Lecture Notes in Economics and Mathematical Systems 673

Philipp Melchiors

Dynamic and Stochastic Multi-Project Planning

Springer

Lecture Notes in Economics and Mathematical Systems 673

For further volumes:
http://www.springer.com/series/300

Philipp Melchiors

Dynamic and Stochastic Multi-Project Planning

Springer

Philipp Melchiors
Technische Universität München
München, Germany

ISSN 0075-8442 ISSN 2196-9957 (electronic)
ISBN 978-3-319-04539-9 ISBN 978-3-319-04540-5 (eBook)
DOI 10.1007/978-3-319-04540-5
Springer Cham Heidelberg New York Dordrecht London

Library of Congress Control Number: 2014933551

Printed on acid-free paper

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

Preface

This thesis was written during my current employment as a teaching assistant and doctoral student at the Department for Business Administration and Service Operations Management at Technische Unversität München. It was accepted as a dissertation in May 2013.

This work was made possible by the support of many persons. First of all, I would like to express my gratitude towards my advisor, Prof. Rainer Kolisch, for his guidance in finding this interesting research topic and his ongoing support in many aspects. My time at his department belongs to the most interesting and inspiring periods in my life. In particular, I would like to thank Prof. Roel Leus for giving me the opportunity of a research stay at his department at Katholieke Universiteit Leuven (Belgium) and for being the co-referee of my dissertation. I also owe special thanks to Prof. Stefan Creemers for contributing his expertise and experience in stochastic models. I really enjoyed the open-minded atmosphere and the intensive exchange of ideas during my time in Belgium.

At this point, I would also like to thank my colleagues for their help and for many activities (not only scientific ones). Especially, together with my colleague Hans-Jörg Schütz, who shared the room with me, I could dive into the theory of Markov decision processes for our research problems.

Last but not least, I would like to thank my family for supporting and encouraging me in pursuing my research project.

München, Germany — Philipp Melchiors

Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Background

The starting point of our research were the observations that we made in the research and development (R&D) department of a well known German manufacturing company. The focus of the department has been on the development of hardware components for the control of motor systems which is organized in projects. For each project typically a number of milestones are defined where a certain amount of work should be completed. Thus, project planning is an important task which is, at the same time, very complex for the following reasons. Firstly, multiple projects are processed at the same time which compete for a set of shared resources with limited capacity such as functional departments, specialists or computers. Secondly, processing times of activities as well as precedence relations between activities are *stochastic*. This is caused mainly by the lack of information and by operational uncertainties (cf. Hans et al. [56]). Thirdly, the situation is *dynamic* in the sense that new projects continuously arrive with stochastic interarrival times. In the following we thus refer to such environments as *dynamic-stochastic* multi-project environments. Further examples in the literature, for example the product development (PD) of a chemical company presented by Adler et al. [2], give evidence that such environments are typical for R&D departments. Additionally, they also can be found in other domains such as maintenance for aircrafts (cf. Cohen et al. [28]) or *engineering-to-order* (ETO) in which a project refers to the development and production of a product on order (cf. Zijm [139]).

One issue which frequently occurs in dynamic-stochastic multi-project environments is that flow times of projects through the system tend to be much larger than originally planned and due dates that are not met (cf. Adler et al.[2]). Typical consequences are losses due to a delayed market entry in case of PD-projects (cf. Blackburn [19]) or penalties for not meeting due dates such as for ETO-projects.

The main cause for large flow times is congestion in front of the resources due to different reasons. Firstly, congestion is a consequence of stochastic processing times or interarrival times as it is well known from queueing theory (cf. Gross and

P. Melchiors, *Dynamic and Stochastic Multi-Project Planning*, Lecture Notes in Economics and Mathematical Systems 673, DOI 10.1007/978-3-319-04540-5_1,

