12.3). Thus, in order to solve problem $1|\text{Effect (18.2)}, RMP(K), \Delta^{[x]}(\tau;\varpi)|C_{\max}$, we apply Procedure RMP1, which involves solving an LAP with a product matrix as a subroutine in Step 1(b), so that the following statement holds.

**Theorem 18.2** *Problem* $1|\text{Effect}$ *(18.2),* $RMP(K), \Delta^{[x]}(\tau;\varpi)|\sum C_j$ *can be solved in* $O\left(n^{K+2}\right)$ *time-applying Procedure RMP1 and using Algorithm Match as a subroutine.*

*Proof* The proof of Theorem 18.2 is similar to the proof of Theorem 18.1. The only difference is that it takes $T(W) = O\left(n^2\right)$ time to compute all positional weights given by (18.21). Thus, the total running time required to solve problem $1|\text{Effect (18.2)}$, $RMP(K), \Delta^{[x]}(\tau;\varpi)|\sum C_j$ is given by $O\left(\left(n^2 + n\log n\right)n^K\right) = O\left(n^{K+2}\right)$. □

Below we provide a numerical example, in which we solve the problems of minimizing the makespan and the total completion time for the situation discussed in Example 18.1. We present the solution for a particular combination of outcomes (A1), (A2) and, (B1).

*Example 18.2* Eight jobs must be processed on a single machine. The normal processing times of the jobs, after renumbering them in LPT order, are:

$$p_1 = 8,\ p_2 = 7,\ p_3 = 6,\ p_4 = 6,\ p_5 = 4,\ p_6 = 2,\ p_7 = 1,\ p_8 = 1.$$

The number of jobs in each of the three groups is known in advance as

$$n^{[1]} = 3,\ n^{[2]} = 2,\ n^{[3]} = 3.$$

The values of the parameters for the machine and the operators as defined in Example 18.1 are also known and given by

$$d'_m = 0.1,\ d''_m = 0.15,\ d'_w = 0.2,\ l'_w = -0.15,\ d''_w = 0.15,\ l''_w = -0.1,\ l'''_w = -0.02,$$
$$\zeta' = 2,\ \eta' = 5,\ \eta'' = 6.$$

Using the formulae provided in Table 18.1, the functions for the positional factors reduce to

$$g^{[1]}(r) = \left(d'_m + 1\right)^{r-1} r^{d'_w + l'_w} = (1.1)^{r-1} r^{0.05},\ 1 \le r \le 3;$$
$$g^{[2]}(r) = \left(d''_m + 1\right)^{r-1}\left(n^{[1]} + r\right)^{l'_w} r^{d'_w} = (1.15)^{r-1}\frac{r^{0.2}}{(r+3)^{0.15}},\ 1 \le r \le 2;$$
$$g^{[3]}(r) = \left(d''_m + 1\right)^{n^{[2]}+r-1} r^{d''_w + l''_w} = (1.15)^{r+1} r^{0.05},\ 1 \le r \le 3,$$

**Table 18.2** Values of $B^{[x]}(u,r)$, $1 \le u \le r \le n^{[x]}$, $1 \le x \le 3$, for Example 18.2

| | | | |
|---|---|---|---|
| Group 1 | $B^{[1]}(1,1) = 1.00$; | | |
| | $B^{[1]}(1,2) = 1.17$, | $B^{[1]}(2,2) = 1.14$; | |
| | $B^{[1]}(1,3) = 1.40$, | $B^{[1]}(2,3) = 1.36$, | $B^{[1]}(3,3) = 1.28$; |
| Group 2 | $B^{[2]}(1,1) = 0.81$; | | |
| | $B^{[2]}(1,2) = 0.98$, | $B^{[2]}(2,2) = 1.04$; | |
| Group 3 | $B^{[3]}(1,1) = 1.32$; | | |
| | $B^{[3]}(1,2) = 1.74$, | $B^{[3]}(2,2) = 1.57$; | |
| | $B^{[3]}(1,3) = 2.38$, | $B^{[3]}(2,3) = 2.16$, | $B^{[3]}(3,3) = 1.85$. |

and all required input parameters are computed as

$$\begin{array}{ll}
g^{[1]}(1) = 1.00,\ g^{[1]}(2) = 1.14,\ g^{[1]}(3) = 1.28; & a_1^{[1]} = 0.15;\\
g^{[2]}(1) = 0.81,\ g^{[2]}(2) = 1.04; & a_1^{[2]} = -0.15,\ a_2^{[2]} = 0.20;\\
g^{[3]}(1) = 1.32,\ g^{[3]}(2) = 1.57,\ g^{[3]}(3) = 1.85; & a_1^{[3]} = -0.02,\ a_2^{[3]} = 0.13,\ a_3^{[3]} = 0.20.
\end{array}$$

and

$$\zeta_1^{[1]} = 2,\ \widehat{\eta}^{[1]} = 5,\ \zeta_1^{[2]} = \zeta_2^{[2]} = 0,\ \widehat{\eta}^{[2]} = 6.$$

Notice that while solving this example, we have computed all values with high precision, but here and below for the ease of presentation we report them rounded to two decimal places. Applying (18.6), all values of $B^{[x]}(u,r)$, $1 \le u \le r \le n^{[x]}$, $1 \le x \le 3$, are computed and are presented in Table 18.2. For the problem of minimizing the makespan, we will only need to use the values given in the last row of each block in Table 18.2, whereas for the problem of minimizing the total completion time we will require all of them.

Applying (18.9), all values of $b_v^{[x]}$, $1 \le v \le x-1$, $1 \le x \le 3$, are computed as

$$\begin{aligned}
b_1^{[2]} &= (-0.15)(0.98 + 1.04) = -0.30;\\
b_1^{[3]} &= (-0.02)(2.38 + 2.16 + 1.85) = -0.13,\\
b_2^{[3]} &= (0.13)(2.38 + 2.16 + 1.85) = 0.83.
\end{aligned}$$

Applying (18.12), all values of $E^{[v,x]}$, $1 \le v \le x$, $1 \le x \le 3$, are computed as

$$\begin{aligned}
&E^{[1,1]} = 1.00;\\
&E^{[1,2]} = b_1^{[2]} = -0.30,\ E^{[2,2]} = 1.00;\\
&E^{[1,3]} = b_1^{[3]} + b_1^{[2]} b_2^{[3]} = -0.38,\ E^{[2,3]} = b_2^{[3]} = 0.83,\ E^{[3,3]} = 1.00.
\end{aligned}$$

Finally, applying (18.17), the positional weights $W^{[x]}(r)$, $1 \le r \le n^{[x]}$, $1 \le x \le 3$, for the problem of minimizing the makespan can be rewritten as

$$W^{[1]}(r) = B^{[1]}(r,3)\left(\left(1+\zeta_1^{[1]}+\zeta_1^{[2]}\right)E^{[1,1]}+\left(1+\zeta_2^{[2]}\right)E^{[1,2]}+E^{[1,3]}\right),\ 1 \le r \le 3;$$
$$W^{[2]}(r) = B^{[2]}(r,2)\left(\left(1+\zeta_2^{[2]}\right)E^{[2,2]}+E^{[2,3]}\right),\ 1 \le r \le 2;$$
$$W^{[3]}(r) = B^{[3]}(r,3)\left(E^{[3,3]}\right),\ 1 \le r \le 3,$$

and their values computed as

$$W^{[1]}(1) = 3.23,\ W^{[1]}(2) = 3.15,\ W^{[1]}(3) = 2.96;$$
$$W^{[2]}(1) = 1.80,\ W^{[2]}(2) = 1.90;$$
$$W^{[3]}(1) = 2.38,\ W^{[3]}(2) = 2.16,\ W^{[3]}(3) = 1.85.$$

To solve problem $1|\text{Effect }(18.2), RMP(k-1), \Delta^{[x]}(\tau;\varpi)|C_{\max}$, the computed positional weights are sorted in non-decreasing order and stored in list $L' := \left(\gamma'_1, \gamma'_2, \ldots, \gamma'_8\right)$. The constant term $\Gamma(3)$ can be computed as $\Gamma(3) = \widehat{\eta}^{[1]} + \widehat{\eta}^{[2]} = 11.00$. The resulting optimal schedule $S'(3)$ is associated with a permutation $\mu^* = \left(\mu^{[1]}, \mu^{[2]}, \mu^{[3]}\right)$; all relevant computation is presented in Table 18.3.

Next, for the problem of minimizing the total completion time, we also must additionally compute the values of $G^{[v,x]}$, $1 \le v \le x-1$, $1 \le x \le 3$. Using (18.19) and the values obtained in Table 18.2, we compute

$$G^{[1,2]} = 5.58,\ G^{[1,3]} = 8.78,\ G^{[2,3]} = 4.43.$$

Applying (18.21), the positional weights $W^{[x]}(r)$, $1 \le r \le n^{[x]}$, $1 \le x \le 3$, for the problem of minimizing the total completion time can be rewritten as

$$W^{[1]}(r) = B^{[1]}(r,3)\left(G^{[1,2]}E^{[1,1]}+G^{[1,3]}E^{[1,1]}+G^{[2,3]}E^{[1,2]}\right)+\sum_{u=r}^{3}B^{[1]}(r,u),\ 1 \le r \le 3;$$
$$W^{[2]}(r) = B^{[2]}(r,2)\left(G^{[2,3]}E^{[2,2]}\right)+\sum_{u=r}^{2}B^{[2]}(r,u),\ 1 \le r \le 2;$$
$$W^{[3]}(r) = \sum_{u=r}^{3}B^{[3]}(r,u),\ 1 \le r \le 3,$$

and their values computed as

$$W^{[1]}(1) = 21.72,\ W^{[1]}(2) = 20.16,\ W^{[1]}(3) = 17.91;$$
$$W^{[2]}(1) = 6.14,\ W^{[2]}(2) = 5.64;$$
$$W^{[3]}(1) = 5.44,\ W^{[3]}(2) = 3.73,\ W^{[3]}(3) = 1.85.$$

To solve problem $1|\text{Effect }(18.2), RMP(k-1), \Delta^{[x]}(\tau;\varpi)|\sum C_j$, the computed positional weights are sorted in non-decreasing order and stored in list $L'' := \left(\gamma''_1, \gamma''_2, \ldots, \gamma''_n\right)$. The constant term $\Lambda(3)$ can be computed as $\Lambda(3) =$

**Table 18.3** Calculation of the optimal value of the makespan and the optimal value of the total flow time for the problem outlined in Example 18.2

| $j$ | $p_j$ | $\gamma'_j$ | $\mu^*$ | $\gamma'_j p_j$ | $\gamma''_j$ | $\varphi^*$ | $\gamma''_j p_j$ |
|---|---|---|---|---|---|---|---|
| | | Makespan | | | Flow time | | |
| 1 | 8 | 1.80 | $\mu^{[2]}(1)$ | 14.36 | 1.85 | $\varphi^{[3]}(3)$ | 14.78 |
| 2 | 7 | 1.85 | $\mu^{[3]}(3)$ | 12.93 | 3.73 | $\varphi^{[3]}(2)$ | 26.12 |
| 3 | 6 | 1.90 | $\mu^{[2]}(2)$ | 11.39 | 5.44 | $\varphi^{[3]}(1)$ | 32.66 |
| 4 | 6 | 2.16 | $\mu^{[3]}(2)$ | 12.94 | 5.64 | $\varphi^{[2]}(2)$ | 33.82 |
| 5 | 4 | 2.38 | $\mu^{[3]}(1)$ | 9.53 | 6.14 | $\varphi^{[2]}(1)$ | 24.56 |
| 6 | 2 | 2.96 | $\mu^{[1]}(3)$ | 5.93 | 17.91 | $\varphi^{[1]}(3)$ | 35.83 |
| 7 | 1 | 3.15 | $\mu^{[1]}(2)$ | 3.15 | 20.16 | $\varphi^{[1]}(2)$ | 20.16 |
| 8 | 1 | 3.23 | $\mu^{[1]}(1)$ | 3.23 | 21.72 | $\varphi^{[1]}(1)$ | 21.72 |
| | | Total | | 73.46 | Total | | 209.65 |

$C_{\max}(S'(3)) = 73.46 + 11.00 = 84.46$
$\sum C_j(S''(3)) = 209.65 + 43.00 = 252.65$
$\mu^* = (8, 7, 6, 1, 3, 5, 4, 2)$; $\varphi^* = (8, 7, 6, 5, 4, 3, 2, 1)$

$\widehat{\eta}^{[1]} n^{[2]} + (\widehat{\eta}^{[1]} + \widehat{\eta}^{[2]}) n^{[3]} = 43.00$. The resulting optimal schedule $S''(3)$ is associated with a permutation $\varphi^* = (\varphi^{[1]}, \varphi^{[2]}, \varphi^{[3]})$; all relevant computation is presented in Table 18.3.

## 18.5 Some Reduced Models

In this section, we explore single machine models which can be expressed as special cases of the general problems $1|\text{Effect (18.2)}, RMP(K), \Delta^{[x]}(\tau; \varpi)|C_{\max}$ and $1|\text{Effect (18.2)}, RMP(K), \Delta^{[x]}(\tau; \varpi)|\sum C_j$. We look at their simplified versions with effects less general than that given by (18.2) and (18.3), and possibly with additional simplifications. The main purpose of this section is to verify whether the running time of the solution approach given in Sects. 18.3 and 18.4 for the general problems $1|\text{Effect (18.2)}, RMP(K), \Delta^{[x]}(\tau; \varpi)|C_{\max}$ and $1|\text{Effect (18.2)}, RMP(K), \Delta^{[x]}(\tau; \varpi)|\sum C_j$, respectively, can be reduced for their simplified counterparts.

### *18.5.1 Simple Combined Effects*

First, we demonstrate that as long as a time-changing effect combines both a positional effect and a (linear) start-time-dependent effect, the running times established in Sects. 18.3 and 18.4 for the general problems cannot be reduced.

Consider a situation in which a machine undergoes time-dependent deterioration and positional polynomial learning. Moreover, the machine is subject to $K$ identical maintenance periods with constant duration equal to $\eta$, all of which must be run. The model assumes that the effects are group-independent and after an RMP, the learning advantages are lost and both the operator and the machine are brought to the original conditions. For a schedule $S(k)$, with $k = K + 1$ groups, assume that each group contains a total of $n^{[x]}$ jobs, so that $\pi^{[x]} = \left(\pi^{[x]}(1), \pi^{[x]}(2), \dots, \pi^{[x]}\left(n^{[x]}\right)\right)$, $1 \le x \le k$, where $\sum_{x=1}^{k} n^{[x]} = n$. The actual processing time of a job $j = \pi^{[x]}(r)$, scheduled in position $r$, $1 \le r \le n^{[x]}$, of the $x$th group, $1 \le x \le k$, is given by

$$p_j^{[x]}(\tau; r) = \left(p_j + a\tau\right)r^b, 1 \le r \le n^{[x]},\ 1 \le x \le k, \tag{18.22}$$

where $a > 0$, and $b < 0$.

The resulting problem is denoted as $1|\text{Effect (18.22)}, RMP(K), \Delta|\Phi$, for $\Phi \in \left\{C_{\max}, \sum C_j\right\}$. Notice that for this problem, RMP Decisions 1–3 need not be taken, since the RMPs are identical and all of them are known to be included in any order, e.g., in the order of their numbering. As a result, problem $1|\text{Effect (18.22)}, RMP(K), \Delta|\Phi$ can be written as a special case of problem $1|\text{Effect (18.2)}, RMP(K), \Delta^{[x]}(\tau; \varpi)|\Phi$ with the parameters $g^{[x]}(r) = r^b$, and $a_1^{[x]} = a_2^{[x]} = \dots = a_{x-1}^{[x]} = 0$, $a_x^{[x]} = a$, for all $x$, $1 \le x \le k$, and $\zeta_1^{[x]} = \zeta_2^{[x]} = \dots = \zeta_{x-1}^{[x]} = \zeta_x^{[x]} = 0$, and $\widehat{\eta}^{[x]} = \eta$, for all $x$, $1 \le x \le k - 1 = K$.

It follows that the objective function for problem $1|\text{Effect (18.22)}, RMP(K), \Delta|C_{\max}$ may be written in the form (18.16), with the positional weights found by making appropriate substitutions in (18.17). It can be easily verified that the resulting positional weights for problem $1|\text{Effect (18.22)}, RMP(K), \Delta|C_{\max}$ can be found as

$$W^{[x]}(r) = r^b \prod_{i=r+1}^{n^{[x]}} \left(1 + ai^b\right), 1 \le r \le n^{[x]},\ 1 \le x \le K + 1. \tag{18.23}$$

Similar to computing the positional weights for an auxiliary problem, associated with the general problem $1|\text{Effect (18.2)}, RMP(K), \Delta^{[x]}(\tau; \varpi)|C_{\max}$, the running time required to compute the positional weights for an auxiliary problem $1|\text{Effect (18.22)}, RMP(k-1), \Delta|C_{\max}$ (with $k = K + 1$) using (18.23) is given by $T(W) = O(n)$. Also, notice that the computed positional weights $W^{[x]}(r)$, $1 \le r \le n^{[x]}$, $1 \le x \le K + 1$, are non-monotonically ordered within each group $x$, $1 \le x \le k$, since the terms $a$ and $b$ are of opposite signs. Additionally, the term $n^{[x]}$ appears in (18.23), and thus, it is not possible to generate a set of all possible values of $W^{[x]}(r)$, $1 \le r \le n$, $1 \le x \le k$, without prior knowledge of the number of jobs, $n^{[x]}$, in each group. As a result, Theorem 16.2 does not hold and in order to solve problem $1|\text{Effect (18.22)}, RMP(K), \Delta|C_{\max}$, we must apply Procedure RMP1 which involves full enumeration of outcomes of Decision (B1) and solving an LAP with a product matrix as a subroutine in Step 1(b). According to Lemma 12.1, the number of these outcomes for $k = K + 1$ is $O\left(n^K\right)$. Thus, the results of Theorem 18.1 are applicable, so that the resulting running time to solve problem 1|Effect (18.22),

$RMP(K), \Delta\,|C_{\max}$ is equal to $O\left(n^{K+1}\log n\right)$, which is the same as that required for solving the general problem $1|\text{Effect}$ (18.2), $RMP(K), \Delta^{[x]}(\tau;\varpi)\,|C_{\max}$.

Notice that problem $1|\text{Effect}$ (18.22), $RMP(K), \Delta\,|C_{\max}$ may admit a faster running time if the terms $a$ and $b$ are of the same sign. We are still required to generate all values of $n^{[x]}$, since they appears in (18.23), but because $a$ and $b$ have the same sign, the found positional weights $W^{[x]}(r)$, $1 \le r \le n^{[x]}$, are monotonically ordered in each group, $x$, $1 \le x \le k$. Thus, sorting all positional weights in a non-decreasing order requires $O(n\min\{K,\log n\})$ time instead of $O(n\log n)$ time. As a result, Algorithm Match requires $O(n\min\{K,\log n\})$ time and the overall running time of the problem can be given as $O\left((n+n\min\{K,\log n\})n^K\right) = O\left(n^{K+1}\min\{K,\log n\}\right)$.

Now, let us consider problem $1|\text{Effect}$ (18.22), $RMP(K), \Delta\,|\sum C_j$. The resulting objective function may be written in the form (18.20), with positional weights found by making appropriate substitutions in (18.21). It can be easily verified that the resulting positional weights for problem $1|\text{Effect}$ (18.22), $RMP(K), \Delta\,|\sum C_j$ can be found as

$$W^{[x]}(r) = \left(n - \sum_{v=1}^{x} n^{[v]}\right) r^b \prod_{i=r+1}^{n^{[x]}}\left(1+ai^b\right) + r^b \sum_{u=r}^{n^{[x]}} \prod_{i=r+1}^{u}\left(1+ai^b\right), \qquad (18.24)$$
$$1 \le r \le n^{[x]},\ 1 \le x \le k.$$

Similar to computing the positional weights for an auxiliary problem, associated with the general problem $1|\text{Effect}$ (18.2), $RMP(K), \Delta^{[x]}(\tau;\varpi)\,|\sum C_j$, the running time required to compute the positional weights for an auxiliary problem $1|\text{Effect}$ (18.22), $RMP(k-1), \Delta\,|\sum C_j$ (with $k = K+1$) using (18.24) is given by $T(W) = O\left(n^2\right)$, since the total number of terms to be calculated is $O\left(n^2\right)$. Also, notice that the term $n^{[x]}$ appears in (18.24), and thus, similar to the situation of problem $1|\text{Effect}$ (18.22), $RMP(K), \Delta\,|C_{\max}$, we apply Procedure RMP1, which involves full enumeration of outcomes of Decision (B1) and solving an LAP with a product matrix as a subroutine in Step 1(b). Thus, the results of Theorem 18.2 are applicable, so that the resulting running time to solve problem $1|\text{Effect}$ (18.22), $RMP(K), \Delta\,|\sum C_j$ is equal to $O\left(n^{K+2}\right)$, which is the same as that required for solving the general problem $1|\text{Effect}$ (18.2), $RMP(K), \Delta^{[x]}(\tau;\varpi)\,|\sum C_j$.

**Theorem 18.3** *Problem* $1|\text{Effect}$ *(18.22),* $RMP(K), \Delta^{[x]}(\tau;\varpi)\,|\Phi$, *for* $\Phi \in \left\{C_{\max}, \sum C_j\right\}$ *can be solved in* $O\left(n^{K+1}\log n\right)$ *and* $O\left(n^{K+2}\right)$ *time, respectively, by applying Procedure RMP1 and using Algorithm Match as a subroutine.*

Hence, it can be seen that as long as a model incorporates a combined time-dependent and a positional effect, even the simplest versions of the general problem $1|\text{Effect}$ (18.2), $RMP(K), \Delta^{[x]}(\tau;\varpi)\,|\Phi$, for $\Phi \in \left\{C_{\max}, \sum C_j\right\}$, require $O\left(n^{K+1}\log n\right)$ and $O\left(n^{K+2}\right)$ time, respectively. However, if either a pure positional effect or a pure time-dependent effect is considered, faster solutions are possible.

## 18.5.2 Pure Positional Effects

In this subsection, we discuss a version general problem of the form $1|\text{Effect (18.2)}, RMP(K), \Delta^{[x]}(\tau;\varpi)|\Phi$ with a pure positional effect, so that for schedule $S_{\mathrm{B1}}(k)$ associated with a relevant auxiliary problem, the actual processing time of a job $j$ scheduled in position $r$ of a group $x$, $1 \le x \le k$, is given by

$$p_j^{[x]}(r) = p_j g^{[x]}(r),\ 1 \le r \le n,\ 1 \le x \le k. \tag{18.25}$$

A similar model is considered in Chap. 16 for a deterioration environment, in which the positional factors $g^{[x]}(r)$, $1 \le r \le n$, $1 \le x \le k$, are in a non-decreasing order and the RMPs are understood as maintenance periods. Moreover, in Chap. 16 we do not allow the positional factors to be dependent on the number of jobs scheduled in previous groups.

In this subsection, we study positional effects without these restrictions, so that the positional factors can be non-monotone within a group and the RMPs can be of an arbitrary nature with their durations given by (18.3). In other words, we consider position-dependent models that combine deterioration and learning effects with rate-modifying activities. An illustration of such a scenario is presented in Example 12.1. Let us denote problems of this type by $1|\text{Effect (18.25)}, RMP(K), \Delta^{[x]}(\tau;\varpi)|\Phi$, where $\Phi \in \left\{C_{\max}, \sum C_j\right\}$.

Assuming that a particular combination of outcomes (A1) and (A2) is known and a certain outcome of Decision (B1) is chosen, the resulting auxiliary problem $1|\text{Effect (18.25)}, RMP(k-1), \Delta^{[x]}(\tau;\varpi)|\Phi$ can be seen as a special case of problem $1|\text{Effect (18.2)}, RMP(k-1), \Delta^{[x]}(\tau;\varpi)|\Phi$ with $a_1^{[x]} = a_2^{[x]} = \cdots = a_{x-1}^{[x]} = a_x^{[x]} = 0$, for all $x$, $1 \le x \le k$.

To solve an auxiliary problem $1|\text{Effect (18.25)}, RMP(k-1), \Delta^{[x]}(\tau;\varpi)|C_{\max}$, the required positional weights can be obtained by making relevant substitutions in (18.17), so that we have

$$W^{[x]}(r) = \begin{cases} \left(1 + \sum_{w=x}^{k-1} \zeta_x^{[w]}\right) g^{[x]}(r), & 1 \le r \le n^{[x]},\ 1 \le x \le k-1, \\ g^{[x]}(r) & 1 \le r \le n^{[x]},\ x = k. \end{cases} \tag{18.26}$$

The running time required to compute the above positional weights is given by $T(W) = O(n)$, as for a more general problem considered in Sect. 18.3.

To solve problem $1|\text{Effect (18.25)}, RMP(k-1), \Delta^{[x]}(\tau;\varpi)|\sum C_j$, the required positional weights can be derived from (18.21), so that we have

$$W^{[x]}(r) = \begin{cases} \left[\sum_{v=x+1}^{k} n^{[v]}\left(1 + \sum_{w=x}^{v-1} \zeta_x^{[w]}\right) + \left(n^{[x]} - r + 1\right)\right] g^{[x]}(r), & \begin{array}{l} 1 \le r \le n^{[x]}, \\ 1 \le x \le k-1, \end{array} \\ \left(n^{[x]} - r + 1\right) g^{[x]}(r), & \begin{array}{l} 1 \le r \le n^{[x]}, \\ x = k. \end{array} \end{cases} \tag{18.27}$$

To calculate the running time required to compute the above positional weights, rewrite (18.27), so that for each $r$, $1 \le r \le n^{[x]}$, the weights are found by

$$W^{[x]}(r) = \begin{cases} \left(\left(n - r + 1 - \sum_{v=1}^{x-1} n^{[v]}\right) + \sum_{v=x}^{k-1} \zeta_x^{[v]}\left(n - \sum_{w=1}^{v} n^{[w]}\right)\right) g^{[x]}(r), & 1 \le x \le k-1; \\ \left(n^{[x]} - r + 1\right) g^{[x]}(r), & x = k. \end{cases}$$

Notice that for a given group $x$, $1 \le x \le k$, the term $\left(n - \sum_{v=1}^{x-1} n^{[v]}\right) + \sum_{v=x}^{k-1} \zeta_x^{[v]} \left(n - \sum_{w=1}^{v} n^{[w]}\right)$ is constant and essentially requires the summation of $(k - x + 1)$ terms, so that it can be computed in $O(k - x + 1)$ time. If the value of this term is known, the value of the positional weight $W^{[x]}(r)$, for a given $r$, $1 \le r \le n^{[x]}$, $1 \le x \le k$, can be found in constant time. Thus, for a given group, the total number of operations required is $O\left(k - x + 1 + n^{[x]}\right)$, $1 \le x \le k$. For all groups, the running time to compute all required positional weights will be equal to $T(W) = O\left(\sum_{x=1}^{k}\left(x + n^{[x]}\right)\right) = O\left(n + k^2\right) = O(n)$, assuming $k$ is a constant.

It can be observed that the found positional weights for both problems are possibly non-monotonically ordered within each group. Moreover, they do not allow us to generate a set of all possible values of $W^{[x]}(r)$, $1 \le r \le n$, $1 \le x \le k$, without prior knowledge of the number of jobs, $n^{[x]}$, in each group. Therefore, we apply Procedure RMP1, which involves full enumeration of outcomes of Decision (B1) and solving an LAP with a product matrix as a subroutine in Step 1(b), so that the following statement holds.

**Theorem 18.4** *Problem* $1|\text{Effect}$ *(18.25),* $RMP(K)$, $\Delta^{[x]}(\tau; \varpi)\,|\Phi$*, for* $\Phi \in \{C_{\max}, \sum C_j\}$ *can be solved in* $O\left(n^{K+1}\log n\right)$ *time by applying Procedure RMP1 and using Algorithm Match as a subroutine.*

The proof of Theorem 18.4 is similar to that of Theorem 18.1. It can be noted that although there is no change in status for the problem of minimizing the makespan, the problem of minimizing the total completion time is solved faster for this model with a pure positional effect. This speed up is possible because of the time taken to compute the positional weights $W^{[x]}(r)$, $1 \le r \le n^{[x]}$, $1 \le x \le k$, for an auxiliary problem $1|\text{Effect}$ (18.25), $RMP(K)$, $\Delta^{[x]}(\tau; \varpi)\,|\sum C_j$ is $T(W) = O(n)$, one order of magnitude less than required for the corresponding general problem $1|\text{Effect}$ (18.2), $RMP(k-1)$, $\Delta^{[x]}(\tau; \varpi)\,|\sum C_j$.

Let us now extend our consideration to problems in which a pure job-dependent positional effect is observed, so that for known outcomes (A1) and (A2), the actual processing time of a job $j$ scheduled in position $r$ of a group $x$, $1 \le x \le k$, in schedule $S_{\text{B1}}(k)$ is given by

$$p_j^{[x]}(r) = p_j g_j^{[x]}(r), \; 1 \le r \le n, \; 1 \le x \le k, \tag{18.28}$$

Recall that we consider such a model in Chap. 16 for a deterioration environment, in which the positional factors $g_j^{[x]}(r)$, $1 \le r \le n$, $1 \le x \le k$, are in a non-decreasing order for each $j \in N$, and the RMPs are understood as maintenance periods. Below, we study job-dependent positional effects without these restrictions. Denote the resulting problems by $1|\text{Effect (18.28)}, RMP(K), \Delta^{[x]}(\tau;\varpi)|\Phi$, where $\Phi \in \left\{C_{\max}, \sum C_j\right\}$. It can be easily verified by analogy with the job-independent case studied above that an individual auxiliary problem $1|\text{Effect (18.28)}, RMP(k-1), \Delta^{[x]}(\tau;\varpi)|\Phi$ reduces to minimizing a generic objective function

$$\Phi(\pi) = \sum_{x=1}^{k}\sum_{r=1}^{n^{[x]}} W^{[x]}_{\pi^{[x]}(r)}(r)\,p_{\pi^{[x]}(r)} + \Theta(k), \tag{18.29}$$

of the form (12.3).

For problem $1|\text{Effect (18.28)}, RMP(K), \Delta^{[x]}(\tau;\varpi)|C_{\max}$, the constant term $\Theta(k)$ is given by (18.15), and for each $j \in N$ the job-dependent positional weights $W_j^{[x]}(r)$, $1 \le r \le n^{[x]}$, $1 \le x \le k$, can be found by appropriately modifying (18.26), so that we have

$$W_j^{[x]}(r) = \begin{cases} \left(1 + \sum_{w=x}^{k-1} \zeta_x^{[w]}\right) g_j^{[x]}(r), & 1 \le r \le n^{[x]},\ 1 \le x \le k-1, \\ g_j^{[x]}(r) & 1 \le r \le n^{[x]},\ x = k. \end{cases} \tag{18.30}$$

For problem $1|\text{Effect (18.28)}, RMP(K), \Delta^{[x]}(\tau;\varpi)|\sum C_j$, the constant term $\Theta(k)$ is given by (18.18), and for each $j \in N$ the job-dependent positional weights $W_j^{[x]}(r)$, $1 \le r \le n^{[x]}$, $1 \le x \le k$, can be found by appropriately modifying (18.27), so that we have

$$W_j^{[x]}(r) = \begin{cases} \left[\sum_{v=x+1}^{k} n^{[v]}\left(1 + \sum_{w=x}^{v-1} \zeta_x^{[w]}\right) + \left(n^{[x]} - r + 1\right)\right] g_j^{[x]}(r), & 1 \le r \le n^{[x]}, \\ & 1 \le x \le k-1, \\ \left(n^{[x]} - r + 1\right) g_j^{[x]}(r), & 1 \le r \le n^{[x]}, \\ & x = k. \end{cases} \tag{18.31}$$

Notice that for both problems, all positional weights for a given $j \in N$ can be computed by (18.30) and (18.31), respectively, in $O(n)$ time each. The computation is done similarly to the job-independent version of the problem. As a result, for every $j \in N$, positional weights $W_j^{[x]}(r)$, $1 \le r \le n^{[x]}$, $1 \le x \le k$, $j \in N$, can be computed in $T(W) = O\left(n^2\right)$ time, for both problems.

The function (18.29) admits a generic representation (12.3), and Procedure RMP1 is applicable. Again, the found positional weights for both problems are possibly non-monotonically ordered within each group, and all $n^{[x]}$ values must be generated. As a result, the solution approach presented in Chap. 16 for job-dependent positional effects cannot be applied. Thus, in order to solve both problems, which involve full enumeration of outcomes of Decision (B1) and solving an LAP with an $n \times n$ cost matrix as a subroutine in Step 1(b), so that the following statement holds.

**Theorem 18.5** *Problem* $1|\text{Effect}$ *(18.28), $RMP(K)$, $\Delta^{[x]}(\tau;\varpi)\,|\Phi$, for $\Phi \in \{C_{\max}, \sum C_j\}$ can be solved in $O\left(n^{K+3}\right)$ time by applying Procedure RMP1 and solving a series of $n \times n$ linear assignment problems.*

*Proof* According to Lemma 12.1, the number of times an LAP will have to be solved is equal to $O\left(n^K\right)$. The running time required to solve an $n \times n$ LAP (see Sect. 4.1) is equal to $O\left(n^3\right)$. Moreover, it takes $T(W) = O\left(n^2\right)$ time to compute all positional weights given by (18.30) and (18.31), respectively. Thus, the total running time required to solve problem $1|\text{Effect}$ (18.28), $RMP(K)$, $\Delta^{[x]}(\tau;\varpi)\,|\Phi$, for $\Phi \in \left\{C_{\max}, \sum C_j\right\}$ is given by $O\left(\left(n^2 + n^3\right)n^K\right) = O\left(n^{K+3}\right)$. □

### *18.5.3 Pure Time-Dependent Effects*

In this subsection, we discuss a variant of the general problem of the form $1|\text{Effect}$ (18.2), $RMP(K)$, $\Delta^{[x]}(\tau;\varpi)\,|\Phi$ with a pure time-dependent effect, so that for schedule $S_{\text{BP1}}(k)$ associated with a relevant auxiliary problem, the actual processing time of a job $j$ scheduled in position $r$ of a group $x$, $1 \le x \le k$, is given by

$$p_j^{[x]}(r) = p_{\pi^{[x]}(r)} + a_1^{[x]}F_1 + a_2^{[x]}F_2 + \cdots + a_{x-1}^{[x]}F_{x-1} + a_x^{[x]}F_{(x,r-1)},\ 1 \le r \le n,\ 1 \le x \le k. \tag{18.32}$$

Notice that effect (18.32) is obtained from the general effect (18.2), by removing the positional factor $g^{[x]}(r)$. A reduced form of the time-dependent effect (18.32) is considered in Chap. 17, in which, however, we do not allow the duration of the previous groups to affect the actual processing time of the current job, so that $a_1^{[x]} = a_2^{[x]} = \cdots = a_{x-1}^{[x]} = 0$, for all $x$, $1 \le x \le k$. Moreover, in Chap. 17, we only study a deterioration environment, so that $a_x^{[x]} > 0$, $1 \le x \le k$, and the RMPs are understood as maintenance periods. In this subsection, we do not impose these restrictions, so that the rates $a_1^{[x]}, a_2^{[x]}, \ldots, a_x^{[x]}$, $1 \le x \le k$, can be of an arbitrary sign and the RMPs can be of an arbitrary nature with their durations given by (18.3). Let us denote problems of this type by $1|\text{Effect}$ (18.32), $RMP(K)$, $\Delta^{[x]}(\tau;\varpi)\,|\Phi$, where $\Phi \in \left\{C_{\max}, \sum C_j\right\}$.

Assuming that a particular combination of outcomes (A1) and (A2) is known and a certain outcome of Decision (B1) is chosen, the resulting auxiliary problem $1|\text{Effect}$ (18.32), $RMP(k-1)$, $\Delta^{[x]}(\tau;\varpi)\,|\Phi$ can be seen as a special case of problem $1|\text{Effect}$ (18.2), $RMP(k-1)$, $\Delta^{[x]}(\tau;\varpi)\,|\Phi$ with $g^{[x]}(r) = 1$, $1 \le r \le n$, for all $x$, $1 \le x \le k$.

To solve problem $1|\text{Effect}$ (18.32), $RMP(k-1)$, $\Delta^{[x]}(\tau;\varpi)\,|C_{\max}$, the required positional weights can be obtained by making relevant substitutions in (18.17), so that we have

$$W^{[x]}(r) = \left(1 + a_x^{[x]}\right)^{n^{[x]}-r}\left(\sum_{v=x}^{k-1}\left(1 + \zeta^{[v]}\right)E^{[x,v]} + E^{[x,k]}\right),\ 1 \le r \le n^{[x]},\ 1 \le x \le k, \quad (18.33)$$

where the quantities $E^{[v,x]}$, $1 \le v \le x-1$, $1 \le x \le k$, are given by (18.12), and by (18.9) the quantities $b_v^{[x]}$ reduce to

$$b_v^{[x]} = \frac{a_v^{[x]}}{a_x^{[x]}}\left(\left(1 + a_x^{[x]}\right)^{n^{[x]}} - 1\right),\ 1 \le v \le x-1,\ 1 \le x \le k.$$

Notice that all positional weights $W^{[x]}(r)$, $1 \le r \le n^{[x]}$, $1 \le x \le k$, can be computed by (18.33) in $T(W) = O(n)$ time.

Similarly, to solve problem $1|\text{Effect (18.32)}, RMP(k-1), \Delta^{[x]}(\tau;\varpi)|\sum C_j$, the required positional weights can be obtained by making relevant substitutions in (18.21), so that we have

$$W^{[x]}(r) = \left(1 + a_x^{[x]}\right)^{n^{[x]}-r}\sum_{v=x+1}^{k}\sum_{w=x}^{v-1}G^{[w,v]}E^{[x,w]} + \sum_{u=r}^{n^{[x]}}B^{[x]}(r,u),\ 1 \le r \le n^{[x]},\ 1 \le x \le k, \quad (18.34)$$

where the quantities $G^{[v,x]}$, $1 \le v \le x-1$, $1 \le x \le k$, are given by (18.19), while by (18.6) the quantities $B^{[x]}(r,u)$ reduce to

$$B^{[x]}(u,r) = \left(1 + a_x^{[x]}\right)^{r-u},\ 1 \le u \le r \le n^{[x]},\ 1 \le x \le k.$$

For a fixed $x$, $1 \le x \le k$, the difference $r-u$ takes at most $n^{[x]}-1$ values, so that at most $n^{[x]}-1$ distinct values of $B^{[x]}(u,r)$ need to be computed. Summing up for all $x$, we deduce that the number of all distinct values $B^{[x]}(u,r)$ to be found in order to compute the positional weights $W^{[x]}(r)$ is $O(n)$. As a result, all positional weights $W^{[x]}(r)$, $1 \le r \le n^{[x]}$, $1 \le x \le k$, can be computed by (18.34) in $T(W) = O(n)$ time.

It can be observed that the found positional weights for both problems, given by (18.33) and (18.34), respectively, are possibly non-monotonically ordered within each group, and as for most problems in this chapter, all possible values of $n^{[x]}$ have to be generated. Therefore, we apply Procedure RMP1, which involves full enumeration of outcomes of Decision (B1) and solving an LAP with a product matrix as a subroutine in Step 1(b). so that the following statement holds.

**Theorem 18.6** *Problem* $1|\text{Effect}$ *(18.32), $RMP(K)$, $\Delta^{[x]}(\tau;\varpi)|\Phi$, for $\Phi \in \{C_{\max}, \sum C_j\}$ can be solved in $O\left(n^{K+1}\log n\right)$ time by applying Procedure RMP1 and using Algorithm Match as a subroutine.*

The proof of Theorem 18.6 is similar to that of Theorem 18.1. It can be noted that although there is no change in status for the problem of minimizing the makespan, the problem of minimizing the total completion time is solved faster for a model with a pure start-time-dependent effect. This speed up is possible because of the time taken to compute the positional weights $W^{[x]}(r)$, $1 \le r \le n^{[x]}$, $1 \le x \le k$, for problem $1|\text{Effect (18.32)}, RMP(K), \Delta^{[x]}(\tau; \varpi)| \sum C_j$ is $T(W) = O(n)$, as opposed to $T(W) = O\left(n^2\right)$ required for problem $1|\text{Effect (18.2)}, RMP(K), \Delta^{[x]}(\tau; \varpi)| \sum C_j$.

## 18.6 Bibliographic Notes

Most of the results in this chapter follow from Rustogi and Strusevich (2014). The models introduced in this chapter cover most previously known models related to combined effects. Historically, several papers have been published with different combinations of learning and deterioration effects, for various objective functions. Below, we review several important publications, related to minimization of the makespan and the total completion time.

Yang (2010) studies problem $1\left|p_j^{[x]}(\tau; r) = \left(p_j - a\tau\right)r^b, RMP(1), \Delta(\tau)\right|\Phi$, where $\Phi \in \left\{C_{\max}, \sum C_j\right\}$, with a time-dependent learning effect and a polynomial deterioration effect, i.e., $g^{[x]}(r) = r^b$, $b > 0$, along with a single RMP of variable duration. The paper claims to solve the problem of minimizing the makespan and the problem of minimizing the total completion time in $O\left(n^2 \log n\right)$ time each.