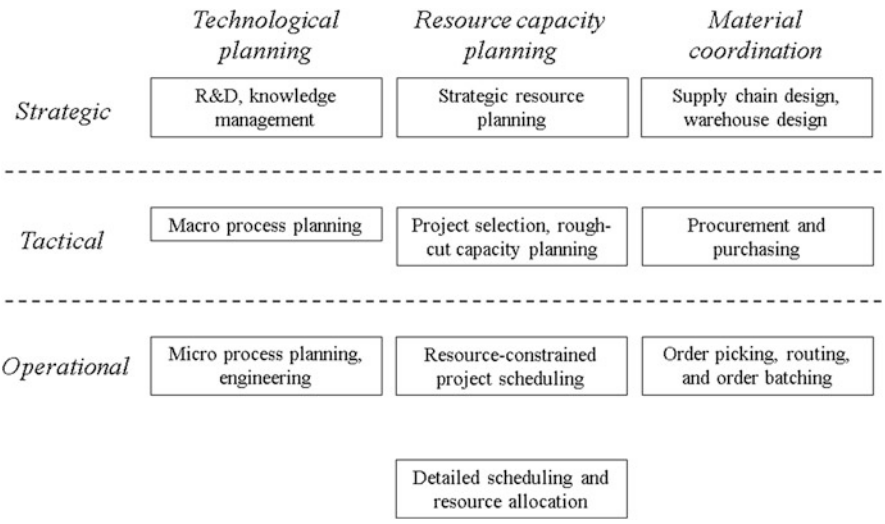

Fig. 1.1 Hierarchical framework (cf. Hans et al. [56])

Harris [54] or Levy and Globerson [87]). As an example, we consider a single resource which processes one activity at a time. Although on average interarrival times may be larger than processing times of activities it may happen that processing times of activities may be larger than interarrival times due to stochastic variations. As a consequence, more activities arrive than are completed in a given period of time such that excess activities are forced to wait.

Secondly, congestion may be a consequence of inappropriate planning. Before going into detail, we outline a hierarchical framework in order to structure planning decisions. We use the framework by Hans et al. [56] as shown in Fig. 1.1. Planning decisions can be categorized according to two dimensions, namely the *hierarchical level* and the *functional area*. Three hierarchical levels – namely *strategic*, *tactical* and *operational* – and three functional areas – namely *technological planning*, *resource capacity planning* and *material coordination* – are distinguished.

Regarding congestion, decisions from resource capacity planning are the most relevant ones.

On the *strategic* level, resource capacity planning refers to dimensioning resource capacities. Thus, if resource capacities are too low, increased congestion is the consequence. However, this topic will not be further addressed in this thesis.

On the *tactical* level, multiple planning steps are typically performed on arrival of an order or project proposal. The first planning step is *macro process planning* (MPP) or rough cut planning. It generally involves the specification of networks consisting of aggregate work packages with rough estimates of resource requirements, durations and precedence relations (cf. De Boer [33]). Resources are aggregated to resource groups such as departments of a company. To make the idea more specific we consider a production department as a part of an ETO-company. In this

context, MPP involves the selection of product routings and global estimation of processing times (cf. Zijm [139]). Selection of product routings involves decisions on the equipment and groups of different resources as well as the sequencing of production steps.

The information obtained by MPP is used to decide whether an order is accepted or rejected. Furthermore, it supports cost estimations for pricing or due date quotation. This is also referred to as *order acceptance* (OA). Furthermore, the information from MPP is used in a *rough cut capacity planning* (RCCP) step. RCCP involves decisions on allocations of aggregated resources where flexibility due to non-regular capacity, such as overtime, may be taken into account.

On the *operational* level, *micro process planning* (and *engineering* for ETO-companies) and *resource-constrained project scheduling* are performed where resource capacities are assumed to be fixed. Furthermore, detailed information is used on the level of single resources such as activities to be done by each resource and their durations (cf. Hans [55]).

In this thesis, we will consider, in more detail, capacity planning problems on the tactical level and scheduling problems on the operational level. Both kinds of problems will be outlined in more detail in Sect. 1.2.

1.2 Research Focus

1.2.1 Order Acceptance and Capacity Planning

In this hierarchical planning process OA and RCCP are typically carried out separately. OA is generally a task of the sales department, which, aiming at high turnovers, accepts as many orders as possible without taking into account the effects on system performance. This conflict of objectives results in large project flow times or due dates that are not met. This suggests a joint optimization (cf. Zijm [139]) of OA decisions and decisions related to capacity planning. Note that the conflict of objectives becomes even more relevant in the face of uncertainty as congestion may occur even if optimal decisions are made.

A related issue that has been neglected so far in the literature is the following. Although less information is needed in RCCP at the tactical level than for detailed scheduling on the operational level, MPP might still need considerable resources, for example from the engineering department (cf. Ishi et al. [63]). Especially, if it is fully performed before OA it can be imagined as a step to be performed with considerable urgency as customers expect quick responses. Due to the urgency, MPP before OA is expected to be *crashed* in the sense of reducing its processing time by allocation of additional resource capacity (cf. Kelley and Walker [68]) at higher cost. Thus, if at least a part of MPP is postponed to a time after OA costs can be reduced by performing MPP with less resource capacity. Furthermore, the customer has to wait for a response which may lead to a price reduction.

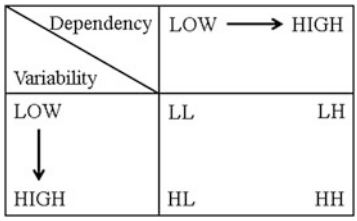

Fig. 1.2 Positioning framework for multi-project environments (cf. Hans et al. [56])

To the best of our knowledge, joint optimization w.r.t. long term optimal decisions of OA and capacity planning has not been coveredyet in the literature for a *dynamic-stochastic* multi-project environment.

1.2.2 Resource-Constrained Multi-project Scheduling

Multi-project environments can be categorized using the positioning framework presented by Herroelen and Leus [58] and Hans et al. [56] according to *dependency* and *variability* as shown in Fig. 1.2. Dependency refers to the degree a particular project is dependent on influences external to the individual project. Dependencies may arise from factors external to the organization but also from internal factors such as shared resources. For multi-project environments that are characterized by low dependency, projects are considered separately such that single-project scheduling problems are obtained. In case of high dependency, projects must be considered simultaneously which leads to multi-project scheduling problems.

Variability refers to variability of the work environment, for example in terms of activity durations, interarrival times or precedence relations between activities. For multi-project environments characterized by low variability, deterministic or *proactive baseline schedules* can be determined in which buffers are used to protect the schedule from variations of activity durations. In case of high variability, either proactive schedules are revised in the scope of *reactive* scheduling or decisions are made using *scheduling policies*. Scheduling policies dynamically make scheduling decisions at stochastic decision times based on available information, for example from the observed past and a-priori knowledge about probability distributions. Thus, policies do not need a baseline schedule anymore.

According to this framework, it becomes obvious that dynamic-stochastic multi-project environments, as outlined before, are characterized by high variability and high dependency. Thus, scheduling is most complex as variability needs to be taken into account in addition to simultaneous consideration of projects. Furthermore, good scheduling decisions have to account for their long term effect due to the fact that new projects continuously arrive while having ongoing projects in the system. This reduces the benefit of proactive baseline schedules such that in our point of view scheduling policies are most appropriate.

To the best of our knowledge the usage of scheduling policies in dynamic-stochastic multi-project environment except for special cases such as queueing networks has not been systematically investigated yet.

1.3 Outline

This thesis is organized as follows. In Chap. 2, we present firstly general assumptions and notation. Secondly, we give formal statements of the dynamic-stochastic multi-project scheduling problem with and without preemptions, and of the dynamic-stochastic order acceptance and capacity planning problem.

Chapter 3 is dedicated to a review of the most relevant literature for the problems considered in this thesis. The review is divided into four parts. The first part refers to the methodologies used in this thesis, namely dynamic and approximate dynamic programming. In the second part, relevant literature from the field of project scheduling is reviewed while the third part refers to relevant literature from the more general problem of capacity planning. In the fourth part, we review literature from the field of order acceptance.

Chapter 4 presents fundamentals of the theory of continuous-time Markov decision processes (CTMDPs) and definitions as needed in this thesis. Furthermore, the most important algorithms for determining optimal policies are outlined. Finally, we briefly address how CTMDPs and algorithms can be implemented efficiently.