Yang (2012) studies problem $1\left|p_j^{[x]}(\tau; r) = \left(p_j + a\tau\right)r^b, RMP(K), \Delta(\tau)\right|\Phi$, where $\Phi \in \left\{C_{\max}, \sum C_j\right\}$, with a time-dependent deterioration effect and a polynomial learning effect, i.e., $g^{[x]}(r) = r^b, b < 0$, along with $K$ identical RMPs of variable duration.
The paper claims to solve the problem of minimizing the makespan and the problem of minimizing the total completion time in $O\left(n^{K+1} \log n\right)$ time each. We notice, however, that the paper underestimates the running time needed to solve the problem of minimizing the total completion time. This is because the running time needed to compute the values of the positional weights has been ignored. It can be easily verified that this running time is $O\left(n^2\right)$, for the problem of minimizing the total completion time. Thus, problem $1\left|p_j^{[x]}(\tau; r) = \left(p_j + a\tau\right)r^b, RMP(K), \Delta\right| \sum C_j$ cannot be solved in less than $O\left(n^{K+2}\right)$ time.

Yang and Yang (2010) study three reduced problems with pure positional and time-dependent effects. First, they solve problem $1\left|p_j^{[x]}(r) = p_j r^{b_j}, RMP(K), \Delta(\tau)\right| \sum C_j$, with a job-dependent polynomial deterioration effect, i.e., with $g^{[x]}(r) = r^{b_j}$, $b_j > 0$, $j \in N$, along with $K$ identical RMPs of variable duration. They reduce the problem to solving a series of linear assignment problems and present a solution approach that requires $O\left(n^{K+3}\right)$ time. Second, they solve a job-independent version of the first problem, i.e., with $g^{[x]}(r) = r^b$, $b > 0$, and present a solution approach that reduces the problem to solving a series of linear assignment problems with a product matrix and requires $O\left(n^{K+1} \log n\right)$ time to minimize the total completion time. Third, they solve problem $1\left|p_j^{[x]}(\tau) = p_j + a\tau, RMP(K), \Delta(\tau)\right| \sum C_j$, with a linear time-dependent deterioration effect, along with $K$ RMPs of variable duration. This problem reduces to a problem to solving a series of linear assignment problems with a product matrix and requires $O\left(n^{K+1}\log n\right)$ time to minimize the total completion time.

Ji and Cheng (2010) study a relatively more complicated problem with a job-dependent group-dependent polynomial learning effect, so that $g_j^{[x]}(r) = \lambda_j^{[x]} \left(\sum_{y=1}^{x-1} n^{[y]} + r\right)^{a_j}, a_j < 0, j \in N$, where $\lambda_j^{[x]}, 0 < \lambda_j^{[x]} \le 1, 1 \le x \le k, j \in N$, represents a job-dependent group-dependent learning factor with $\lambda_j^{[1]} = 1, j \in N$. This is the first paper of its kind, which combines a learning effect with a rate-modifying activity. The RMPs are aimed at further enhancing the learning rate of the operator. Notice that the positional factors in this model are dependent on the number of jobs scheduled in the previous groups. This indicates a continuous learning process across groups. The duration of each RMP is a constant. Ji and Cheng (2010) reduce the problem to solving a series of linear assignment problems and present a solution approach that requires $O\left(n^{K+3}\right)$ time. They further extend this model to a parallel machine environment and show that an optimal solution can be found in $O\left(n^{m+K+2}\right)$ time, where $m$ is the number of machines.

The work by Lee and Wu (2009) on positionally dependent setup times for models that arise in group technology scheduling forms a background for formula (18.3), which allows the duration of an RMP to be dependent on its start time, as well as on its position in the processing sequence.

## References

Ji M, Cheng TCE (2010) Scheduling with job-dependent learning effects and multiple rate-modifying activities. Inf Process Lett 110:460–463

Lee WC, Wu CC (2009) A note on single-machine group scheduling problems with position-based learning effect. Appl Math Model 34:2159–2163

Rustogi K, Strusevich VA (2014) Combining time and position dependent effects on a single machine subject to rate-modifying activities. Omega 42:166–178

Yang SJ (2010) Single-machine scheduling problems with both start-time dependent learning and position dependent aging effects under deteriorating maintenance consideration. Appl Math Comput 217:3321–3329

Yang SJ (2012) Single-machine scheduling problems simultaneously with deterioration and learning effects under deteriorating multi-maintenance activities consideration. Comput Ind Eng 62:271–275

Yang SJ, Yang DL (2010) Minimising the total completion time in single-machine scheduling with ageing/deteriorating effects and deteriorating maintenance activities. Comput Math Appl 60:2161–2169

# Chapter 19
# Scheduling with Maintenance and Linear Cumulative Effects

In this chapter, we study single machine scheduling problems, provided that the actual processing times of the jobs are subject to a cumulative deterioration effect and a single rate-modifying maintenance period is inserted into a schedule.

For a job $j \in N = \{1, 2, \dots, n\}$, its normal processing time $p_j$ is given. Suppose that the jobs are processed on a single machine in accordance with a permutation $\pi = (\pi(1), \dots, \pi(n))$. The most general model studied in this chapter, a job $j \in N$, is additionally associated with two parameters, $b_j$ and $q_j > 0$. The actual processing time of job $j$ scheduled in the $r$th position of permutation $\pi$ is defined by

$$p_j(r) = p_j\left(1 + b_j \sum_{h=1}^{r-1} q_{\pi(h)}\right), \tag{19.1}$$

where $b_j > 0$ can be understood as a job-dependent deterioration rate that reflects how sensitive a particular job is to the accumulated $q$-value $Q_r$ of previously scheduled jobs, i.e.,

$$Q_r = \sum_{h=1}^{r-1} q_{\pi(h)}. \tag{19.2}$$

Problem $1\left|p_j(r) = p_j\left(1 + b_j Q_r\right)\right|C_{\max}$ of minimizing the makespan under the effect (19.1) is studied in Chap. 10, where it is proved that an optimal permutation can be found by scheduling the jobs in non-increasing order of the ratios $\frac{b_j p_j}{q_j}$, $j \in N$ (see Theorem 10.7). This is why throughout this chapter we assume the jobs are renumbered in such a way that

$$\frac{p_1 b_1}{q_1} \geq \frac{p_2 b_2}{q_2} \geq \cdots \geq \frac{p_n b_n}{w_n}. \tag{19.3}$$

As a special case of effect (19.1), we also consider a less general linear effect given by

$$p_j(r) = p_j\left(1 + b \sum_{h=1}^{r-1} p_{\pi(h)}\right). \tag{19.4}$$

Here, the deterioration rate $b$ is the same for all jobs, and the actual processing time explicitly depends on $P_r$, the accumulated normal processing times of previously scheduled jobs, i.e.,

$$P_r = \sum_{h=1}^{r-1} p_{\pi(h)}. \tag{19.5}$$

Problem $1\left|p_j(r) = p_j(1 + bP_r)\right|C_{\max}$ under the effect (19.4) and its generalizations are also studied in Chap. 10. In particular, for problem $1\left|p_j(r) = p_j(1 + bP_r)\right|C_{\max}$ it is proved that any permutation of jobs is optimal (see Lemma 10.2). Notice that for problem $1\left|p_j(r) = p_j(1 + bP_r)\right|C_{\max}$ all ratios $\frac{b_j p_j}{q_j}$ in (19.3) are equal to $b$, since $q_j = p_j$ and $b_j = b$, $j \in N$.

A single maintenance period (MP) can be introduced into a schedule to prevent uncontrolled deterioration. As usual, during an MP no processing takes place, and after the MP the processing facility is in better processing conditions. The duration of an MP is either a constant or depends on its start time $\tau$. The most general problem studied in this chapter is denoted by $1\left|p_j(r) = p_j\left(1 + b_j Q_r\right), MP(\lambda)\right|C_{\max}$, where $MP(\lambda)$ means that the duration of the MP is $\lambda\tau + \eta$, where $\eta$ and $\lambda$ are given constants; in particular, $MP(0)$ corresponds to the MP of constant duration $\eta$. Unless stated otherwise, we additionally assume that in problem $1\left|p_j(r) = p_j\left(1 + b_j Q_r\right),\right.$ $MP(\lambda)|C_{\max}$ the MP does not fully restore the initial processing conditions. More precisely, it is assumed that for a job $j \in N$ scheduled after the MP, the normal processing time changes from $p_j$ to $\sigma p_j$, where $\sigma \geq 1$ is a given constant.

The results presented in this section are fully polynomial-time approximation schemes (FPTAS) for the scheduling problems with a cumulative deterioration and a single MP. The approximation schemes are developed by adapting schemes known for Boolean programming problems such as the problem of minimizing a half-product and the subset-sum problem. The reader is advised to refer to Chap. 4, which contains a detailed discussion of related issues.

This chapter is organized as follows. In Sect. 19.1, we demonstrate that problem $1\left|p_j(r) = p_j\left(1 + b_j Q_r\right), MP(\lambda)\right|C_{\max}$ reduces to Problem HPAdd introduced in Sect. 4.3, i.e., to the problem of minimizing a Boolean quadratic function known as the half-product. The simplest version of the original problem, namely problem $1\left|p_j(r) = p_j(1 + bP_r), MP(0)\right|C_{\max}$ with $\sigma = 1$ is shown to reduce to the subset-sum problem, and in Sect. 19.2, we show how to adapt an FPTAS known for the subset-sum problem to finding an $\varepsilon$ -approximate solution to this scheduling problem in $O(n/\varepsilon)$ time. Finally, in Sect. 19.3, we use Theorem 4.8 on the existence of an FPTAS for Problem HPAdd to develop an FPTAS for problem $1\left|p_j(r) = p_j\left(1 + b_j Q_r\right), MP(\lambda)\right|C_{\max}$ that requires $O\left(n^2/\varepsilon\right)$ time.

## 19.1 Half-Product Reformulations

In this section, we establish links between problem $1\left|p_j(r) = p_j\left(1 + b_j Q_r\right),\right.$ $MP\,(\lambda)|C_{\max}$ and its variants to Boolean programming problems. This will help us to establish the complexity of the simplest problem of this range, i.e., problem $1\left|p_j(r) = p_j(1 + bP_r), MP(0)\right|C_{\max}$ with the deterioration effect (19.4), an MP of a constant duration and $\sigma = 1$.

Recall from Sect. 4.3 that the half-product is the function

$$H(\mathbf{x}) = \sum_{1\le i<j\le n}^{n} \alpha_i\beta_j x_i x_j - \sum_{j=1}^{n} \gamma_j x_j, \tag{19.6}$$

defined for a vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ with $n$ Boolean components; the coefficients $\alpha_j$ and $\beta_j$ are non negative integers, while $\gamma_j$ is an integer that can be either negative or positive.

For problem $1\left|p_j(r) = p_j\left(1 + b_j Q_r\right), MP(\lambda)\right|C_{\max}$, in a schedule with a single MP the jobs are split into two groups: group 1 consists of the jobs scheduled before the maintenance and group 2 contains all other jobs. Let $N^{[i]}$ be the set of jobs in group $i$ and $\left|N^{[i]}\right| = n^{[i]}$, for $i \in \{1, 2\}$. Recall that due to Theorem 10.7, we may only consider schedules in which the jobs of each group are sequenced in the order of their numbering given by (19.3).

Introduce a Boolean variable $x_j$ in such a way that

$$x_j = \begin{cases} 1, & \text{if job } j \text{ is scheduled in the first group} \\ 0, & \text{otherwise} \end{cases}$$

for each job $j,\ 1 \le j \le n$.

Taking the jobs in order of their numbering, if job $j \in N$ is scheduled in the first group then it completes at time

$$C_j = p_j x_j\left(1 + b_j \sum_{i=1}^{j-1} q_i x_i\right),$$

so that the MP starts at time

$$\tau = \sum_{j=1}^{n} p_j x_j\left(1 + b_j \sum_{i=1}^{j-1} q_i x_i\right).$$

If job $j$ is scheduled in the second group, then its completion time is given by

$$\begin{aligned}C_j &= \tau + (\lambda\tau + \eta) + \sigma p_j(1 - x_j)\left(1 + b_j \sum_{i=1}^{j-1} q_i(1 - x_i)\right)\\ &= (\lambda + 1)\sum_{j=1}^{n} p_j x_j \left(1 + b_j \sum_{i=1}^{j-1} q_i x_i\right) + \sigma p_j(1 - x_j)\left(1 + b_j \sum_{i=1}^{j-1} q_i(1 - x_i)\right) + \eta.\end{aligned}$$

This implies that in order to solve problem $1\left|p_j(r) = p_j\left(1 + b_j Q_r\right), MP(\lambda)\right| C_{\max}$, we need to minimize the function

$$\begin{aligned}Y(\mathbf{x}) &= (\lambda + 1)\sum_{j=1}^{n} p_j x_j \left(1 + b_j \sum_{i=1}^{j-1} q_i x_i\right) + \sigma \sum_{j=1}^{n} p_j(1 - x_j)\left(1 + b_j \sum_{i=1}^{j-1} q_i(1 - x_i)\right) + \eta\\ &= \sum_{j=1}^{n}(\lambda + 1) b_j p_j x_j \left(\sum_{i=1}^{j-1} q_i x_i\right) + \sum_{j=1}^{n} \sigma b_j p_j (1 - x_j)\left(\sum_{i=1}^{j-1} q_i(1 - x_i)\right)\\ &\quad + (\lambda + 1)\sum_{j=1}^{n} p_j x_j + \sigma \sum_{j=1}^{n} p_j(1 - x_j) + \eta.\end{aligned}$$

Define $w_j := b_j p_j$, $j \in N$, and rewrite $Y(\mathbf{x})$ as

$$\begin{aligned}Y(\mathbf{x}) = &\sum_{1\le i<j\le n} (\lambda + 1) q_i w_j x_i x_j + \sum_{1\le i<j\le n} \sigma q_i w_j (1 - x_i)(1 - x_j)\\ &+ (\lambda + 1)\sum_{j=1}^{n} p_j x_j + \sigma \sum_{j=1}^{n} p_j(1 - x_j) + \eta.\end{aligned} \tag{19.7}$$

This function reminds a so-called symmetric quadratic function (4.31) with non-negative coefficients reproduced below as

$$Z(\mathbf{x}) = \sum_{1\le i<j\le n} \alpha_i \beta_j x_i x_j + \sum_{1\le i<j\le n} \alpha_i \beta_j (1 - x_i)(1 - x_j) + \sum_{j=1}^{n} \mu_j x_j + \sum_{j=1}^{n} \nu_j (1 - x_j) + K, \tag{19.8}$$

which can be seen if we define

$$\alpha_j = q_j,\ \beta_j = w_j,\ \mu_j = (\lambda + 1) p_j,\ \nu_j = \sigma p_j,\ j \in N;\ K = \eta.$$

In fact, (19.7) is more general than (19.8). Indeed, the quadratic terms in (19.7) are not entirely symmetric, since they contain additional factors of $\lambda + 1$ and $\sigma$, respectively.

This loss of symmetry does not prevent us from reformulating (19.7) as a special case of the half-product function (19.6), as demonstrated below.

Since

$$\sum_{1\le i<j\le n} q_i w_j (1-x_i)(1-x_j) = \sum_{1\le i<j\le n} q_i w_j x_i x_j + \sum_{1\le i<j\le n} q_i w_j - \sum_{j=1}^{n}\left( w_j\left(\sum_{i=1}^{j-1} q_i\right) + q_j\left(\sum_{i=j+1}^{n} w_i\right)\right) x_j,$$

function $Y(\mathbf{x})$ derived above may be written as

$$\begin{aligned} Y(\mathbf{x}) = & \sum_{1\le i<j\le n} (\lambda+\sigma+1) q_i w_j x_i x_j \\ & + \sum_{j=1}^{n}\left( (\lambda-\sigma+1)p_j - \sigma\left( w_j\left(\sum_{i=1}^{j-1} q_i\right) + q_j\left(\sum_{i=j+1}^{n} w_i\right)\right)\right) x_j \\ & + \left(\eta + \sigma\left(\sum_{j=1}^{n} p_j + \sum_{1\le i<j\le n} q_i w_j\right)\right). \end{aligned} \tag{19.9}$$

The variable terms of (19.9) can be written in the form (19.6) of

$$\begin{aligned} \alpha_j &= (\lambda+\sigma+1) w_j,\ \beta_j = q_j, \\ \gamma_j &= (\lambda-\sigma+1)p_j - \sigma\left( w_j\left(\sum_{i=1}^{j-1} q_i\right) + q_j\left(\sum_{i=j+1}^{n} w_i\right)\right),\ j \in N, \end{aligned}$$

so that function $Y(\mathbf{x})$ is a special case of function

$$F(\mathbf{x}) = H(\mathbf{x}) + K. \tag{19.10}$$

Recall that in Sect. 4 the Boolean programming problem of minimizing a function of the form (19.10) is called Problem HPAdd.

Thus, we obtain the following statement.

**Theorem 19.1** *Problem* $1\left|p_j(r) = p_j\left(1 + b_j Q_r\right), MP(\lambda)\right|C_{\max}$ *reduces to minimizing function (19.7) over the set of all n-dimensional Boolean vectors, which is a special case of Problem HPAdd.*

Consider the simplest form of problem $1\left|p_j(r) = p_j\left(1 + b_j Q_r\right), MP(\lambda)\right|C_{\max}$ with $q_j = p_j$, $b_j = b$, $j \in N$, $\lambda = 0$ and $\sigma = 1$, i.e., problem $1\left|p_j(r) = p_j(1 + bP_r), MP(0)\right|C_{\max}$. Notice that for this problem the assumption $\sigma = 1$ is made, which implies that the processing conditions after the maintenance period are fully restored.

It follows directly from Theorem 19.1 and they made simplifying assumptions that problem $1\left|p_j(r) = p_j(1 + bP_r), MP(0)\right|C_{\max}$ reduces to minimizing function

$$Y_0(\mathbf{x}) = \sum_{j=1}^{n} p_j x_j \left(1 + b\sum_{i=1}^{j-1} p_i x_i\right) + \sum_{j=1}^{n} p_j(1 - x_j)\left(1 + b\sum_{i=1}^{j-1} p_i(1 - x_i)\right) + \eta. \tag{19.11}$$

Notice that the first quadratic term of function $Y_0(\mathbf{x})$ defined by (19.11) can be rewritten as

$$\begin{aligned}
&\sum_{j=1}^{n} p_j x_j \left(1 + b\sum_{i=1}^{j-1} p_i x_i\right)\\
&\quad = \sum_{j=1}^{n} p_j x_j + b\sum_{j=1}^{n} p_j x_j \sum_{i=1}^{j-1} p_i x_i\\
&\quad = \sum_{j=1}^{n} p_j x_j + \frac{b}{2}\left(\sum_{j=1}^{n} p_j^2 x_j^2 + 2\sum_{j=1}^{n} p_j x_j \sum_{i=1}^{j-1} p_i x_i - \sum_{j=1}^{n} p_j^2 x_j^2\right)\\
&\quad = \sum_{j=1}^{n} p_j x_j + \frac{b}{2}\left(\sum_{j=1}^{n} p_j x_j\right)^2 - \frac{b}{2}\sum_{j=1}^{n} p_j^2 x_j^2.
\end{aligned}$$

Since $x_j \in \{0, 1\}$, we deduce that

$$\sum_{j=1}^{n} p_j x_j \left(1 + b\sum_{i=1}^{j-1} p_i x_i\right) = \frac{b}{2}\left(\sum_{j=1}^{n} p_j x_j\right)^2 + \sum_{j=1}^{n} p_j x_j - \frac{b}{2}\sum_{j=1}^{n} p_j^2 x_j.$$

Recall that in a schedule associated with a Boolean vector $\mathbf{x}$, the jobs of the first group form the set $N^{[1]} = \left\{j \in N | x_j = 1\right\}$, so that we may further rewrite

$$\sum_{j=1}^{n} p_j x_j \left(1 + b\sum_{i=1}^{j-1} p_i x_i\right) = \frac{b}{2}\left(\sum_{j \in N^{[1]}} p_j\right)^2 + \sum_{j \in N^{[1]}} p_j - \frac{b}{2}\sum_{j \in N^{[1]}} p_j^2,$$

which, using our standard notation $p(Q) = \sum_{j \in Q} p_j$, can be expressed as

$$\sum_{j=1}^{n} p_j x_j \left(1 + b\sum_{i=1}^{j-1} p_i x_i\right) = \frac{b}{2} p\left(N^{[1]}\right)^2 + p\left(N^{[1]}\right) - \frac{b}{2}\sum_{j \in N^{[1]}} p_j^2.$$

In a similar way, the second quadratic term of function $Y_0(\mathbf{x})$ defined by (19.11) can be expressed in terms of the set $N^{[2]} := N \backslash N^{[1]}$ of jobs of the second group as

$$\sum_{j=1}^{n} p_j(1-x_j)\left(1+b\sum_{i=1}^{j-1} p_i(1-x_i)\right) = \frac{b}{2}p\big(N^{[2]}\big)^2 + p\big(N^{[2]}\big) - \frac{b}{2}\sum_{j\in N^{[2]}} p_j^2.$$

Thus, for problem $1\big|p_j(r) = p_j(1+bP_r), MP(0)\big|C_{\max}$ with $\sigma = 1$ a schedule $S$ is determined by splitting the set of jobs into two subsets $N^{[1]}$ and $N^{[2]}$, so that the objective function can be written as

$$C_{\max}(S) = \frac{b}{2}\Big(p\big(N^{[1]}\big)^2 + p\big(N^{[2]}\big)^2\Big) + p(N) - \frac{b}{2}\sum_{j\in N} p_j^2 + \eta. \tag{19.12}$$

This implies that in order to achieve the smallest makespan in problem $1\big|p_j(r) = p_j(1+bP_r), MP(0)\big|C_{\max}$ with $\sigma = 1$, we only need to partition the set $N$ of jobs into two subsets, $N^{[1]}$ and $N^{[2]}$, in such a way that the value $p\big(N^{[1]}\big)^2 + p\big(N^{[2]}\big)^2$ is as small as possible.

The obtained representation allows us to give an easy proof that problem $1\big|p_j(r) = p_j(1+bP_r), MP(0)\big|C_{\max}$ is NP-hard in the ordinary sense, even if $\sigma = b = 1$. In the proof of the NP-hardness given below, the following well-known NP-complete problem is used for reduction (see Sect. 1.3.2).

Partition: Given positive integers $e_1, \ldots, e_r$ and the index set $R = \{1, \ldots, r\}$ such that $e(R) = \sum_{i\in R} e_i = 2R$, is it possible to partition set $R$ into disjoint subsets $R_1$ and $R_2$ such that $e(R_1) = \sum_{i\in R_1} e_i = E$ and $e(R_2) = \sum_{i\in R_2} e_i = E$?

**Theorem 19.2** *Problem $1\big|p_j(r) = p_j(1+bP_r), MP(0)\big|C_{\max}$ is NP-hard in the ordinary sense, even if $\sigma = b = 1$.*

*Proof* Given an instance of Partition, define the instance of problem $1\big|p_j(r) = p_j(1+bP_r), MP(0)\big|C_{\max}$ with

$$N = R;\ p_j = e_j,\ j \in N;\ \sigma = b = 1;\ \eta = E^2 - 1.$$

We show that Partition has a solution if and only if in the constructed problem there exists a schedule $S_0$ such that $C_{\max}(S_0) \le y := 2E^2 + 2E - \frac{1}{2}\sum_{j\in R} e_j^2 - 1$.

Suppose that Partition has a solution represented by the sets $R_1$ and $R_2$. Then schedule $S_0$ with $C_{\max}(S_0) = y$ exists and can be found as follows. Define $N^{[i]} := R_i$, $i \in \{1, 2\}$ and let in $S$ the jobs of set $N^{[1]}$ form the first group, while the jobs of set $N^{[2]}$ form the second group. It follows from (19.12) that

$$C_{\max}(S_0) = \frac{1}{2}\left(E^2 + E^2\right) + 2E - \frac{1}{2}\sum_{j\in R} e_j^2 + \left(E^2 - 1\right)$$
$$= 2E^2 + 2E - \frac{1}{2}\sum_{j\in R} e_j^2 - 1 = y,$$

as required.

Suppose now that a schedule $S_0$ such that $C_{\max}(S_0) \leq y$ exists and $N^{[i]}$ for $i \in \{1, 2\}$ are the sets of jobs processed in the first and the second groups, respectively. Thus, due to (19.12),

$$C_{\max}(S_0) = \frac{1}{2}\left(p\left(N^{[1]}\right)^2 + p\left(N^{[2]}\right)^2\right) + p(N) - \frac{1}{2}\sum_{j\in N} p_j^2 + \left(E^2 - 1\right)$$
$$= \frac{1}{2}\left(p\left(N^{[1]}\right)^2 + p\left(N^{[2]}\right)^2\right) + 2E - \frac{1}{2}\sum_{j\in N} p_j^2 + \left(E^2 - 1\right) \leq y$$

Denote

$$E_1 := p\left(N^{[1]}\right),\ E_2 := p\left(N^{[2]}\right).$$

For schedule $S_0$, we must have

$$\frac{1}{2}\left(E_1^2 + E_2^2\right) \leq E^2.$$

However, the minimum of the expression $\frac{1}{2}\left(E_1^2 + E_2^2\right)$ under the condition $E_1 + E_2 = 2E$ is equal to $E^2$ and is achieved for $E_1 = E_2 = E$. Thus, if we set $R_i = N^{[i]}$, $i \in \{1, 2, \}$, we obtain a solution to PARTITION. $\square$

It is clear that problem $1\left|p_j(r) = p_j(1 + bP_r), MP(0)\right|C_{\max}$ with $\sigma = b = 1$ is the simplest version of problem $1\left|p_j(r) = p_j\left(1 + b_j Q_r\right), MP(\lambda)\right|C_{\max}$. Thus, the best possible approximation result that can be derived for either problem is an FPTAS. In the subsequent sections, we develop such approximation schemes.

## 19.2 Constant Maintenance: FPTAS via Subset-Sum

In this section, we present an FPTAS for problem $1\left|p_j(r) = p_j(1 + bP_r), MP(0)\right|$ $C_{\max}$ with $\sigma = 1$. Although the problem has been reduced to quadratic Boolean programming, we demonstrate that the problem in fact is related to the subset-sum problem, with a linear objective function. The latter problem admits an FPTAS (see the discussion in Sect. 4.2). We demonstrate that an FPTAS available for the subset-sum problem can be adapted to handle problem $1\left|p_j(r) = p_j(1 + bP_r), MP(0)\right|C_{\max}$.

First, we show that the smallest value of $p(N^{[1]})^2 + p(N^{[2]})^2$ can be achieved if the values $p(N^{[1]})$ and $p(N^{[2]})$ are as close as possible. The latter problem can be formulated as the well-known subset-sum problem discussed in Sect. 4.2. The generic formulation of the subset-sum problem is

$$\begin{aligned} &\text{maximize } \sum_{j=1}^{n} \alpha_j x_j \\ &\text{subject to } \sum_{j=1}^{n} \alpha_j x_j \le A \\ &\qquad\qquad x_j \in \{0, 1\}, \ j = 1, 2, \ldots, n; \end{aligned} \tag{19.13}$$

however, for the purposes of this section, we will use the formulation written in terms of normal processing times of problem $1\left|p_j(r) = p_j(1 + bP_r), MP(0)\right|C_{\max}$:

$$\begin{aligned} &\text{maximize } \sum_{j \in N} p_j x_j \\ &\text{subject to } \sum_{j \in N} p_j x_j \le T \\ &\qquad\qquad x_j \in \{0, 1\}, \ j \in N, \end{aligned} \tag{19.14}$$

with

$$T =: p(N)/2. \tag{19.15}$$

Assume that a schedule $S^*$ that is optimal for problem $1|p_j(r) = p_j(1 + bP_r), MP(0)|C_{\max}$ is associated with the partition of the jobs into two groups $H^{[1]}$ and $H^{[2]}$, processed before and after the MP, respectively.

**Lemma 19.1** *Suppose that* $x_j^* \in \{0, 1\}$, $j \in N$, *are the optimal values of the decision variables for the problem (19.14). Define* $N_1^* := \left\{ j \in N | x_j^* = 1 \right\}$ *and* $N_2^* = N \backslash N_1^*$. *Then for problem* $1\left|p_j(r) = p_j(1 + bP_r), MP(0)\right|C_{\max}$ *with* $\sigma = 1$ *there exists an optimal schedule* $S^*$ *in which the jobs of set* $H^{[1]} = N_1^*$ *are scheduled in one group and the jobs of set* $H^{[2]} = N_2^*$ *are scheduled in the other group.*

*Proof* Without loss of generality, we may assume that $p(N_1^*) \le p(N_2^*)$, so that there exists a non negative $\delta$ such that

$$p(N_1^*) = T - \delta, \ p(N_2^*) = T + \delta.$$

Suppose that the sets $N_1^*$ and $N_2^*$ defined by an optimal solution to the subset-sum problem (19.14) do not define an optimal schedule, i.e., in schedule $S^*$ the first group and the second group consists of the jobs of set $H^{[1]} = N_1'$ and $H^{[2]} = N_2'$, respectively, such that

$$p(N_1')^2 + p(N_2')^2 < p(N_1^*)^2 + p(N_2^*)^2.$$

Without loss of generality, we may assume that $p(N_1') \le p(N_2')$. Since the sets $N_1'$ and $N_2'$ are not defined by an optimal solution to (19.14), we deduce that there exists a positive $\nu$ such that

$$p(N_1') = p(N_1^*) - \nu, \; p(N_2') = p(N_2^*) + \nu.$$

If we set $\varepsilon = \delta + \nu > \delta$, we obtain

$$p(N_1') = T - \varepsilon, \; p(N_2') = T + \varepsilon.$$

On the other hand,

$$p(N_1')^2 + p(N_2')^2 = (T - \varepsilon)^2 + (T + \varepsilon)^2 = 2T^2 + 2\varepsilon^2;$$
$$p(N_1^*)^2 + p(N_2^*)^2 = (T - \delta)^2 + (T + \delta)^2 = 2T^2 + 2\delta^2 < p(N_1')^2 + p(N_2')^2.$$

The obtained contradiction proves the lemma. $\square$

Lemma 19.1 implies that for solving problem $1\left|p_j(r) = p_j(1 + bP_r), MP(0)\right| C_{\max}$ with $\sigma = 1$ it suffices to find a solution to the subset-sum problem (19.14). The latter problem being a special case of the linear knapsack problem admits an FPTAS (see the discussion in Sect. 4.2), in particular Theorem 4.5. For convenience, we reformulate that theorem below.

**Theorem 19.3** *Problem SSP of the form (19.14) admits an FPTAS that for a given positive $\varepsilon$, either finds an optimal solution $x_j^* \in \{0, 1\}$, $j \in N$, such that*

$$\sum_{j \in N} p_j x_j^* < (1 - \varepsilon)T$$

*or finds an approximate solution $x_j^\varepsilon \in \{0, 1\}$, $j \in N$, such that*

$$(1 - \varepsilon)T \le \sum_{j \in N} p_j x_j^\varepsilon \le T.$$

*Such an FPTAS requires $O\left(\min\left\{n/\varepsilon, n + \frac{1}{\varepsilon^2}\log\left(\frac{1}{\varepsilon}\right)\right\}\right)$ time and $O\left(n + \frac{1}{\varepsilon}\right)$ space.*

Below we explain how an FPTAS that exists for the subset-sum problem can be adapted for solving problem $1\left|p_j(r) = p_j(1 + bP_r), MP(0)\right|C_{\max}$ with $\sigma = 1$. It can be seen from (19.12) that the objective function in the scheduling problem under consideration consists of a variable part and a constant part. Due to Lemma 19.1, an FPTAS available for the subset-sum problem can be used to approximate the variable part of the objective function. However, since the variable part $p\left(N^{[1]}\right)^2 + p\left(N^{[2]}\right)^2$ is positive and the constant part $p(N) - \frac{b}{2}\sum_{j \in N} p_j^2 + \eta$ can be negative, a solution delivered by an FPTAS for minimizing the variable part need not be to an $\varepsilon$-approximate solution for the overall function.

To illustrate this, consider a function of the form

$$F(\mathbf{x}) = G(\mathbf{x}) + K,$$

where $G(\mathbf{x})$ represents a variable part of the overall function $F(\mathbf{x})$ to be minimized, and $K$ is a constant. If $\mathbf{x}^*$ minimizes the function $G(\mathbf{x})$, then it will obviously minimize the function $F(\mathbf{x})$ as well. Assume that $G(\mathbf{x}^*) \geq 0$ and suppose that for minimizing function $G(\mathbf{x})$, an FPTAS is available that delivers a solution $\mathbf{x}^\varepsilon$, such that $G(\mathbf{x}^\varepsilon) - G(\mathbf{x}^*) \leq \varepsilon G(\mathbf{x}^*)$.

For $\mathbf{x}^\varepsilon$ to be accepted as an $\varepsilon$-approximate solution for minimizing the function $F(\mathbf{x})$, we must establish the inequality

$$F(\mathbf{x}^\varepsilon) \leq (1+\varepsilon)F(\mathbf{x}^*). \tag{19.16}$$

For a solution $\mathbf{x}^\varepsilon$ found by an FPTAS for minimizing $G(\mathbf{x})$, we will have

$$F(\mathbf{x}^\varepsilon) = G(\mathbf{x}^\varepsilon)+K \leq (1+\varepsilon)G(\mathbf{x}^*)+K = F(\mathbf{x}^*)+\varepsilon G(\mathbf{x}^*) = (1+\varepsilon)F(\mathbf{x}^*)-\varepsilon K.$$

If $K \geq 0$, the inequality (19.16) holds; however, if $K < 0$, there is no direct evidence that (19.16) will hold, and further analysis must be performed. Recall that we have faced a similar situation in Sect. 4.3, where for Problem HPAdd the variable part of the objective function is negative and an additive constant is positive, and therefore converting an FPTAS that exists for Problem HP of minimizing a half-product function with no additive constant to an FPTAS for Problem HPAdd is non-trivial.

In the remainder of this section, we prove that in order to obtain an FPTAS with the accuracy $\varepsilon$ for problem $1\left|p_j(r) = p_j(1+bP_r), MP(0)\right|C_{\max}$ with $\sigma = 1$, we may apply an FPTAS with $\varepsilon_0 = \frac{\varepsilon}{\varepsilon+1}$ to problem (19.14).

**Algorithm EpsCumuMP0**

INPUT: An instance of $1\left|p_j(r) = p_j(1+bP_r), MP(0)\right|C_{\max}$ with $\sigma = 1$ and an $\varepsilon > 0$

OUTPUT: A schedule $S^\varepsilon$ such that $C_{\max}(S^\varepsilon) \leq (1+\varepsilon)C_{\max}(S^*)$

**Step 1**. For a given $\varepsilon > 0$ define $\varepsilon_0 := \frac{\varepsilon}{\varepsilon+1}$.
**Step 2**. With the defined $\varepsilon_0$, run an FPTAS for problem (19.14) to find the values $x_j^\varepsilon \in \{0,1\}$, $j \in N$. Define $N_1^\varepsilon := \left\{j \in N | x_j^\varepsilon = 1\right\}$ and $N_2^\varepsilon := N \backslash N_1^\varepsilon$.
**Step 3**. Output schedule $S^\varepsilon$ for the original scheduling problem, in which the jobs of set $N^{[1]} := N_1^\varepsilon$ are assigned to one group and sequenced before the maintenance and the jobs of set $N^{[2]} := N_2^\varepsilon$ are assigned to the other group to be scheduled after the maintenance. Stop.

We prove that Algorithm EpsCumuMP0 gives an appropriate treatment to the negative constant and therefore allows us to adapt an FPTAS that is guaranteed by Theorem 19.3 to deliver an $\varepsilon$-approximate solution for minimizing the makespan in the scheduling problem under consideration.

**Theorem 19.4** *For problem* $1\left|p_j(r) = p_j(1 + bP_r), MP(0)\right|C_{\max}$ *with* $\sigma = 1$, *Algorithm EpsCumuMP0 is an FPTAS that runs in* $O\left(\min\left\{n/\varepsilon, n + \left(1 + \frac{1}{\varepsilon}\right)^2 \log\left(1 + \frac{1}{\varepsilon}\right)\right\}\right)$ *time.*

*Proof* Using an FPTAS from Theorem 19.3 with $\varepsilon_0 = \frac{\varepsilon}{\varepsilon+1}$, we observe that $O(n/\varepsilon_0) = O\left(n\frac{\varepsilon+1}{\varepsilon}\right) = O(n/\varepsilon)$ and $\frac{1}{\varepsilon_0^2}\log\left(\frac{1}{\varepsilon_0}\right) = \left(1 + \frac{1}{\varepsilon}\right)^2 \log\left(1 + \frac{1}{\varepsilon}\right)$, so that the required running time is achieved. To complete the proof, we need to prove that $C_{\max}(S^{\varepsilon}) \le (1 + \varepsilon)C_{\max}(S^*)$.

For our analysis, we will need a lower bound

$$C_{\max}(S^*) \ge bT^2 + p(N) - \frac{b}{2}\sum_{j \in N} p_j^2 + \eta, \tag{19.17}$$

where $T$ is defined by (19.15).

To see that (19.17) holds, apply (19.12) and observe that for any partition of set $N$ into subsets $N_1$ and $N_2$ the inequality $p(N_1)^2 + p(N_2)^2 \ge 2T^2$ holds. The latter fact is used in the proof of Theorem 19.2.

Due to Theorem 19.3, we only need to consider the case that the FPTAS used as a subroutine in Step 2 of Algorithm EpsCumuMP0 does not find an optimal solution to problem (19.14); otherwise, schedule $S^{\varepsilon}$ is optimal. Therefore, below we only look at the instances of problem $1\left|p_j(r) = p_j(1 + bP_r), MP(0)\right|C_{\max}$ for which $p_j \le T$, $j \in N$, since otherwise an optimal solution can be obtained by scheduling the largest job in one group and the remaining jobs in the other.

It follows from Theorem 19.3 that

$$(1 - \varepsilon_0)T \le p(N_1^{\varepsilon}) < T.$$

There exists a $\delta$, $\delta \le \varepsilon_0$, such that

$$\begin{aligned} (1 - \varepsilon_0)T \le p(N_1^{\varepsilon}) &= T(1 - \delta) < T; \\ p(N_2^{\varepsilon}) &= (1 + \delta)T. \end{aligned}$$

Applying (19.12) and (19.17), we have that

$$\begin{aligned} C_{\max}(S^{\varepsilon}) &= \frac{b}{2}\left(p(N_1^{\varepsilon})^2 + p(N_2^{\varepsilon})^2\right) + p(N) - \frac{b}{2}\sum_{j \in N} p_j^2 + \eta \\ &= b\left(T^2 + \delta^2 T^2\right) + p(N) - \frac{b}{2}\sum_{j \in N} p_j^2 + \eta \le C_{\max}(S^*) + b\delta^2 T^2. \end{aligned}$$

Below we demonstrate that $b\delta(1-\delta)T^2$ is a lower bound on the optimal makespan $C_{\max}(S^*)$. Consider the problem

$$\begin{aligned} \max \quad & \sum_{j \in N} p_j^2 \\ \text{subject to} \quad & \sum_{j \in N_1^\varepsilon} p_j = (1-\delta)T \\ & \sum_{j \in N_2^\varepsilon} p_j = (1+\delta)T \\ & 0 \le p_j \le T, \ j \in N. \end{aligned} \tag{19.18}$$

The problem of this structure is known to be solvable by the greedy algorithm, which in the case under consideration scans the values $p_j$ in any order and gives each of them the largest possible value. Thus, the greedy algorithm will find an optimal solution to (19.18) in which one of the $p_j$'s is equal to $T$, one to $(1-\delta)T$, and one to $\delta T$, while all others are equal to zero. We deduce that

$$\sum_{j \in N} p_j^2 \le (1-\delta)^2 T^2 + T^2 + (\delta T)^2 = 2T^2(\delta^2 - \delta + 1)$$

provides an upper bound on the sum of squares of the processing times for all instances of the original scheduling problem for which Step 2 of Algorithm EpsCumuMP0 delivers $p(N_1^\varepsilon) = T(1-\delta)$ and $p(N_2^\varepsilon) = (1+\delta)T$, including the instance under consideration.

Substituting this upper bound into (19.17), we derive a lower bound

$$C_{\max}(S^*) \ge b\big(\delta - \delta^2\big)T^2 + p(N) + \eta \ge b\delta(1-\delta)T^2.$$

This lower bound implies that $bT^2 \le \frac{C_{\max}(S^*)}{\delta(1-\delta)}$, so that

$$C_{\max}(S^\varepsilon) \le C_{\max}(S^*) + b\delta^2 T^2 \le \left(1 + \frac{\delta}{1-\delta}\right) C_{\max}(S^*).$$

Since $\frac{\delta}{1-\delta}$ increases and $\delta \le \varepsilon_0$, we have that

$$C_{\max}(S^\varepsilon) \le \left(1 + \frac{\varepsilon_0}{1-\varepsilon_0}\right) C_{\max}(S^*).$$

Thus, if we use an FPTAS for problem (19.14) applied with $\varepsilon_0 = \frac{\varepsilon}{\varepsilon+1}$, we obtain an FPTAS for our scheduling problem which has the accuracy $\varepsilon$. $\square$

## 19.3 The General Problem: FPTAS via Half-Product

We now turn to the general problem $1\left|p_j(r) = p_j\left(1 + b_j Q_r\right), MP(\lambda)\right|C_{\max}$. As shown in Sect. 19.1, this problem reduces to minimizing a quadratic function of the form (19.7), which is a special case of the objective function in Problem HPAdd.