The focus of Chap. 5 is on a formal description of relevant problem parameters and the procedure for generating problem instances.

In Chap. 6, we investigate, in an extensive simulation study, the performance of non-preemptive *resource-based priority policies* (RBPs) for scheduling in a dynamic-stochastic multi-project environment. Non-preemptive RBPs combine priority rules for prioritizing waiting activities with the parallel scheduling scheme. Thus, they schedule as many activities as possible at a time while no activities in process may be preempted. The priority rules for the RBPs have been selected according to their performance for related problems while the problem instances have been generated with different problem parameters being controlled. The analysis of the results is focused, firstly, on the main effects of problem parameters, and, secondly, on recommendations concerning the policies to be used for given values of the problem parameters.

Chapter 7 considers the computations of optimal and near optimal scheduling policies for the scheduling problem without preemptions (*non-preemptive problem*) and with preemptions (*preemptive problem*). The chapter is divided into five parts. In the first part, both problems are modeled as CTMDPs where structural properties are exploited to obtain simplified models and to reduce the computational burden for obtaining optimal policies. Complexity results in terms of state space cardinalities are also obtained.

In the second part, we show that the optimal non-preemptive policy is a priority index policy for systems consisting of a single resource. The result applies also to cases where activity durations are generally distributed.

In the third part, the class of *project state ordering* policies (POPs) is presented. The main benefit of POPs is their potential to considerably reduce state space cardinality, especially if project networks have only few precedence relations, at only little loss of performance. Thus, larger problem instances can be solved.

As, even when using POPs, the range of problem instances is still limited we consider in the fourth part two approaches of *approximate dynamic programming* (ADP). The first approach exploits the fact that the state space for the preemptive problem is much smaller than for the non-preemptive problem and uses the value function from the preemptive problem in order to obtain a policy for the non-preemptive problem. The second approach uses *linear function approximation* in order to obtain approximations of the value function. In this context, we address a number of methodologies to obtain linear function approximations.

To investigate the performance of optimal policies as well as near optimal policies from ADP, we carried out an extensive computational study of which the results are presented in the fifth part. The study is based on a set of problem instances, where a number of problem parameters are controlled. The analysis is divided into three main parts. In the first part, we analyze for the preemptive problem the potential of POPs to reduce state space cardinality and the performance of optimal POPs relative to the performance of optimal policies without restriction to a certain class (*general policies*). As we have found that the restriction to POPs does not lead to a loss of performance for most problem instances we compare optimal POPs instead of general policies with a number of RBPs. In addition, we also consider the performance of resource-based priority POPs (RBPOPs) that combine the idea of POPs with the idea of RBPs. In the second part, we carry out a similar analysis for the non-preemptive problem. In addition, we investigate the performance of non-preemptive POPs which are based on the value function from the preemptive problem. In the third part, we investigate the performance of POPs based on linear function approximation. At first, we analyze the performance of the different methodologies for obtaining an approximation of the value function. Afterwards, we demonstrate, based on a number of case studies, that linear function approximation may lead to policies that clearly outperform RBPs.

Chapter 8 addresses the joint optimization of order acceptance decisions with capacity planning on the tactical level. After modeling the problem as a CTMDP, we show in a first step how optimal policies can be efficiently determined. In a second step, we investigate the structure of optimal policies which may guide the search for good heuristic policies. In a third step, we investigate conditions when the planning problem can be simplified by performing MPP after OA or ignoring the existence non-regular capacity.

Chapter 2
Problem Statements

2.1 General Assumptions and Notation

In this section, we present general assumptions and introduce basic notation.

Our assumptions are guided by the model proposed by Adler et al. [2] who have presented a generalized queueing network referred to as *processing network* for the analysis of a dynamic-stochastic multi-project environment. We believe that such a model is useful for three reasons. Firstly, important aspects such as congestion leading to considerable delays of projects are captured. Secondly, the assumption of stationary distributions permits the long term analysis of decisions. Finally, the assumptions of the model appear to be realistic enough to make predictions of the behavior of realistic dynamic-stochastic multi-project environments.

Next, we describe the most important components and assumptions of our models.

2.1.1 Projects

Projects arrive dynamically according to a stochastic arrival process. Project j is composed of multiple work packages being referred to as activities where activity i has a stochastic duration $\boldsymbol{d}_{ij}$. Between the activities there are typically precedence relations.

The life cycle of a project can be outlined as follows. On arrival of project j at time $\boldsymbol{t}_j^a$ the due date is assigned. It is given by $\boldsymbol{t}_j^a + \boldsymbol{D}_j^{\max}$ where $\boldsymbol{D}_j^{\max}$ is defined to be the maximum flow time that is allowed to a project. As $\boldsymbol{D}_j^{\max}$ may be subject to negotiations between the customer and the organization the maximum flow time is modeled as a random variable.

Furthermore, we define $\mathcal{U}_j(t)$ to be the set of *unfinished activities* of project j and $\mathcal{U}_j^{\mathrm{S}}(t)$ the set of *unscheduled activities* at time t. In addition to $\mathcal{U}_j^{\mathrm{S}}(t)$, $\mathcal{U}_j(t)$

P. Melchiors, *Dynamic and Stochastic Multi-Project Planning*, Lecture Notes in Economics and Mathematical Systems 673, DOI 10.1007/978-3-319-04540-5_2,

may also contain activities that already have been scheduled but still are in process at time t such that they are unfinished but not unscheduled.

Afterwards, activities $i \in \mathcal{U}_j(t)$ are processed until all activities are completed at time $\boldsymbol{t}_j^c$ such that $\mathcal{U}_j(\boldsymbol{t}_j^c) = \emptyset$. At completion time $\boldsymbol{t}_j^c$, a project is assumed to leave the system. Thus, we obtain the flow time $\boldsymbol{F}_j = \boldsymbol{t}_j^c - \boldsymbol{t}_j^a$.

As we assume a stochastic arrival process we have a random variable $\boldsymbol{\delta}_j = \boldsymbol{t}_j^a - \boldsymbol{t}_{j-1}^a$ for the time between the arrivals of project $j-1$ and project j. Obviously, we may have multiple projects in the system at a time such that we have a set $\mathcal{J}(t)$ of the projects being in the system at time t.

2.1.2 Resources

Resources can be departments, single employees or machines depending on the level of detail. As in a company multiple resources may have the same set of skills, such that they are able to process the same types of activities, they are grouped as a resource type. In addition, this helps to exploit pooling effects (cf. Adler et al. [2] or Hopp and Spearman [61]). Thus, an activity may wait in front of a resource type as a group of resources with the necessary skills instead of waiting in front of a single resource that has been specified in advance.

In the following the multi-project environment is considered as a system composed of a set $\mathcal{R}$ of resource type where each resource type $r \in \mathcal{R}$ comprises c_r identical resources w.r.t. skills and qualifications such as product engineers or application engineers (cf. Adler et al. [2]). As scheduling decisions typically take place at the operational level c_r can be considered as fixed. Such resources being constantly available for the entire planning horizon are also referred to as *renewable* resources (cf. Demeulemeester and Herroelen [39]). For the special case where $c_r = 1 \ \forall r \in \mathcal{R}$, we also refer to resources $r \in \mathcal{R}$.

Note that at the level of order acceptance (OA) decisions terminology is slightly changed. As order acceptance and capacity planning decisions refer to the tactical level there are more degrees of freedom concerning the usage of resources. Resources are more aggregated, such as departments, and are taken into account using capacities being the amount of work, measured in time units, which can be processed per time unit. Capacity may be extended using overtime or hiring additional employees (cf. Hans [55]). Thus, in this context, it is more useful to refer to resources and capacity instead of resource types and resources.