Section 4.3 discusses several approaches to converting an FPTAS available for the problem of minimizing the half-product function (19.6) into an FPTAS for Problem HPAdd, in which the objective function (19.10) contains an additive constant. One of these approaches is addressed in Theorem 4.8, which for convenience is reproduced below.

**Theorem 19.5** *For Problem HPAdd of minimizing function (19.10), denote the lower and upper bounds on the value of $F(\mathbf{x}^*)$ by $LB$ and $UB$, respectively, i.e., $LB \le F(\mathbf{x}^*) \le UB$. If the ratio $UB/LB$ is bounded from above by some $\rho$, then there exists an algorithm that delivers a solution $\mathbf{x}^0$ such that $F(\mathbf{x}^0) - LB \le \varepsilon LB$ in $O(\rho n^2/\varepsilon)$ time.*

Section 4.3.1 presents an algorithm that is guaranteed by Theorem 19.5, namely Algorithm HPRhoFPTAS. If the value of $\rho$ is bounded from above by a constant, then Algorithm HPRhoFPTAS is an FPTAS that requires the best possible running time of $O\left(n^2/\varepsilon\right)$.

Section 4.4 describes an approach to developing an FPTAS for minimizing a symmetric quadratic function (19.8), provided that the function is convex. Problem $1\left|p_j(r) = p_j\left(1 + b_j Q_r\right), MP(\lambda)\right|C_{\max}$ reduces to minimizing a slightly more general and not entirely symmetric quadratic function (19.7); however, an FPTAS for this problem can be developed in a very similar way.

First, notice that the convexity of function (19.7) is guaranteed by the numbering (19.3) (see Theorem 4.9). Since the variable part of function $Y(\mathbf{x})$ defined by (19.7), or rather by (19.9), is a special form of the half-product, it follows that Theorem 4.11 holds, so that the continuous relaxation of the problem of minimizing function (19.9) can be solved in $O\left(n^2\right)$ time.

Let $\mathbf{x}^C = (x_1^C, \dots, x_n^C)$, $0 \le x_j^C \le 1$, be the corresponding solution vector of the continuous relaxation of the problem of minimizing a convex function $Y(\mathbf{x})$. Clearly $Y(\mathbf{x}^C) \le Y(\mathbf{x}^*)$, and we may set $LB := Y(\mathbf{x}^C)$, as a lower bound that is required by Theorem 19.5.

To obtain an upper bound $UB$, we perform a simple rounding of the fractional components of vector $\mathbf{x}^C$, as described below.

**Algorithm SQRound2**

**Step 1**. Given a vector $\mathbf{x}^C = (x_1^C, \dots, x_n^C)$, $0 \le x_j^C \le 1$, a solution to the continuous relaxation of the problem of minimizing a convex function $Y(\mathbf{x})$ of the form (19.9) determines the sets $I_1 = \left\{j \in N,\ x_j^C \le \frac{1}{2}\right\}$ and $I_2 = \left\{j \in N,\ x_j^C > \frac{1}{2}\right\}$ and finds vector $\mathbf{x}^H = (x_1^H, \dots, x_n^H)$ with components

$$x_j^H = \begin{cases} 0 \text{ if } j \in I_1 \\ 1 \text{ if } j \in I_2 \end{cases}.$$

**Step 2**. Output vector $\mathbf{x}^H = (x_1^H, \dots, x_n^H)$ as heuristic solution for the problem of minimizing function (19.9).

The running time of Algorithm SQRound2 is $O(n)$. Clearly, the inequalities $Y(\mathbf{x}^C) \le Y(\mathbf{x}^*) \le Y(\mathbf{x}^H)$ hold; i.e., we may take $Y(\mathbf{x}^H)$ as an upper bound $UB$ on the optimal value $Y(\mathbf{x}^*)$. We now estimate the ratio $UB/LB = Y(\mathbf{x}^H)/Y(\mathbf{x}^C)$.

**Lemma 19.2** *Let $\mathbf{x}^C$ be an optimal solution to the continuous relaxation of the problem of minimizing function $Y(\mathbf{x})$ of the form (19.9) and $\mathbf{x}^H$ be a vector found by Algorithm SQRound2. Then,*

$$\rho = \frac{Y(\mathbf{x}^H)}{Y(\mathbf{x}^C)} \le 4.$$

*Proof* For a vector $\mathbf{x}^C$, let $I_1$ and $I_2$ be the index sets found in Step 2 of Algorithm SQRound2. For a vector $\mathbf{x} = (x_1, \dots, x_n)$, where $0 \le x_j \le 1$, define

$$Y_1(\mathbf{x}) := (\lambda+1) \sum_{\substack{1 \le i < j \le n \\ i,j \in I_1}} q_i w_j x_i x_j + \sigma \sum_{\substack{1 \le i < j \le n \\ i,j \in I_1}} q_i w_j (1-x_i)\big(1-x_j\big);$$

$$Y_2(\mathbf{x}) := (\lambda+1) \sum_{\substack{1 \le i < j \le n \\ i \in I_1, j \in I_2}} q_i w_j x_i x_j + \sigma \sum_{\substack{1 \le i < j \le n \\ i \in I_1, j \in I_2}} q_i w_j (1-x_i)\big(1-x_j\big);$$

$$Y_3(\mathbf{x}) := (\lambda+1) \sum_{\substack{1 \le i < j \le n \\ i \in I_2, j \in I_1}} q_i w_j x_i x_j + \sigma \sum_{\substack{1 \le i < j \le n \\ i \in I_2, j \in I_1}} q_i w_j (1-x_i)\big(1-x_j\big);$$

$$Y_4(\mathbf{x}) := (\lambda+1) \sum_{\substack{1 \le i < j \le n \\ i,j \in I_2}} q_i w_j x_i x_j + \sigma \sum_{\substack{1 \le i < j \le n \\ i,j \in I_2}} q_i w_j (1-x_i)\big(1-x_j\big);$$

$$Y_5(\mathbf{x}) := (\lambda+1) \sum_{j \in I_1} p_j x_j + \sigma \sum_{j \in I_1} p_j \big(1-x_j\big);$$

$$Y_6(\mathbf{x}) := (\lambda+1) \sum_{j \in I_2} p_j x_j + \sigma \sum_{j \in I_2} p_j \big(1-x_j\big).$$

It follows from (19.7) that

$$Y(\mathbf{x}) = \sum_{k=1}^{6} Y_k(\mathbf{x}) + \eta.$$

By the rounding conditions in Step 2 of Algorithm SQRound2, we derive

$$Y_2(\mathbf{x}^H) = Y_3(\mathbf{x}^H) = 0,$$

while

$$
\begin{aligned}
&Y_1(\mathbf{x}^H) = \sigma \sum_{\substack{1\le i<j\le n\\ i,j\in I_1}} q_i w_j; && Y_1(\mathbf{x}^C) \ge \frac{\sigma}{4} \sum_{\substack{1\le i<j\le n\\ i,j\in I_1}} q_i w_j;\\
&Y_4(\mathbf{x}^H) = (\lambda+1) \sum_{\substack{1\le i<j\le n\\ i,j\in I_2}} q_i w_j; && Y_4(x^C) \ge \frac{(\lambda+1)}{4} \sum_{\substack{1\le i<j\le n\\ i,j\in I_2}} q_i w_j;\\
&Y_5(\mathbf{x}^H) = \sigma \sum_{j\in I_1} p_j; && Y_5(\mathbf{x}^C) \ge \frac{\sigma}{2} \sum_{j\in I_1} p_j;\\
&Y_6(\mathbf{x}^H) = (\lambda+1) \sum_{j\in I_2} p_j; && Y_6(\mathbf{x}^C) \ge \frac{\lambda+1}{2} \sum_{j\in I_2} p_j;
\end{aligned}
$$

Thus, we have that

$$
\begin{aligned}
Y(\mathbf{x}^H) &= \sum_{k=1}^{6} Y_k(\mathbf{x}^H) + \eta = Y_1(\mathbf{x}^H) + Y_4(\mathbf{x}^H) + Y_5(\mathbf{x}^H) + Y_6(\mathbf{x}^H) + \eta\\
&\le 4Y_1(\mathbf{x}^C) + 4Y_4(\mathbf{x}^C) + 2Y_5(\mathbf{x}^C) + 2Y_6(\mathbf{x}^C) + \eta\\
&\le 4\sum_{k=1}^{6} Y_k(\mathbf{x}^C) + 4\eta = 4Y(\mathbf{x}^C),
\end{aligned}
$$

as required. □

It follows immediately from Lemma 19.2 that for problem $1\left|p_j(r) = p_j\left(1 + b_j Q_r\right), MP(\lambda)\right|C_{\max}$ Theorem 19.5 with $\rho = 4$ is applicable. Hence, we obtain the following statement.

**Theorem 19.6** *Problem* $1\left|p_j(r) = p_j\left(1 + b_j Q_r\right), MP(\lambda)\right|C_{\max}$ *admits an FPTAS that requires* $O\left(n^2/\varepsilon\right)$ *time.*

## 19.4 Bibliographic Notes

Kellerer et al. (2013) study problems $1\left|p_j(r) = p_j(1 + bP_r), MP(0)\right|C_{\max}$ and $1\left|p_j(r) = p_j(1 + bP_r), MP(\lambda)\right|C_{\max}$ (in a slightly general form) with exactly one MP introduced into a schedule and $\sigma = 1$. In other words, for the problems studied in Kellerer et al. (2013) the MP is assumed to fully restore the processing conditions, so that after the maintenance the machine is "as good as new."

It is shown in Kellerer et al. (2013) that problem $1\left|p_j(r) = p_j(1 + bP_r), MP(0)\right|C_{\max}$ is related to the subset-sum problem and an FPTAS presented in Sect. 19.2 is developed. The FPTAS for the subset-sum problem in Theorem 19.3 is due to Kellerer and Pferschy (2004) (see also Sect. 4.2).

Problem (19.18) is related to one of the basic problems of submodular optimization, a so-called resource allocation problem with a convex separable objective function. This explains the fact that problem (19.18) admits a solution by the greedy algorithm (see Hochbaum and Hong (1995) and Katoh and Ibaraki (1998) for details on optimizing convex functions under submodular constraints).

The general problem $1\left|p_j(r) = p_j\left(1 + b_j Q_r\right), MP(\lambda)\right|C_{\max}$ with a single maintenance period is introduced and studied by Rustogi and Strusevich (2016). Unlike in Kellerer et al. (2013), here the MP is a rate-modifying activity, in a sense used by Lee and Leon (2001) (see also Chap. 14).

Theorem 19.5 is due to Erel and Ghosh (2008), the algorithm for solving the continuous relaxation of the problem of minimizing 19.9 is given in Rustogi and Strusevich (2016) (see also Sect. 4.3.2). Lemma 19.2 and Theorem 19.6 are from Rustogi and Strusevich (2016).

Kellerer et al. (2013) in their study of problem $1\left|p_j(r) = p_j(1 + bP_r), MP(\lambda)\right| C_{\max}$ with $\lambda > 0$ and $\sigma = 1$ also rely on Theorem 19.5, but an approximate solution to problem $1\left|p_j(r) = p_j(1 + bP_r), MP(0)\right|C_{\max}$ is used as a lower bound $LB$, and the ratio $UB/LB$ is bounded by a linear function of $\lambda$. To make Theorem 19.5 applicable, an additional assumption is needed, that $\lambda \leq 1$. The approach described by Rustogi and Strusevich (2016) and in Sect. 19.3, based on the rounding algorithm and Lemma 19.2 can also be applied to handling problem $1\left|p_j(r) = p_j(1 + bP_r), MP(\lambda)\right|C_{\max}$ with $\lambda > 0$ and $\sigma = 1$. It will lead to an FPTAS with the running time of $O\left(n^2/\varepsilon\right)$, as in Kellerer et al. (2013), but no additional assumption regarding the value of $\lambda$ is needed.

## References

Erel E, Ghosh JB (2008) FPTAS for half-products minimization with scheduling applications. Discr Appl Math 156:3046–3056

Hochbaum DS, Hong S-P (1995) About strongly polynomial time algorithms for quadratic optimization over submodular constraints. Math Progr 69:269–309

Katoh N, Ibaraki T (1998) Resource allocation problems. In: Du D-Z, Pardalos PM (eds) Handbook of combinatorial optimization, vol 2. Kluwer, Dordrecht, pp 159–260

Kellerer H, Pferschy U (2004) Improved dynamic programming in connection with an FPTAS for the knapsack problem. J Comb Opti 8:5–11

Kellerer H, Rustogi K, Strusevich VA (2013) Approximation schemes for scheduling on a single machine subject to cumulative deterioration and maintenance. J Sched 16:675–683

Lee C-Y, Leon VJ (2001) Machine scheduling with a rate-modifying activity. Eur J Oper Res 128:119–128

Rustogi K, Strusevich VA (2016) Single machine scheduling with a generalized job-dependent cumulative effect. J Sched (In Press). doi:10.1007/s10951-016-0497-6

Tamir A (1993) A strongly polynomial algorithm for minimum convex separable quadratic cost flow problems on two-terminal series-parallel networks. Math Progr 59:117–132

# Chapter 20
# Scheduling with Rate-Modifying Activities on Parallel Machines Under Various Effects

Unlike in the preceding chapters of this part, in this chapter we turn to the models on parallel machines. The jobs of a set $N = \{1, 2, \ldots, n\}$ have to be processed on $m$ parallel machines $M_1, M_2, \ldots, M_m$, where $m \leq n$. Additionally, the decision-maker is presented with a list (RMP$^{[1]}$, RMP$^{[2]}$, ..., RMP$^{[K]}$) of $K \geq 1$ possible rate-modifying activities. The decision-maker may decide which of the listed RMPs to insert into a schedule, on which machine and in which order. We present algorithms that solve problems on parallel machines, provided that the processing times of the jobs are subject to various time-changing effects, and rate-modifying activities can be inserted into a schedule. We aim at designing algorithms with a running time that is polynomial in $n$. The only considered objective function is the total completion time $F(S) = \sum C_j(S)$, since minimizing the objectives $C_{\max}$ and $\sum w_j C_j$ is NP-hard even on two parallel identical machines.

As a rule, the problems that we consider in this chapter are reduced to solving a series of linear assignment problems (LAP). Recall from Chap. 4 that solving an LAP with a full-form $n \times n$ square cost matrix takes $O\left(n^3\right)$ time, while for a product cost matrix Algorithm Match is applicable, which requires $O(n \log n)$ time.

In order to count the number of the LAPs to be solved, we use various combinatorial configurations and identities listed in Sect. 5.3. In the estimations presented in this chapter, we assume that the number $K$ of available RMPs and the number of machines $m$ are constants.

## 20.1 Generic Procedure for Parallel Machines

Recall that in Sect. 12.4 we develop a generic Procedure RMP1 that handles all single machine models that combine time-changing effects and rate-modifying activities. In this section, we present an extension of that procedure to solving problems on parallel

V.A. Strusevich and K. Rustogi, *Scheduling with Times-Changing Effects and Rate-Modifying Activities*, International Series in Operations Research & Management Science 243, DOI 10.1007/978-3-319-39574-6_20

machines. The generic problem to be solved is denoted by $\alpha m|\beta, RMP(K)|\sum C_j$, where $\alpha \in \{P, Q, R\}$, depending on whether the parallel machines are identical, uniform, or unrelated. In the presentation below, we consider the most general case of the unrelated machines, so that the normal processing time of job $j \in N$ on machine $M_i$ is equal to $p_{ij.}$

Similarly to a generic single machine model considered in Sect. 12.4, here the original problem $\alpha m|\beta, RMP(K)|\sum C_j$ reduces to a sequence of the auxiliary problems. Such an auxiliary problem is denoted by $\alpha m'|\beta, RMP(k-1)|\sum C_j$, where $m'$, $1 \le m' \le m$, denotes the number of busy machines, i.e., the machines that will be assigned at least one job. Notice that in the case of unrelated or uniform machines, some machines are not necessarily used. Each auxiliary problem $\alpha m'|\beta, RMP(k-1)|\sum C_j$ is defined by the following outcomes:

**(AP1)** a collection of $m'$ busy machines, $1 \le m' \le m$, which will have jobs assigned to each of them; if required, the selected busy machines are renumbered to become $M_1, M_2, \ldots, M_{m'}$;

**(AP2)** an outcome of the RMP Decisions 1–3 and a distribution of the RMPs over the chosen $m'$ machines, i.e., by a selection of $k-1$ RMPs from a list of $K$ available RMPs and an assignment of the selected RMPs to busy machines, so that a busy machine $M_i$, $1 \le i \le m'$, is assigned $k_i - 1$ RMPs in a given order, where $\sum_{i=1}^{m'} k_i = k-1$;

**(AP3)** a numbering of the $k_i - 1$ RMPs chosen to be introduced on a each busy machine $M_i$ by strings $(i, x)$, $1 \le x \le k_i - 1$, in the order of their appearance in a schedule.

To solve an auxiliary problem $\alpha m'|\beta, RMP(k-1)|\sum C_j$, we need to take three decisions, which essentially are multi-machine extensions of Decisions (B1)-(B3) from Sect. 12.4:

**(BP1)** split the jobs into such a way that $q_i$ jobs are assigned to be processed on machine $M_i$, $1 \le i \le m'$, where $\sum_{i=1}^{m'} q^{[i]} = n$, and determine the number $n^{[i,x]}$ of jobs in group $N^{[i,x]}$, $1 \le x \le k_i$, where for a machine $M_i$, $1 \le i \le m'$, the equality $\sum_{x=1}^{k_i} n^{[i,x]} = q_i$ holds, and the jobs of group $N^{[i,x]}$ are sequenced before the $x$th RMP, while the jobs of group $N^{[i,k_i]}$ are scheduled after the last RMP;

**(BP2)** for each busy machine $M_i$, $1 \le i \le m'$, determine a partition of its $q_i$ jobs into $k_i$ groups $N^{[i,x]}$, $1 \le x \le k_i$;

**(BP3)** find a permutation $\pi^{[i,x]}$ for the jobs in each group $N^{[i,x]}$, $1 \le i \le m'$, $1 \le x \le k_i$.

Extending the reasoning presented in Sect. 12.4, for a particular outcome of Decision (BP1), introduce a schedule $S_{\mathrm{BP1}}(k)$ for an auxiliary problem $\alpha m'|\beta, RMP(k-1)|\sum C_j$ associated with certain outcomes of Decisions (BP2) and (BP3). In schedule $S_{\mathrm{BP1}}(k)$, on each busy machine $M_i$, $1 \le i \le m'$, the jobs are organized into groups $N^{[i,x]}$, $1 \le x \le k_i$, and each group $N^{[i,x]}$ contains $n^{[i,x]}$ jobs, where $\sum_{x=1}^{k_i} n^{[i,x]} = q_i$. Further, let the jobs in $N^{[i,x]}$ be sequenced in accordance with a permutation $\pi^{[i,x]} = \left(\pi^{[i,x]}(1), \pi^{[i,x]}(2), \ldots, \pi^{[i,x]}\left(n^{[i,x]}\right)\right)$, $1 \le x \le k_i$. The actual processing time of a job $j = \pi^{[i,x]}(r)$, scheduled in position $r$, $1 \le r \le n^{[i,x]}$, of the $x$th group, $1 \le x \le k$, on machine $M_i$, $1 \le i \le m'$, is denoted by $p_j^{[i,x]}(r)$ and depends on particular features of the model, normally captured by the strings $\alpha$ and $\beta$. In any case, the total number of groups is equal to $m' + k - 1$, which counts $\sum_{i=1}^{m} k_i$ groups scheduled before all RMPs and one group scheduled on each busy machine after the last RMP.