In our model, we assume that each resource entirely processes a given activity. This assumption deviates from classical static and deterministic resource-constrained multi-project scheduling problems e.g. Pritsker et al. [107] where one activity may need multiple resources of possibly multiple types. However, the assumption is more realistic in the stochastic case as the work content a resource has to process may be subject to stochastic variations. Hence, we consider the problem at the level of *tasks* that are to be processed by one resource (cf. Adler et al. [2]) and thus are parts of an activity in the classical sense. For simplicity, we refer to tasks

as activities. Furthermore, scheduling may also be done at the level of departments where activities are larger work packages to be entirely processed by a department. Such problems may occur as part of *rough-cut capacity planning* (cf. Hans [55]) considered in Chap. 8.

2.1.3 Project Types

We assume that projects can be categorized into a set $\mathcal{P}$ of *project types* composed of multiple *activity types*.

For each project type, we assume the arrival process to be Poisson. The assumption of project types is realistic as, for many cases, projects have characteristics in common such as projects that develop new products and projects that modify existing products (cf. Adler et al. [2]). A Poisson arrival process is realistic as projects typically come from a large pool of customers such that arrivals can be considered as nearly independent stochastic events.

Thus, our model has two levels of abstraction. Project types and individual projects. Therefore, we use the following notation in order to make the difference clear. For individual projects, we use index j and refer to the type of project j by $\boldsymbol{p}_j$. For addressing activity type i of project type p we use tuple (i, p). As each activity of an individual project is of an activity type we reuse index i and refer to individual activities by tuples (i, j). Hence, activity (i, j) is of type $(i, \boldsymbol{p}_j)$ such that the information for each project j and activity (i, j) is obtained through project type $\boldsymbol{p}_j$.

For each project type $p \in \mathcal{P}$, we assume the following information to be given.

- Arrival rate λ_p.
- Distribution for the maximum flow time with mean $\overline{D}_p^{\max}$.
- Holding cost w_p incurred per time unit, the due date is exceeded.
- Payoff y_p obtained on completion. We assume that for y_p, any project related fixed costs that do not depend on decisions are already subtracted e.g. cost for material.
- A network depicted as a graph $\mathcal{G}_p = (\mathcal{V}_p, \mathcal{A}_p)$ with a node set $\mathcal{V}_p$ representing activity types and arc set $\mathcal{A}_p$ representing precedence relations between activity types. $(i, i') \in \mathcal{A}_p$ depicts a precedence relation between activity type (i, p) and activity type (i', p) of the type *finish-to-start* with minimum duration 0.
- For each activity type (i, p) with $i \in \mathcal{V}_p$: Distribution for the duration with mean $\overline{d}_{ip}$ and resource type r_{ip} required for processing.

Figure 2.1 illustrates the relationship between project types and individual projects with the given information. Furthermore, we define

$$\mathcal{V}_p^{\text{Succ}}(i) = \{i' \in \mathcal{V}_p | < i, i' > \in \mathcal{A}_p\} \tag{2.1}$$

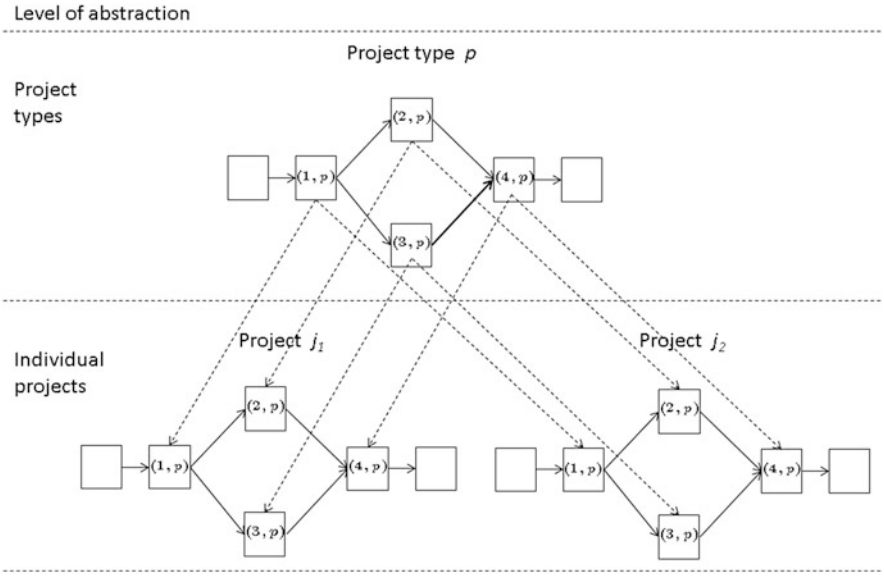

Fig. 2.1 Relationship between project types and individual projects

to be the set of immediate successors and

$$\mathcal{V}_p^{\text{Pred}}(i) = \{i' \in \mathcal{V}_p | < i', i > \in \mathcal{A}_p\} \tag{2.2}$$

to be the set of immediate predecessors of activity type (i, p) with $i \in \mathcal{V}_p$.

As the arrival process is a superposition of Poisson arrival processes random variables $\boldsymbol{\delta}_j$ are stochastically independent and exponentially distributed with rate $\lambda = \sum_{p \in \mathcal{P}} \lambda_p$. Thus, for each project j arriving at the system the type $\boldsymbol{p}_j$ is a random variable where $\boldsymbol{p}_j = p$ occurs with probability

$$\frac{\lambda_p}{\lambda}$$

Furthermore, we assume that activity durations $\boldsymbol{d}_{ij}$ are stochastically independent random variables with mean $\overline{d}_{i\boldsymbol{p}_j}$.

Note that distributions for the random variables considered so far are typically given by a *density function* or *cumulative distribution function* (CDF) that may have more parameters than the mean. For such cases, further parameters need to be added to the information for each activity type (i, p). When assumptions w.r.t. the distributions are made more specific later we will address the necessary parameters.

Finally, we assume that projects are accepted as long as a maximum number $K^{\max}$ of projects in the system has not been reached. Hence, we consider

a *semi-open system* (cf. Buzacott and Shantikumar [25]). This assumption is motivated by three reasons. Firstly, bounding problem size is necessary to apply optimization methodologies as used in Chap. 7. Secondly, it is realistic to restrict acceptance of new projects to the system as holding costs may become too large. Finally, we avoid the well known problem that the system may become *unstable* in the sense that the long term number of projects in the system may be unbounded. Be ρ_r the *traffic intensity* of resource type $r \in \mathcal{R}$ as given by

$$\rho_r = \sum_{p \in \mathcal{P}} \lambda_p \sum_{i \in \mathcal{V}_{pr}} \overline{d}_{ip} \tag{2.3}$$

where $\mathcal{V}_{pr}$ is the set of activity types to be processed by resource type r. ρ_r can be interpreted as the amount of work that arrives per time unit at resource type r (cf. Brémaud [21]). Then, a necessary condition for *stability* is

$$\rho_r < c_r \tag{2.4}$$

(2.4) states that the amount of work that arrives at the system per time unit must not exceed the number of resources per time unit as given by c_r. Unfortunately, (2.4) is not sufficient as it is well known from scheduling in queueing networks that under some policies the system may become unstable although the conditions is met (cf. Kumar and Seidman [80] or Meyn [93]).

2.1.4 Objective Functions

The most general objective function is the maximization of the long term average profit per project as given by

$$Z = \max \lim_{J \longrightarrow \infty} E\left[\frac{1}{J}\sum_{j=1}^{J} \mathbb{1}\left\{|\mathcal{J}(\boldsymbol{t}_j^a)| < K^{\max}\right\}(y_{\boldsymbol{p}_j} - w_{\boldsymbol{p}_j} \cdot (\boldsymbol{F}_j - \boldsymbol{D}_j^{\max})^+)\right] \tag{2.5}$$

$\mathbb{1}\left\{|\mathcal{J}(\boldsymbol{t}_j^a)| < K^{\max}\right\}$ is an indicator function being 1 if the number of projects in the system *without* consideration of the new project is below $K^{\max}$. The term $w_{\boldsymbol{p}_j} \cdot (\boldsymbol{F}_j - \boldsymbol{D}_j^{\max})^+$ gives the tardiness cost (or weighted tardiness) that increases linearly with the tardiness $(\boldsymbol{F}_j - \boldsymbol{D}_j^{\max})^+$ of a project. Further cost are assumed to be fixed (at least on average) such that they may be implicitly taken into account by $y_{\boldsymbol{p}_j}$.