Associate schedule $S_{\text{BP1}}(k)$ with $m'$ permutations $\pi^{[i]} = \left(\pi^{[i,1]}, \pi^{[i,2]}, \ldots, \pi^{[i,k_i]}\right)$, which specify the sequence of jobs on each busy machine $M_i$, $1 \le i \le m'$. In most problems on parallel machines considered later in this chapter, the sum of the completion times $G_i(S_{\text{BP1}}(k))$ on machine $M_i$, $1 \le i \le m'$, in schedule $S_{\text{BP1}}(k)$ admits a generic representation

$$G_i(S_{\text{BP1}}(k)) = G\left(\pi^{[i]}\right) = \sum_{x=1}^{k_i} \sum_{r=1}^{n^{[i,x]}} W^{[i,x]}_{\pi^{[i,x]}(r)}(r) p_{i,\pi^{[i,x]}(r)} + \Gamma(k_i), \ 1 \le i \le m', \quad (20.1)$$

where $\Gamma(k_i)$ depends only on $k_i$ and some constant terms, while $W^{[i,x]}_{\pi^{[i,x]}(r)}(r)$ is a positional weight that in general is job-, machine- and group-dependent. The product $W^{[i,x]}_{\pi^{[i,x]}(r)}(r) p_{i,\pi^{[i,x]}(r)}$ represents the contribution of job $j = \pi^{[i,x]}(r)$ scheduled in position $r$, $1 \le r \le n^{[i,x]}$, of group $x$, $1 \le x \le k_i$, on machine $M_i$, $1 \le i \le m'$, to the expression (20.1). Since $F(S_{\text{BP1}}(k)) = \sum_{i=1}^{m} G_i(S_{\text{BP1}}(k))$, it follows that $W^{[i,x]}_{\pi^{[i,x]}(r)}(r) p_{i,\pi^{[i,x]}(r)}$ defines the contribution of the respective job into the overall value of the total completion time, which can be written as

$$F(S_{\text{BP1}}(k)) = \sum_{i=1}^{m'} \sum_{x=1}^{k_i} \sum_{r=1}^{n^{[i,x]}} W^{[i,x]}_{\pi^{[i,x]}(r)}(r) p_{i,\pi^{[i,x]}(r)} + \sum_{i=1}^{m'} \Gamma(k_i). \quad (20.2)$$

For an outcome of Decision (BP1), let $S^*_{\text{BP1}}(k)$ be the best schedule that is defined by taking Decisions (BP2) and (BP3) in such a way that schedule $S^*_{\text{BP1}}(k)$ minimizes function (20.2), i.e., $F\left(S^*_{\text{BP1}}(k)\right) \le F(S_{\text{BP1}}(k))$ holds for all possible schedules $S_{\text{BP1}}(k)$. Further, let $S^*(k)$ denote a schedule that is optimal for problem $\alpha m'|\beta, RMP(k-1)|\sum C_j$, i.e., $F(S^*(k)) \le F\left(S^*_{\text{BP1}}(k)\right)$ holds for all possible outcomes of Decision (BP1).

Assuming that for a particular outcome of Decision (BP1) each weight $W_j^{[i,x]}(r)$, $j \in N$, $1 \le x \le k_i$, $1 \le r \le n^{[i,x]}$, can be computed in advance, finding schedule $S^*_{\text{BP1}}(k)$ can be reduced to solving a linear assignment problem (LAP) with an $n \times n$ cost matrix $\mathbf{C} = \left(c_{j,(i,x,r)}\right)$ with the cost values equal to

$$c_{j,(i,x,r)} = W_j^{[i,x]}(r) p_{ij}, \ 1 \le r \le n^{[i,x]}, \ 1 \le i \le m', \ 1 \le x \le k_i, \ j \in N. \quad (20.3)$$

In the cost matrix $\mathbf{C}$, each of the $n$ rows correspond to a job$j \in N$. We number the columns of matrix $\mathbf{C}$ by strings of the form $(i, x, r)$, i.e., (machine, group on the machine, position in the group). For each machine $M_i$, $1 \le i \le m'$, the first $n^{[i,1]}$ of $n$ available positions are related to group $N^{[i,1]}$, the next $n^{[i,2]}$ positions are related to group $N^{[i,2]}$, and so on. The first $\sum_{x=1}^{k_1} n^{[1,x]}$ columns of matrix $\mathbf{C}$ correspond to the available positions associated with machine $M_1$, and the next $\sum_{x=1}^{k_2} n^{[2,x]}$ columns are associated with the positions on machine $M_2$, etc. Since $n = \sum_{i=1}^{m'} \sum_{x=1}^{k_i} n^{[i,x]}$, there will be exactly $n$ columns in matrix $\mathbf{C}$.

As a result, finding schedule $S^*_{\mathrm{BP1}}(k)$ reduces to an LAP written out below

$$\begin{aligned}
\text{minimize} \ & \sum_{j=1}^{n}\sum_{i=1}^{m'}\sum_{x=1}^{k_i}\sum_{r=1}^{n^{[i,x]}} c_{j,(i,x,r)} z_{j,(i,x,r)} \\
\text{subject to} \ & \sum_{i=1}^{m}\sum_{x=1}^{k_i}\sum_{r=1}^{n^{[i,x]}} z_{j,(i,x,r)} = 1, && 1 \le j \le n; \\
& \sum_{j=1}^{n} z_{j,(i,x,r)} = 1, && 1 \le i \le m',\ 1 \le x \le k_i, \\
& && 1 \le r \le n^{[i,x]}; \\
& z_{j,(i,x,r)} \in \{0, 1\}, && 1 \le j \le n,\ 1 \le i \le m', \\
& && 1 \le x \le k_i,\ 1 \le r \le n^{[i,x]}.
\end{aligned} \tag{20.4}$$

Extending Procedure RMP1 from Sect. 12.4, we can set up a generic procedure for finding a schedule $S^*$ that is optimal for problem $\alpha m|\beta, RMP(K)|\sum C_j$.

**Procedure RMPPar**

**Step 1**. Given problem $\alpha m|\beta, RMP(K)|\sum C_j$, for each combination of outcomes (AP1)-(AP3) do

- **(a)** Define an auxiliary problem $\alpha m'|\beta, RMP(k-1)|\sum C_j$.
- **(b)** For each outcome of Decision (BP1) do
  - **(i)** For each busy machine $M_i$, $1 \le i \le m'$, compute appropriate positional weights $W_j^{[i,x]}(r)$, $j \in N$, $1 \le x \le k_i$, $1 \le r \le n^{[i,x]}$, and the constant $\Gamma(k_i)$ that define (20.2).
  - **(ii)** Find schedule $S^*_{\mathrm{BP1}}(k)$ by solving the linear assignment problem (20.4) or its suitable version that is applicable for a particular version of the generic model $\alpha m|\beta, RMP(K)|\sum C_j$.
- **(c)** Determine schedule $S^*(k)$ that is optimal for the current auxiliary problem $\alpha m'|\beta, RMP(k-1)|\sum C_j$, i.e., such that $F(S^*(k)) \le F\left(S^*_{\mathrm{BP1}}(k)\right)$ holds for all schedules $S^*_{\mathrm{BP1}}(k)$ found in Step 1(b).

**Step 2**. Determine schedule $S^*$ that is optimal for the original problem $\alpha m|\beta$, $RMP\,(K)|\sum C_j$, i.e., such that $F(S^*) \le F(S^*(k))$ holds for all schedules $S^*(k)$ found in Step 1.

The lemma below gives an estimation of the number of possible outcomes (AP1)-(AP3) in Step 1 of Procedure RMPPar, as well as the number of all possible Decisions (BP1) to be taken for a particular combination of outcomes (AP1)-(AP3). These estimations are used in the forthcoming sections of this chapter for a purpose of evaluating the running times of algorithms based on Procedure RMPPar. Below, the counting arguments are presented for the most general environment, with unrelated parallel machines.

**Lemma 20.1** *For problem $\alpha m|\beta, RMP(K)|\sum C_j$, the number of outcomes (AP1)-(AP3) in Step 1 of Procedure RMPPar can be estimated as $O\big(K^K n^{m-1}\big)$. For a particular combination of outcomes (AP1)-(AP3), the number of possible outcomes of Decision (BP1) that may lead to an overall optimal schedule $S^*$ is $O\big(n^{m+k-2}\big)$. Moreover, the total number of all possible outcomes (AP1)-(AP3) and outcomes of Decision (BP1), i.e., the number of the assignment problems to be solved in order to find an overall optimal schedule $S^*$ is $O\big(n^{m+K-1}\big)$.*

*Proof* For an outcome (AP1), there are $\binom{m}{m'}$ ways to choose a value of $m'$. It follows from Lemma 12.1 that the number of ways that $k-1$ RMPs can be selected from the given list of $K$ available RMPs and ordered is $\binom{K}{k-1}(k-1)! \le K^{k-1}$. Now, we need to find a way of sharing $m'+k-1$ groups between $m'$ machines. The number of options is equal to the number of compositions $C_{m'+k-1}^{(m')}$ of $m'+k-1$ in exactly $m'$ positive summands. Applying (5.11) with $u = m'+k-1$ and $v = m'$, we have that the number of generated options is

$$\binom{m'+k-2}{m'-1} \le \frac{\big(m'+k-2\big)^{m'-1}}{(m'-1)!}.$$

Recall that with $k-1$ RMPs introduced into a schedule on $m'$ parallel machines the total number of groups is $m'+k-1 \le n$, so that

$$\frac{\big(m'+k-2\big)^{m'-1}}{(m'-1)!} \le \frac{(n-1)^{m'-1}}{(m'-1)!}.$$

Finally, an upper bound on the number of outcomes (AP1)-(AP3) can be estimated as

$$\sum_{k=1}^{K+1} K^{k-1} \sum_{m'=1}^{m} \frac{n^{m'-1}}{(m'-1)!} \le n^{m-1} \sum_{k=1}^{K+1} K^{k-1} = O\big(K^K n^{m-1}\big),$$

as required.

In order to count all possible outcomes of Decision (BP1) we need to count the number of ways that integer $n$ can be split in exactly $m'+k-1$ positive summands $n^{[i,x]}$. The required number is equal to the number of compositions $C_n^{(\le m'+k-1)}$ and applying (5.11) with $u = n$ and $v = m'+k-1$, we have that the total number of all values $n^{[i,x]}$, $1 \le i \le m$, $1 \le x \le k_i$, to be generated for each auxiliary problem $\alpha m'|\beta, RMP(k-1)|\sum C_j$ is $\binom{n+m'+k-2}{m'+k-2} \le \binom{2n}{m'+k-2}$, since the number of groups $m'+k-1$ does not exceed the number of jobs $n$. Thus, we have that

$$\binom{2n}{m'+k-2} \le (2n)^{m'+k-2} = O\left(n^{m'+k-2}\right),$$

as required.

The total number of the linear assignment problems to be solved is given by

$$\begin{aligned}
&\sum_{k=1}^{K+1}\binom{K}{k-1}(k-1)!\sum_{m'=1}^{m}\binom{m}{m'}\binom{m'+k-2}{m'-1}\binom{n+m'+k-2}{m'+k-2}\\
&\le \sum_{k=1}^{K+1}\binom{K}{k-1}(k-1)!\sum_{m'=1}^{m}\binom{m}{m'}\binom{m'+k-2}{m'-1}\frac{(2n)^{m'+k-2}}{(m'+k-2)!},
\end{aligned}$$

where the inequality follows from (5.8) applied with $u = n-1$ and $v = m+k-2$. Further,

$$\binom{m'+k-2}{m'-1}\frac{1}{(m'+k-2)!} = \frac{1}{(m'-1)!(k-1)!},$$

so that

$$\begin{aligned}
&\sum_{k=1}^{K+1}\binom{K}{k-1}(k-1)!\sum_{m'=1}^{m}\binom{m}{m'}\binom{m'+k-2}{m'-1}\frac{(2n)^{m'+k-2}}{(m'+k-2)!}\\
&\le \sum_{k=1}^{K+1}\binom{K}{k-1}\sum_{m'=1}^{m}\binom{m}{m'}\frac{(2n)^{m'+k-2}}{(m'-1)!}\\
&\le \sum_{k=1}^{K+1}\binom{K}{k-1}(2n)^{m+k-2}\sum_{m'=1}^{m}\binom{m}{m'}\frac{1}{(m'-1)!}\\
&\le 2^{2m+K}\sum_{k=1}^{K+1}\binom{K}{k-1}n^{m+k-2} = 2^{2m+K}n^{m-1}\sum_{k=1}^{K+1}\binom{K}{k-1}n^{k-1}\\
&= 2^{2m+K}n^{m-1}(n+1)^{K} = O\left(n^{m+K-1}\right),
\end{aligned}$$

which proves the lemma. □

Notice that if the machines are identical, then all machines will be busy in an optimal schedule, while in the case of uniform machines the $m'$ fastest machines will be selected as busy, i.e., there are 1, $m$ and $2^m$ possible outcomes (AP1) for identical, uniform and unrelated parallel machines, respectively. This observation, however, does not affect the order of magnitude of all estimates presented in Lemma 20.1.

In the forthcoming sections of this chapter, Procedure RMPPar forms the basis of the design of the solutions algorithms for scheduling problems, in which various effects are combined with the introduction of available RMPs.

## 20.2 Models with Rate-Modifying Activities

In this section, we show how to adapt Procedure RMPPar from Sect. 20.1 to solving scheduling problems on parallel machines, provided that multiple RMPs can be introduced into a schedule and they may affect the production rate of the jobs that are scheduled after such an RMP. The models we study here are extensions of a single machine model addressed in Sect. 15.2.

### *20.2.1 Unrelated Machines*

Formally, the jobs of a set $N = \{1, 2, \ldots, n\}$ have to be processed on $m$ unrelated machines $M_1, M_2, \ldots, M_m$, where $m \leq n$. Additionally, the decision-maker is presented with a list of $K \geq 0$ possible rate-modifying activities, which can be either distinct or alike. For an RMP$^{[y]}$, $1 \leq y \leq K$, it is known if that RMP is introduced on machine $M_i$, $1 \leq i \leq m$, then the normal processing time of every job $j$ scheduled after that RMP is multiplied by $\lambda_j^{[i,y]}$. Further, the duration $\bar{\Delta}^{[i,x]}$ of each RMP introduced into a schedule is given by an extension of the general formula (12.2); see (20.6) below for details. Under these conditions, the problem of minimizing the sum of completion times can be denoted by $Rm\left|RMP(K), \bar{\Delta}^{[i,x]}\right| \sum C_j$.

In order to solve problem $Rm\left|RMP(K), \bar{\Delta}^{[i,x]}\right| \sum C_j$, we adapt Procedure RMPPar. Fix outcomes (AP1)-(AP3). Recall that the machines are renumbered in such a way that in accordance with an outcome (AP1), the machines $M_1, \ldots, M_{m'}$ and only those are busy. For a particular outcome of Decision (BP1), introduce a schedule $S_{\text{BP1}}(k)$ for problem $Rm'\left|RMP(k-1), \bar{\Delta}^{[i,x]}\right| \sum C_j$ associated with certain outcomes of Decisions (BP2) and (BP3). In schedule $S_{\text{BP1}}(k)$, the jobs are organized in groups $N^{[i,x]}$, $1 \leq i \leq m'$, $1 \leq x \leq k_i$, so that each group $N^{[i,x]}$ contains $n^{[i,x]}$ jobs, where $\sum_{x=1}^{k_i} n^{[i,x]} = q_i$. Further, let the jobs in $N^{[i,x]}$ be sequenced in accordance with a permutation $\pi^{[i,x]} = \left(\pi^{[i,x]}(1), \pi^{[i,x]}(2), \ldots, \pi^{[i,x]}\left(n^{[i,x]}\right)\right)$, $1 \leq i \leq m'$, $1 \leq x \leq k_i$. Associate schedule $S_{\text{BP1}}(k)$ with $m$ permutations $\pi^{[i]} = \left(\pi^{[i,1]}, \pi^{[i,2]}, \ldots, \pi^{[i,k_i]}\right)$, which specify the sequence of jobs on each machine $M_i$, $1 \leq i \leq m$.

Depending on outcomes (AP1)-(AP3), i.e., on which RMPs are chosen, on which busy machines they are to be performed and in which order, the actual processing time of a job $j = \pi^{[i,x]}(r)$, scheduled in position $r$ of the $x$th group of machine $M_i$ is given by

$$p_j^{[i,x]}(r) = \mu_j^{[i,x]} p_{ij},\ 1 \leq r \leq n^{[i,x]},\ 1 \leq x \leq k,\ 1 \leq i \leq m',$$

where $\mu_j^{[i,x]}$ represents a group-dependent rate-modifying multiplier, which depends on the previously scheduled RMPs and can be written as

$$\mu_j^{[i,x]} = \prod_{v=1}^{x-1} \lambda_j^{[i,x-1]}, \ 1 \le x \le k. \tag{20.5}$$

For a schedule $S_{\mathrm{BP1}}(k)$ with $k-1$ selected RMPs, the duration $\bar{\Delta}^{[i,x]}$ of the $x$-th RMP on a busy machine $M_i$, $1 \le x \le k-1$, $1 \le i \le m'$, is determined as a linear function of the actual durations of the jobs in the preceding group. The corresponding formula is a multi-machine extension of (12.2) and can be written as

$$\bar{\Delta}^{[i,x]} = \sum_{j \in N^{[x]}} \zeta_j^{[i,x]} p_j^{[i,x]} + \eta^{[i,x]}, \tag{20.6}$$

where $\zeta_j^{[i,x]}$ and $\eta^{[i,x]}$ are given positive coefficients. Assuming that the actual processing time of job $j \in N^{[i,x]}$ is equal to $p_j^{[i,x]}$, this job contributes $\zeta_j^{[i,x]} p_j^{[i,x]}$ towards the duration $\bar{\Delta}^{[i,x]}$ of the $x$-th RMP on machine $M_i$.

It follows from (15.21), where the sum of the completion times has been computed for a single machine under the same effect that the sum of the completion times $G_i(S_{\mathrm{BP1}}(k))$ on a busy machine $M_i$, $1 \le i \le m'$, in schedule $S_{\mathrm{BP1}}(k)$ can be written as

$$G_i(S_{\mathrm{BP1}}(k)) = \sum_{x=1}^{k_i} \sum_{r=1}^{n^{[i,x]}} W_{\pi^{[i,x]}(r)}^{[i,x]}(r) p_{i,\pi^{[i,x]}(r)} + \Gamma(k_i), \ 1 \le i \le m', \tag{20.7}$$

where the constant term is given by

$$\Gamma(k_i) = \sum_{x=1}^{k_i} \sum_{r=1}^{n^{[i,x]}} \sum_{v=1}^{x-1} \eta^{[i,v]}, \tag{20.8}$$

and

$$W_{\pi^{[i,x]}(r)}^{[i,x]}(r) = \left[ \left( \sum_{v=x+1}^{k_i} n^{[i,v]} \right) \left(1 + \zeta_{\pi^{[i,x]}(r)}^{[i,x]}\right) + \left(n^{[i,x]} - r + 1\right) \right] \mu_{\pi^{[i,x]}(r)}^{[i,x]}, \tag{20.9}$$
$$1 \le r \le n^{[i,x]}, \ 1 \le x \le k_i, \ 1 \le i \le m',$$

is a job-dependent positional weight, such that the product $W_{\pi^{[i,x]}(r)}^{[i,x]} p_{\pi^{[i,x]}(r)}$ represents the contribution of job $j = \pi^{[i,x]}(r)$ scheduled in position $r$, $1 \le r \le n^{[i,x]}$, of group $x$, $1 \le x \le k_i$, on machine $M_i$, $1 \le i \le m'$, to the objective function $F(S_{\mathrm{BP1}}(k))$.

Thus, for schedule $S_{\mathrm{BP1}}(k)$ the total completion time on all busy machines can be written as

$$F(S_{\mathrm{BP1}}(k)) = \sum_{i=1}^{m'} \sum_{x=1}^{k_i} \sum_{r=1}^{n^{[i,x]}} W_{\pi^{[i,x]}(r)}^{[i,x]}(r) p_{i,\pi^{[i,x]}(r)} + \sum_{i=1}^{m'} \Gamma(k_i). \tag{20.10}$$

The objective (20.10) admits a generic representation (20.2), so that Procedure RMPPar is applicable. Let $S^*_{\mathrm{BP1}}(k)$ denote a schedule such that $F\left(S^*_{\mathrm{BP1}}(k)\right) \leq F(S_{\mathrm{BP1}}(k))$ for a fixed outcome of Decision (BP1).

We know from Sect. 20.1 that finding schedule $S^*_{\mathrm{BP1}}(k)$ reduces to solving an LAP (20.4) with a square cost matrix defined by

$$c_{j,(i,x,r)} = W_j^{[i,x]}(r)p_{ij},\ 1 \leq i \leq m',\ 1 \leq x \leq k_i\ 1 \leq r \leq n^{[i,x]}, \tag{20.11}$$

where the positional weights are defined by (20.9). Notice that solving each of these linear assignment problems requires $O\left(n^3\right)$ time, and the total number of these problems is $O\left(n^{m+K-1}\right)$ as proved in Lemma 20.1. Thus, a direct application of Procedure RMPPar with full enumeration of all outcomes (AP1)-(AP3) and all outcomes of Decision (BP1) leads to the following statement.

**Theorem 20.1** *Problem $Rm\left|RMP(K), \bar{\Delta}^{[i,x]}\right|\sum C_j$ reduces to solving $O\left(n^{m+K-1}\right)$ linear assignment problems of the form (20.4), each of which requires $O\left(n^3\right)$ time to solve, so that the overall running time needed to find an optimal schedule is $O\left(n^{m+K+2}\right)$.*

Below we present an alternative, less straightforward approach. Recall that in order to compute the positional weights $W^{[i,x]}_{\pi^{[i,x]}(r)}(r)$, $1 \leq r \leq n^{[i,x]}$, $2 \leq x \leq k_i$, $1 \leq i \leq m'$, we require prior knowledge of the number of jobs $n^{[i,x]}$ in each group, which is determined by an outcome of Decision (BP1). However, for the first group on every machine, i.e., for $x = 1$, the positional weights $W^{[i,1]}_{\pi^{[i,1]}(r)}(r)$ can be computed without that knowledge. It is easy to verify that irrespective of the number of jobs $n^{[i,1]}$ in the first group on a busy machine $M_i$, the positional weights can be computed as

$$\begin{aligned}
W_j^{[i,1]}\left(n^{[i,1]}\right) &= \left[\left(\sum_{v=2}^{k_i} n^{[i,v]}\right)\left(1+\zeta_j^{[i,1]}\right)+1\right]\mu_j^{[i,1]};\\
W_j^{[i,1]}\left(n^{[i,1]}-1\right) &= \left[\left(\sum_{v=2}^{k_i} n^{[i,v]}\right)\left(1+\zeta_j^{[i,1]}\right)+2\right]\mu_j^{[i,1]};\\
&\ \vdots\\
W_j^{[i,1]}(1) &= \left[\left(\sum_{v=2}^{k_i} n^{[i,v]}\right)\left(1+\zeta_j^{[i,1]}\right)+n^{[i,1]}\right]\mu_j^{[i,1]}.
\end{aligned}$$

The above formulae for the positional weights hold for all values of $n^{[i,1]}$, $1 \leq n^{[i,1]} \leq n$, $1 \leq i \leq m'$. Now, assume that we know the number of jobs $n^{[i,x]}$ in each group $x$, $2 \leq x \leq k_i$, $1 \leq i \leq m'$, in advance, so that $\sum_{i=1}^{m'}\sum_{x=2}^{k_i} n^{[i,x]} \leq n$. Thus, for a busy machine $M_i$, the maximum value of $n^{[i,1]}$ can be $h^{[i,1]} = n - \sum_{x=2}^{k_i} n^{[i,x]}$. In such a situation, the function $\sum C_j$ of the form (20.10) can be minimized by reducing the problem to an LAP of the form (4.1) with a rectangular cost matrix. The number of jobs $n^{[i,1]}$, $1 \le n^{[i,1]} \le h^{[i,1]}$, to be scheduled in the first group of each machine $M_i$, $1 \le i \le m$, can be found on the fly using this procedure.

Define an LAP with a cost matrix $\mathbf{C} = \left(c_{j,(i,x,r)}\right)$ that has $n$ rows and $nm'$ columns. The cost values are still given by (20.11), with $W_j^{[i,x]}(r)$ defined according to (20.9). In the cost matrix $\mathbf{C}$, each of the $n$ rows corresponds to a job$j \in N$, and each of the $nm$ columns corresponds to positions available on $m'$ busy machines, where each machine has $n$ positions associated with it. We number the columns of matrix $\mathbf{C}$ by strings of the form $(i, x, r)$, i.e., machine, group on the machine, and position in the group. The first $n$ columns of matrix $\mathbf{C}$ correspond to the available positions associated with machine $M_1$, the next $n$ columns are associated with the positions on machine $M_2$, etc. For each busy machine $M_i$, $1 \le i \le m'$, the first $h^{[i,1]}$ of $n$ available positions are related to group $N^{[i,1]}$, the next $n^{[i,2]}$ positions are related to group $N^{[i,2]}$, and so on. Recall that $h^{[i,1]} + \sum_{x=2}^{k_i} n^{[i,x]} = n$, $1 \le i \le m$.

As a result, the problem of minimizing the objective function $\sum C_j$ reduces to a rectangular assignment problem written out below

$$
\begin{aligned}
&\text{minimize} \sum_{j=1}^{n}\sum_{i=1}^{m'}\sum_{x=1}^{k_i}\sum_{r=1}^{n^{[i,x]}} c_{j,(i,x,r)} z_{j,(i,x,r)} \\
&\text{subject to} \sum_{i=1}^{m}\sum_{x=1}^{k_i}\sum_{r=1}^{n^{[i,x]}} z_{j,(i,x,r)} = 1, && 1 \le j \le n; \\
&\sum_{j=1}^{n} z_{j,(i,1,r)} \le 1, && 1 \le i \le m',\ 1 \le r \le h^{[i,1]}; \\
&\sum_{j=1}^{n} z_{j,(i,x,r)} = 1, && 1 \le i \le m',\ 2 \le x \le k_i, \\
& && 1 \le r \le n^{[i,x]}; \\
&z_{j,(i,x,r)} \in \{0, 1\}, && 1 \le j \le n,\ 1 \le i \le m', \\
& && 1 \le x \le k_i,\ 1 \le r \le n^{[i,x]}.
\end{aligned}
\tag{20.12}
$$

For the found solution, $z_{j,(i,x,r)} = 1$ implies that in the associated schedule $S^*_{\mathrm{BP1}}(k)$ job $j$ is assigned to the $r$th position of the $x$th group on a busy machine $M_i$. The conditions of (20.12) mean that each job will be assigned to a position and no position will be used more than once. Notice that the above problem of the form (20.12) is more constrained, compared to the rectangular assignment problem defined in (4.1). The third constraint in the above formulation is added to ensure that $n^{[i,x]}$ jobs are scheduled in each group $x$, $2 \le x \le k_i$, $1 \le i \le m'$, provided that $k_i \ge 2$. On the other hand, the number of jobs scheduled in the first group of each machine can be given by $n^{[i,1]} = \sum_{j=1}^{n} \sum_{r=1}^{h^{[i,1]}} z_{j,(i,1,r)}$, $1 \le i \le m'$.

A constrained linear assignment problem of the form (20.12) can be solved by a version of Algorithm LAPBL outlined in Sect. 4.1.1, while a faster Algorithm LAPD apparently is not applicable. In our case, the algorithm is applied to an $n \times \left(nm'\right)$ cost matrix and therefore requires $O(n^3 m')$ time.

Thus, for solving the original problem $Rm\left|RMP(K),\ \bar{\Delta}^{[i,x]}\right|\sum C_j$, we may modify the generic Procedure RMPPar by replacing Step 1(b) by the following

**Step 1(b)$'$.** For each choice of values $n^{[i,x]}$, $1 \le i \le m'$, $2 \le x \le k_i$, such that $\sum_{i=1}^{m'}\sum_{x=2}^{k_i} n^{[i,x]} \le n$ do

- **(i)** For each busy machine $M_i$, $1 \le i \le m'$, define $h^{[i,1]} := n - \sum_{x=2}^{k_i} n^{[i,x]}$, compute the positional weights $W_j^{[i,x]}(r)$ and the constant $\Gamma(k_i)$ by (20.9) and (20.8), respectively, applied with $n^{[i,1]} = h^{[i,1]}$.
- **(ii)** Find schedule $S_{\mathrm{BP1}}^*(k)$ by solving the linear assignment problem (20.12).

In accordance with Procedure RMPPar, in order to find schedule $S^*(k)$ that is the best among all generated in Step 1(b)$'$, the LAP defined in (20.12) must be solved for all assumed values of $n^{[i,x]}$ in each group $x$, $1 \le i \le m'$, $2 \le x \le k_i$, so that $\sum_{i=1}^{m'}\sum_{x=2}^{k_i} n^{[i,x]} \le n$. Since $\sum_{i=1}^{m'}(k_i - 1) = k - 1$, and there is at least one, first group on each machine, finding all values $n^{[i,x]}$, $1 \le i \le m'$, $2 \le x \le k_i$, requires enumeration of integer compositions of integers that do not exceed $n$ in at most $k-1$ positive parts. Applying (5.13) with $u = n$ and $v = k - 1$, this value is equal to $\binom{n+k-1}{k-1}$, which does not exceed $\frac{(n+k-1)^{k-1}}{(k-1)!}$. Thus, since $k$ is a constant, problem $Rm'\left|RMP(k-1),\ \bar{\Delta}^{[i,x]}\right|\sum C_j$ can be solved in $O\left(m'n^{k+2}\right)$ time.

**Lemma 20.2** *A schedule $S^*(k)$ that is optimal for an auxiliary problem $Rm'\left|RMP(k-1),\ \bar{\Delta}^{[i,x]}\right|\sum C_j$ can be found in $O\left(m'n^{k+2}\right)$ time by reducing the problem to a series of $O\left(n^{k-1}\right)$ constrained linear assignment problems of the form (20.12) with a rectangular cost matrix.*

To determine schedule $S^*$ that is optimal for the general problem $Rm|RMP(K),\ \bar{\Delta}^{[i,x]}\left|\sum C_j\right.$, all outcomes (AP1)-(AP3) must be enumerated and the solutions of the resulting auxiliary problems $Rm'\left|RMP(k-1),\ \bar{\Delta}^{[i,x]}\right|\sum C_j$ be compared. Applying Lemma 20.1, the total number of auxiliary problems $Rm'\left|RMP(k-1),\ \bar{\Delta}^{[i,x]}\right|\sum C_j$ that must be solved is given by $\sum_{k=1}^{K+1}\binom{K}{k-1}(k-1)!\sum_{m'=1}^{m}\binom{m}{m'}\binom{m'+k-2}{m'-1}$.

Thus, the running time required to solve problem $Rm\left|RMP(K),\ \bar{\Delta}^{[i,x]}\right|\sum C_j$ can be estimated as

$$O\left(\sum_{k=1}^{K+1}\binom{K}{k-1}(k-1)!\sum_{m'=1}^{m}\binom{m}{m'}\binom{m'+k-2}{m'-1}!\frac{m'n^{k+2}}{(k-1)!}\right)$$
$$= O\left(m'\sum_{k=1}^{K+1}\binom{K}{k-1}n^{k+2}\sum_{m'=1}^{m}\binom{m}{m'}\binom{m'+k-2}{m'-1}\right).$$

Recall that as in the proof of Lemma 20.1, the term $\binom{m'+k-2}{m'-1}$ is shown to have an upper bound of $n^{m'-1}/\left(m'-1\right)!$, so that

$$\sum_{m'=1}^{m}\binom{m}{m'}\binom{m'+k-2}{m'-1} \le 2^m\sum_{m'=1}^{m} n^{m'-1} = O\left(n^{m-1}\right).$$

Besides, $\sum_{k=1}^{K+1}\binom{K}{k-1}n^{k+2} = n^3(n+1)^K = O\big(n^{K+3}\big)$. Thus, the overall running time can be written as $O\big(n^{m+K+2}\big)$.

**Theorem 20.2** *An optimal solution for problem $Rm\big|RMP(K), \bar{\Delta}^{[i,x]}\big|\sum C_j$ can be found in $O\big(n^{m+K+2}\big)$ time by reduction to a sequence of rectangular linear assignment problems.*

Both approaches, a straightforward one and a more elaborated one, considered in Theorems 20.1 and 20.2, respectively, result in the same running time for the general problem. One disadvantage of the former approach is that solving each auxiliary problem $Rm'\big|RMP(k-1), \bar{\Delta}^{[i,x]}\big|\sum C_j$ requires $O\big(n^{m+k+1}\big)$ time, since according to Lemma 20.1 there are $O\big(n^{m+k-2}\big)$ LAPs to solve, in $O\big(n^3\big)$ time each. For the latter approach, Lemma 20.2 holds that gives a better running time for each auxiliary problem.

The difference between the two approaches becomes noticeable if we apply them to simpler versions of the general problem. For example, assume that all given $K \le m$ RMPs are identical and no more than one RMP is allowed on a single machine. The number of all possible outcomes (AP1)-(AP3) does not depend on $n$. For the straightforward approach, the overall running time due to Theorem 20.1 is $O\big(n^{2m+2}\big)$. For the elaborated approach, it can be seen that the overall running time is $O\big(n^{m+3}\big)$ (see Sect. 20.4 for a discussion and references).

## 20.2.2 Uniform Machines

We now consider the problem of minimizing the total completion time $F(S) = \sum C_j(S)$ on $m$ uniform parallel machines. There are three main points of difference between the problem under consideration and problem $Rm\big|RMP(K), \bar{\Delta}^{[i,x]}\big|\sum C_j$ considered in Sect. 20.2.1:

- the rate-modifying multipliers associated with RMP$^{[y]}$ are job-independent and machine-independent, i.e., $\lambda_j^{[y]} = \lambda^{[y]}, j \in N, 1 \le x \le K$.
- the jobs of a set $N = \{1, 2, \dots, n\}$ have to be processed on $m$ uniform machines $M_1, M_2, \dots, M_m$, $m \le n$, where the speed of machine $M_i$, $1 \le i \le m$, is equal to $s_i$, so that we have $p_{ij} = p_j/s_i$, for $j \in N$;
- the duration of RMP$^{[y]}$ chosen to be performed on machine $M_i$ is given by

$$\Delta^{[i,y]}(\tau) = \zeta^{[i,y]}\tau + \eta^{[i,y]}, \tag{20.13}$$

  where $\tau$ is the start time of the RMP, measured from either time zero or from the completion time of the previous RMP on $M_i$, so that $\zeta_j^{[i,y]} = \zeta^{[i,y]}$ for $j \in N$.

We denote the resulting problem by $Qm\big|RMP(K), \Delta^{[i,x]}(\tau)\big|\sum C_j$. Similarly to Sect. 20.2.1, fix outcomes (AP1)-(AP3) and for a particular outcome of Decision (BP1), introduce a schedule $S_{\mathrm{BP1}}(k)$ for an auxiliary problem $Qm'|RMP(k-1),\ \Delta^{[i,x]}(\tau)\left|\sum C_j\right.$ associated with certain outcomes of Decisions (BP2) and (BP3).

Making the above-mentioned changes in (20.9) and (20.10), we deduce that for schedule $S_{\mathrm{BP1}}(k)$ the total completion time on all busy machines can be written as

$$F(S_{\mathrm{BP1}}(k)) = \sum_{i=1}^{m'}\sum_{x=1}^{k_i}\sum_{r=1}^{n^{[i,x]}} W^{[i,x]}(r)p_{\pi^{[i,x]}(r)} + \sum_{i=1}^{m'}\Gamma(k_i), \tag{20.14}$$

where the constant term $\Gamma(k_i)$ is given by (20.8) and

$$W^{[i,x]}(r) = \left[\left(\sum_{v=x+1}^{k_i} n^{[i,v]}\right)(1+\zeta^{[i,x]}) + \left(n^{[i,x]} - r + 1\right)\right]\frac{\mu^{[i,x]}}{s_i},$$
$$1 \le r \le n^{[i,x]},\ 1 \le x \le k_i,\ 1 \le i \le m',$$

is a job-independent positional weight, such that the product $W^{[i,x]}p_{\pi^{[i,x]}(r)}$ represents the contribution of job $j = \pi^{[i,x]}(r)$ scheduled in position $r$, $1 \le r \le n^{[i,x]}$, of group $x$, $1 \le x \le k_i$, on machine $M_i$, $1 \le i \le m'$, to the objective function $F(S_{\mathrm{BP1}}(k))$.

The objective (20.14) admits a generic representation (20.2), so that Procedure RMPPar is applicable. Let $S^*_{\mathrm{BP1}}(k)$ denote a schedule such that $F\left(S^*_{\mathrm{BP1}}(k)\right) \le F(S_{\mathrm{BP1}}(k))$ for a fixed outcome of Decision (BP1). We know from Sect. 20.1 that finding schedule $S^*_{\mathrm{BP1}}(k)$ reduces to solving a linear assignment problem. Since the weights $W^{[i,x]}(r)$ are job-independent, it follows that the cost matrix of such an LAP is a product matrix and the problem can be solved by Algorithm Match (see Sect. 4.1.3).

Assume that the jobs are numbered in accordance with the LPT rule, so that

$$p_1 \ge p_2 \ge \cdots \ge p_n. \tag{20.15}$$

holds. Finding this sequence is done once, for all auxiliary problems to be solved. Algorithm Match will match the $u$th largest processing time to the $u$th smallest positional weight, $1 \le u \le n$. Notice that within each group $N^{[i,x]}$ the positional weights $W^{[i,x]}(r)$, $1 \le r \le n^{[i,x]}$, $1 \le x \le k_i$, $1 \le i \le m$, are monotone, so that a sorted sequence of all positional weights can be obtained in $O(n)$ time. Thus, if we know the number of jobs scheduled in each group, then schedule $S^*_{\mathrm{BP1}}(k)$ can be found in $O(n)$ time.

To determine the optimal solution for the general problem $Qm\left|RMP(K),\ \Delta^{[i,x]}(\tau)\right|\sum C_j$, all options associated with outcomes (AP1)-(AP3) have to be generated and the solutions to the resulting auxiliary problems $Qm'\left|RMP(k-1),\ \Delta^{[i,x]}(\tau)\right|\sum C_j$ must be compared. As established in Lemma 20.1, the total number of options to be enumerated can be estimated as $O\left(n^{m+K-1}\right)$, and we deduce the following statement.

**Theorem 20.3** *An optimal solution for problem $Qm\left|RMP(K),\ \Delta^{[i,x]}(\tau)\right|\sum C_j$ can be found in $O\left(n^{m+K}\right)$ time by using Algorithm Match as a subroutine.*

Notice that for $m = 1$ and $K = 1$, Theorem 20.3 complies with the result of Theorem 15.3.

## 20.3 Models with Changing Processing Times and Rate-Modifying Activities

In this section, we extend our results from Chap. 18 to the problems of minimizing total completion times on parallel machines with time-changing effects and rate-modifying activities.

Formally, the jobs of a set $N = \{1, 2, \dots, n\}$ have to be processed on $m$ parallel machines $M_1, M_2, \dots, M_m$, where $m \le n$, which may be subject to effects of the forms (18.2), (18.25), (18.28), or (18.32), which correspond to an enhanced combined effect, an enhanced positional effect, or an enhanced time-dependent effect, respectively, as presented in Chap. 18. Additionally, the decision-maker is presented with a list (RMP$^{[1]}$,RMP$^{[2]}$, …,RMP$^{[K]}$) of $K \ge 1$ possible rate-modifying activities. The duration of each RMP is given by the general formula (18.4).

If an RMP is inserted into a schedule to become the $x$th RMP on machine $M_i$, denote its duration by $\Delta^{[i,x]}(\tau; \varpi)$. It depends on time $\tau$ elapsed and the number of jobs $\varpi$ processed on machine $M_i$ before the RMP starts, relative to a certain reference point. Such a reference point is content-dependent, and its nature can be explained as in Chap. 18, for single machine models. The way to compute the durations of the RMPs is explained below, see (20.16). The range of the resulting problem is generically denoted by $\alpha m\left|\text{Effect }(e), RMP(K), \Delta^{[i,x]}(\tau; \varpi)\right|\sum C_j$, where $\alpha \in \{P, Q, R\}$, and $e \in \{(18.2), (18.25), (18.28), (18.32)\}$.

Given a problem $\alpha m\left|\text{Effect }(e), RMP(K), \Delta^{[i,x]}(\tau; \varpi)\right|\sum C_j$ of the indicated range, fix outcomes (AP1)-(AP3). Recall that the machines are renumbered in such a way that in accordance with an outcome (AP1), the machines $M_1, \dots, M_{m'}$ and only those are busy. For a particular outcome of Decision (BP1), introduce a schedule $S_{\text{BP1}}(k)$ for an auxiliary problem $\alpha m'\left|\text{Effect }(e), RMP(k-1), \Delta^{[i,x]}(\tau; \varpi)\right|\sum C_j$ associated with certain outcomes of Decisions (BP2) and (BP3). In schedule $S_{\text{BP1}}(k)$, the jobs are organized in groups $N^{[i,x]}$, $1 \le i \le m'$, $1 \le x \le k_i$, so that each group $N^{[i,x]}$ contains $n^{[i,x]}$ jobs, where $\sum_{x=1}^{k_i} n^{[i,x]} = q_i$. Further, let the jobs in $N^{[i,x]}$ be sequenced in accordance with a permutation $\pi^{[i,x]} = \left(\pi^{[i,x]}(1), \pi^{[i,x]}(2), \dots, \pi^{[i,x]}\left(n^{[i,x]}\right)\right)$, $1 \le i \le m'$, $1 \le x \le k_i$. Associate schedule $S_{\text{BP1}}(k)$ with $m'$ permutations $\pi^{[i]} = \left(\pi^{[i,1]}, \pi^{[i,2]}, \dots, \pi^{[i,k_i]}\right)$, which specify the sequence of jobs on each machine $M_i$, $1 \le i \le m'$.

For schedule $S_{\text{BP1}}(k)$, let $F_{i,x}$ denote the durations of the $x$th group on machine $M_i$, $1 \le i \le m'$. Extending (18.4), we can write out an expression for $T_{i,x}$, the duration of the $x$th RMP, $1 \le x \le k_i - 1$, on machine $M_i$ in schedule $S_{\text{BP1}}(k)$, as

$$T_{i,x} = \zeta_{i,1}^{[i,x]} F_{i,1} + \zeta_{i,2}^{[i,x]} F_{i,2} + \cdots + \zeta_{i,x}^{[i,x]} F_{i,x} + \widehat{\eta}^{[i,x]},\ 1 \le x \le k - 1, \qquad (20.16)$$

where for machine $M_i$, $1 \le i \le m'$, the values $\zeta_{i,1}^{[i,x]}, \zeta_2^{[i,x]}, \dots, \zeta_{i,x}^{[i,x]}$, determine how the length of each previous group affects the duration of the RMP scheduled after the $x$-th group, and $\widehat{\eta}^{[i,x]} > 0$ is a constant, $1 \le x \le k - 1$.

### 20.3.1 Unrelated Machines

Let us begin with problem $Rm|\text{Effect (18.28)}, RMP(K), \Delta^{[i,x]}(\tau;\varpi)\big|\sum C_j$, in which unrelated parallel machines are subject to positional effects of the form (18.28). Depending on outcomes (AP1)-(AP3), i.e., on which RMPs are chosen, on which busy machine they are to be performed and in which order, the actual processing time of a job $j=\pi^{[i,x]}(r)$, scheduled in position $r$ of the $x$th group of machine $M_i$ is given by

$$p_j^{[i,x]}(r)=p_{ij}g_j^{[i,x]}(r),\ 1\le r\le n^{[i,x]},\ 1\le x\le k_i,\ 1\le i\le m'. \tag{20.17}$$

It follows from Sect. 18.5.2, where the sum of the completion times has been computed for a single machine under an effect of the form (20.17), that the sum of the completion times $G_i(S_{\mathrm{BP1}}(k_i))$ on a busy machine $M_i$, $1\le i\le m'$, in schedule $S_{\mathrm{BP1}}(k)$ can be written in the form (20.7). The constant term $\Gamma(k_i)$ is given by (18.18), and the positional weights are given by (18.31), so that for a machine $M_i$, $1\le i\le m'$, we have

$$\Gamma(k_i)=\sum_{x=1}^{k_i}\sum_{r=1}^{n^{[i,x]}}\sum_{v=1}^{x-1}\widehat{\eta}^{[i,v]},$$

and

$$W_j^{[i,x]}(r)=\begin{cases}\left[\sum_{v=x+1}^{k_i}n^{[i,v]}\left(1+\sum_{w=x}^{v-1}\zeta_{i,x}^{[i,w]}\right)+\left(n^{[i,x]}-r+1\right)\right]g_j^{[i,x]}(r), & 1\le r\le n^{[i,x]},\\ & 1\le x\le k_i-1,\\ \left(n^{[i,x]}-r+1\right)g_j^{[i,x]}(r), & 1\le r\le n^{[i,x]},\\ & x=k_i.\end{cases} \tag{20.18}$$

where the latter is a job-dependent positional weight, such that the product $W_{\pi^{[i,x]}(r)}^{[i,x]}(r)$ $p_{\pi^{[i,x]}(r)}$ represents the contribution of job $j=\pi^{[i,x]}(r)$ scheduled in position $r$, $1\le r\le n^{[i,x]}$, of group $x$, $1\le x\le k_i$, on machine $M_i$, $1\le i\le m'$, to the objective function $F(S_{\mathrm{BP1}}(k))$.

Thus, for schedule $S_{\mathrm{BP1}}(k)$ the total completion time on all busy machines can be written in the form (20.10), which admits a generic representation (20.2), so that Procedure RMPPar is applicable. As explained for problem $1|\text{Effect (18.28)}, RMP(K), \Delta^{[i,x]}(\tau;\varpi)\big|\sum C_j$ considered in Sect. 18.5.2, all positional weights $W_j^{[i,x]}(r)$, $1\le r\le n^{[i,x]}$, $1\le x\le k_i$, $1\le i\le m'$, $j\in N$, can be computed by (18.31) in $O(n^2)$ time. Since the found positional weights are job-dependent, a full-form LAP will be employed in Step 1(b) of Procedure RMPPar. Solving each of these LAPs requires $O(n^3)$ time, and the total number of these problems is $O(n^{m+K-1})$ as proved in Lemma 20.1. Thus, a direct application of Procedure RMPPar with full enumeration of all outcomes (AP1)-(AP3) and all outcomes of Decision (BP1) leads to the following statement.

**Theorem 20.4** *Problem $Rm|\text{Effect}$ (18.28), $RMP(K)$, $\Delta^{[i,x]}(\tau;\varpi)\big|\sum C_j$ reduces to solving $O\big(n^{m+K-1}\big)$ linear assignment problems of the form (20.4). Each such assignment problem can be created in $O\big(n^2\big)$ time and solved in $O\big(n^3\big)$ time, so that the overall running time needed to find an optimal schedule is $O\big(n^{m+K+2}\big)$.*

Similar to the solution of problem $Rm|\text{Effect}$ (18.28), $RMP(K)$, $\Delta^{[i,x]}(\tau;\varpi)\big|\sum C_j$, problem $Rm|\text{Effect}$ (18.32), $RMP(K)$, $\Delta^{[i,x]}(\tau;\varpi)\big|\sum C_j$ with a time-dependent effect of the form (18.32), and problem $Rm|\text{Effect}$ (18.2), $RMP(K)$, $\Delta^{[i,x]}(\tau;\varpi)|\sum C_j$ with a combined effect of the form (18.2), may also be solved. We skip most of the technical details. It suffices to state that the objective function for each of these problems reduces to the form (20.10), which admits a generic representation (20.2), so that Procedure RMPPar is applicable. The relevant positional weights may be found by adapting the positional weights (18.34) and (18.21), found for problems $1|\text{Effect}$ (18.32), $RMP(K)$, $\Delta^{[i,x]}(\tau;\varpi)\big|\sum C_j$ and $1|\text{Effect}$ (18.2), $RMP(K)$, $\Delta^{[i,x]}(\tau;\varpi)\big|\sum C_j$, respectively, in Chap. 18. Notice that for each problem, the positional weights can be computed in $O(n)$ and $O\big(n^2\big)$ time, respectively. Please also notice that although the positional weights found for the latter problem are of job-independent nature, a full-form LAP is still required to be solved in Step 1(b) of Procedure RMPPar in order to solve each problem $Rm|\text{Effect}$ (18.32), $RMP(K)$, $\Delta^{[i,x]}(\tau;\varpi)\big|\sum C_j$ and $Rm|\text{Effect}$ (18.2), $RMP(K)$, $\Delta^{[i,x]}(\tau;\varpi)\big|\sum C_j$. This is because the resulting square cost matrix $\mathbf{C} = \big(c_{j,(i,x,r)}\big)_{n\times n}$, where

$$c_{j,(i,x,r)} = W^{[i,x]}(r)p_{ij},\ 1\le i\le m',\ 1\le x\le k_i\ 1\le r\le n^{[i,x]},$$

does not satisfy the Monge property (4.9). Intuitively, this happens because the value of the normal processing time is dependent on the machine it is assigned to. Thus, the following statement follows from Lemma 20.1.

**Theorem 20.5** *Each of problems $Rm|\text{Effect}$ (18.32), $RMP(K)$, $\Delta^{[i,x]}(\tau;\varpi)\big|\sum C_j$ and $Rm|\text{Effect}$ (18.2), $RMP(K)$, $\Delta^{[i,x]}(\tau;\varpi)\big|\sum C_j$ reduces to solving $O\big(n^{m+K-1}\big)$ linear assignment problems of the form (20.4), so that the overall running time needed to find an optimal schedule is $O\big(n^{m+K+2}\big)$.*

## 20.3.2 Uniform Machines

Let us begin with problem $Qm|\text{Effect}$ (18.25), $RMP(K)$, $\Delta^{[i,x]}(\tau;\varpi)\big|\sum C_j$, in which uniform parallel machines are subject to positional effects of the form (18.25). Depending on outcomes (AP1)-(AP3), i.e., on which RMPs are chosen, on which busy machine they are to be performed and in which order, the actual processing time of a job $j=\pi^{[i,x]}(r)$, scheduled in position $r$ of the $x$th group of machine $M_i$ is given by

$$p_j^{[i,x]}(r) = \left(\frac{p_j}{s_i}\right)g^{[i,x]}(r),\ 1\le r\le n,\ 1\le x\le k,\ 1\le i\le m'.$$

Redefining the positional factors $\widehat{g}^{[i,x]}(r) := \frac{g^{[i,x]}(r)}{s_i}$, $1 \le r \le n$, $1 \le x \le k$, $1 \le i \le m'$, we can write

$$p_j^{[i,x]}(r) = p_j \widehat{g}^{[i,x]}(r),\ 1 \le r \le n,\ 1 \le x \le k,\ 1 \le i \le m'. \tag{20.19}$$

It follows from Sect. 18.5.2, where the sum of the completion times has been computed for a single machine under an effect of the form (20.19) that the sum of the completion times $G_i(S_{\mathrm{BP1}}(k_i))$ on a busy machine $M_i$, $1 \le i \le m'$, in schedule $S_{\mathrm{BP1}}(k)$ can be written in the form (20.7). The constant term $\Gamma(k_i)$ is given by (18.18), and the positional weights are given by (18.27), so that for a machine $M_i$, $1 \le i \le m'$, we have the constant term $\Gamma(k_i)$ as before and

$$W^{[i,x]}(r) = \begin{cases} \left[\sum_{v=x+1}^{k_i} n^{[i,v]}\left(1+\sum_{w=x}^{v-1}\zeta_{i,x}^{[i,w]}\right) + \left(n^{[i,x]}-r+1\right)\right]\widehat{g}^{[i,x]}(r), & 1 \le r \le n^{[i,x]},\\ & 1 \le x \le k_i - 1, \\ \left(n^{[i,x]}-r+1\right)\widehat{g}^{[i,x]}(r), & 1 \le r \le n^{[i,x]}, \\ & x = k_i. \end{cases} \tag{20.20}$$

where the latter is a job-independent positional weight, such that the product $W^{[i,x]}(r)p_{\pi^{[i,x]}(r)}$ represents the contribution of job $j = \pi^{[i,x]}(r)$ scheduled in position $r$, $1 \le r \le n^{[i,x]}$, of group $x$, $1 \le x \le k_i$, on machine $M_i$, $1 \le i \le m'$, to the objective function $F(S_{\mathrm{BP1}}(k))$.

Thus, for schedule $S_{\mathrm{BP1}}(k)$ the total completion time on all busy machines can be written in the form (20.10), which admits a generic representation (20.2), so that Procedure RMPPar is applicable. As explained for problem $1|\mathrm{Effect}$ (18.25), $RMP(K)$, $\Delta^{[i,x]}(\tau;\varpi)\big|\sum C_j$ considered in Sect. 18.5.2, all positional weights $W^{[i,x]}(r)$, $1 \le r \le n^{[i,x]}$, $1 \le x \le k_i$, $1 \le i \le m'$, can be computed by (20.20) in $O(n)$ time. Since the found positional weights are job-independent, an LAP with a product matrix will have to be solved in Step 1(b) of Procedure RMPPar (see Sect. 20.2.2 for details). Solving each of these LAPs requires $O(n \log n)$ time, and the total number of these problems is $O\left(n^{m+K-1}\right)$ as proved in Lemma 20.1. Thus, a direct application of Procedure RMPPar with full enumeration of all outcomes (AP1)-(AP3) and all outcomes of Decision (BP1) leads to the following statement.

**Theorem 20.6** *Problem* $Qm|\mathrm{Effect}$ *(18.25),* $RMP(K)$, $\Delta^{[i,x]}(\tau;\varpi)\big|\sum C_j$ *reduces to solving* $O\left(n^{m+K-1}\right)$ *linear assignment problems with a product matrix, each of which is solved using Algorithm Match in* $O(n \log n)$ *time, so that the overall running time needed to find an optimal schedule is* $O\left(n^{m+K}\log n\right)$.

Similar to the solution of problem $Qm|\mathrm{Effect}$ (18.25), $RMP(K)$, $\Delta^{[i,x]}(\tau;\varpi)\big|\sum C_j$, problem $Qm|\mathrm{Effect}$ (18.32), $RMP(K)$, $\Delta^{[i,x]}(\tau;\varpi)\big|\sum C_j$ with a time-dependent effect of the form (18.32), and problem $Qm|\mathrm{Effect}$ (18.2), $RMP(K)$, $\Delta^{[i,x]}(\tau;\varpi)\big|\sum C_j$ with a combined effect of the form (18.2), may also be solved. To do this, the approaches outlined in Sect. 20.3.1 for the respective problems in an environment with unrelated machines can be adopted, and the only difference

**Table 20.1** Computational complexities of different versions of problem $\alpha m|\text{Effect }(e), RMP(K), \Delta^{[i,x]}(\tau;\varpi)\left|\sum C_j\right.$

| Problem | Reference | Running time |
|---|---|---|
| $Rm|\text{Effect (18.28), RMP(K)}, \Delta^{[i,x]}(\tau;\varpi)\left|\sum C_j\right.$ | Theorem 20.4 | $O\left(n^{m+K+2}\right)$ |
| $Rm|\text{Effect (18.32), RMP(K)}, \Delta^{[i,x]}(\tau;\varpi)\left|\sum C_j\right.$ | Theorem 20.5 | $O\left(n^{m+K+2}\right)$ |
| $Rm|\text{Effect (18.2), RMP(K)}, \Delta^{[i,x]}(\tau;\varpi)\left|\sum C_j\right.$ | Theorem 20.5 | $O\left(n^{m+K+2}\right)$ |
| $Qm|\text{Effect (18.25), RMP(K)}, \Delta^{[i,x]}(\tau;\varpi)\left|\sum C_j\right.$ | Theorem 20.6 | $O\left(n^{m+K}\log n\right)$ |
| $Qm|\text{Effect (18.32), RMP(K)}, \Delta^{[i,x]}(\tau;\varpi)\left|\sum C_j\right.$ | Theorem 20.7 | $O\left(n^{m+K}\log n\right)$ |
| $Qm|\text{Effect (18.2), RMP(K)}, \Delta^{[i,x]}(\tau;\varpi)\left|\sum C_j\right.$ | Theorem 20.7 | $O\left(n^{m+K+1}\right)$ |

is that since the machines are uniform, there is no need to employ a full-form LAP in Step 1(b) of Procedure RMPPar. In fact, problems $Qm|\text{Effect (18.32)}, RMP(K), \Delta^{[i,x]}(\tau;\varpi)\left|\sum C_j\right.$ and $Qm|\text{Effect (18.2)}, RMP(K), \Delta^{[i,x]}(\tau;\varpi)\left|\sum C_j\right.$ may be solved by reduction to $O\left(n^{m+K-1}\right)$ linear assignment problems with a product matrix, and each of which can be solved in $O(n\log n)$ time. Recall that the relevant positional weights for problems $Qm|\text{Effect (18.32)}, RMP(K), \Delta^{[i,x]}(\tau;\varpi)\left|\sum C_j\right.$ and $Qm|\text{Effect (18.2)}, RMP(K), \Delta^{[i,x]}(\tau;\varpi)\left|\sum C_j\right.$ may be computed by using the formulae (18.34) and (18.21) in $O(n)$ and $O\left(n^2\right)$ time, respectively. Thus, the following statement follows.

**Theorem 20.7** *Each of problems* $Qm|\text{Effect (18.32)}, RMP(K), \Delta^{[i,x]}(\tau;\varpi)\left|\sum C_j\right.$ *and* $Qm|\text{Effect (18.2)}, RMP(K), \Delta^{[i,x]}(\tau;\varpi)\left|\sum C_j\right.$ *reduces to solving* $O\left(n^{m+K-1}\right)$ *linear assignment problems with a product matrix. Each of these assignment problems can be created in* $O(n)$ *and* $O\left(n^2\right)$ *time, respectively, and solved in* $O(n\log n)$ *time for both problems, so that the overall running time needed to find an optimal schedule is* $O\left(n^{m+K}\log n\right)$ *and* $O\left(n^{m+K+1}\right)$*, respectively.*

The main results of this section are summarized in Table 20.1.

## 20.4 Bibliographic Notes

For the models with no time-changing effects, the problems on parallel machines to minimize the total completion time, provided that at most one RMP per machine can be introduced into a schedule, are studied by Wang et al. (2011), for identical machines, and by Cheng et al. (2011) for unrelated machines. Wang et al. (2011) present a rather straightforward result, similar to Theorem 20.1, to minimize the total flow time, so that a solution can be found in $O\left(n^{2m+3}\right)$ time. Notice that this running time has been overestimated, since applying Theorem 20.1 with $K = m$ will result in a running time of $O\left(n^{2m+2}\right)$. Cheng et al. (2011) present an algorithm that is based on the elaborated approach discussed in Sect. 20.2.1 and allows the problem of minimizing the flow time to be solved in $O\left(n^{m+3}\right)$ time. The material of Sect. 20.2.1

is a generalization of the method presented by Cheng et al. (2011) for models with multiple RMPs (see Theorem 20.2).

Theorems 20.5 and 20.7, related to models with rate-modifying activities and combined or time-dependent effects, follow from Rustogi and Strusevich (2014). We are not aware of any others publications that study these effects on parallel machines.

Gara-Ali et al. (2016) study problem $Rm\left|p_{ij}^{[i,x]}(r) = p_{ij}g_j^{[i,x]}(r), RMP(K), \bar{\Delta}^{[x]}\right| \sum C_j$, with job-dependent positional effects on unrelated machines and RMPs whose durations are given by (16.6). They propose a solution approach similar to that discussed in Theorem 20.4, so that the problem of minimizing the total flow time is solvable in $O\left(n^{m+K+2}\right)$ time. Rustogi and Strusevich (2012) also solve the same problem in $O\left(n^{m+K+2}\right)$ time, but for a model in which the RMP durations are given by a simpler formula of the form (16.15). Notice that the solution approach presented in Sect. 20.3.1 for solving problem $Rm|\text{Effect (18.28)}, RMP(K), \Delta^{[i,x]}(\tau; \varpi)\left| \sum C_j\right.$ can easily be extended so that it is able to handle RMP durations of the form (16.6). Another special case of problem $Rm|\text{Effect (18.28)}, RMP(K), \Delta^{[i,x]}(\tau; \varpi)\left| \sum C_j\right.$ is considered by Ji and Cheng (2010), who study the effect of RMPs on identical parallel machines that are subject to a job-dependent learning effect. Ji and Cheng (2010) show that the problem of minimizing the total flow time is solvable in $O\left(n^{m+K+2}\right)$ time (see Sect. 18.6 for a review of this model for a single machine environment).

Rustogi and Strusevich (2012) study problem $Qm\left|p_{ij}^{[i,x]}(r) = p_{ij}g^{[i,x]}(r), RMP(K), \Delta^{[x]}\right| \sum C_j$ with job-independent positional effects on uniform machines and RMPs whose durations are given by (16.15). They propose a solution approach similar to that discussed in Theorem 20.6, so that the problem of minimizing the total flow time is solvable in $O\left(n^{m+K} \log n\right)$ time.

## References

Cheng TCE, Hsu CJ, Yang DL (2011) Unrelated parallel-machine scheduling with deteriorating maintenance activities. Comput Ind Eng 60:602–605

Gara-Ali A, Finke G, Espinouse ML (2016) Parallel-machine scheduling with maintenance: praising the assignment problem. Eur J Oper Res 252:90–97

Ji M, Cheng TCE (2010) Scheduling with job-dependent learning effects and multiple rate-modifying activities. Inf Process Lett 110:460–463

Rustogi K, Strusevich VA (2012) Simple matching vs linear assignment in scheduling models with positional effects: a critical review. Eur J Oper Res 222:393–407

Rustogi K, Strusevich VA (2014) Combining time and position dependent effects on a single machine subject to rate-modifying activities. Omega 42:166–178

Wang J-J, Wang J-B, Liu F (2011) Parallel machines scheduling with a deteriorating maintenance activity. J Oper Res Soc 62:1898–1902

# Index