For our investigation, instead of the profit-oriented objective function we will also use an equivalent cost-oriented objective function that is given by

$$Z = \min \lim_{J \longrightarrow \infty} E \left[\frac{1}{J} \sum_{j=0}^{J} \mathbb{1} \left\{ |\mathcal{J}(\boldsymbol{t}_j^a)| < K^{\max} \right\} w_{\boldsymbol{p}_j} (\boldsymbol{F}_j - \boldsymbol{D}_j^{\max})^+ + \left(1 - \mathbb{1} \left\{ |\mathcal{J}(\boldsymbol{t}_j^a)| < K^{\max} \right\} \right) y_{\boldsymbol{p}_j} \right] \quad (2.6)$$

Instead of accounting for the payoff y_p obtained for each project admitted to the system, we account for the lost payoffs for the projects not admitted to the system when $|\mathcal{J}(\boldsymbol{t}_j^a)| = K^{\max}$.

Note that for unbounded $K^{\max}$ (2.5) and (2.6) turn into the minimization of the long term average weighted tardiness per project. If, in addition, $\boldsymbol{D}_j^{\max} = 0 \; \forall j = 1, \ldots, J$ holding cost w_p are incurred per time unit a project of type p is in the system such that both objective functions reduce to the minimization of the average weighted flow time per project.

2.2 Dynamic-Stochastic Multi-project Scheduling Problem

In this section, we describe in Sect. 2.2.1 the scheduling problem *without* preemptions (non-preemptive problem) and in Sect. 2.2.2 the scheduling problem *with* preemptions (preemptive problem).

2.2.1 Non-preemptive Scheduling Problem

Generally scheduling is a step where activities are laid out in the time order in which they have to be performed (cf. Demeulemeester and Herroelen [39]). This implies decisions on the start times of the activities. Then, scheduling decisions are subject to the following constraints

$$\boldsymbol{s}_{ij} + \boldsymbol{d}_{ij} \leq \boldsymbol{s}_{i'j} \quad \forall j \in \mathcal{J}(t); (i, i') \in \mathcal{A}_{\boldsymbol{p}_j}; t \geq 0 \quad (2.7)$$

$$|\mathcal{E}(r, t)| \leq c_r \quad \forall r \in \mathcal{R}; t \geq 0 \quad (2.8)$$

where $\boldsymbol{s}_{ij}$ is the start time of activity (i, j) and $\mathcal{E}_r(t)$ the set of activities executed at time t on resource type $r \in \mathcal{R}$.

Constraints (2.7) depict the precedence constraints between activities. The start time $\boldsymbol{s}_{i'j}$ of activity (i', j) must be greater or equal the completion time of each immediate predecessor activities (i, j). Constraints (2.8) depict the resource constraints. For each resource type $r \in \mathcal{R}$, the numbers of activities $|\mathcal{E}(r, t)|$ processed at time t, has to be less than or equal to the number of resources c_r.

As we consider a dynamic-stochastic environment, scheduling decisions have to meet the *nonanticipativity constraint* (cf. Fernandez et al. [48]) which states that

only information that has become known up to decision time t may be used. Thus, it is not possible to compute a baseline schedule based on the realizations of the random variables in advance. Instead, we employ a scheduling policy (or strategy) π. In the literature there are multiple views on the scheduling problem (cf. Stork [122]) such that the definition of a scheduling policy depends on the respective view. In this thesis, we interpret the dynamic and stochastic scheduling problem as a multistage decision process where decisions are to be made at decision times such that we define a scheduling policy as follows.

Definition 2.2.1. A **non-preemptive scheduling policy** π defines actions at a decision time t and defines a tentative next decision time $\boldsymbol{t}^{\text{next}}$. A decision consists of the sets $\mathcal{B}^{\text{S}}(r,t) \subseteq \mathcal{W}(r,t)\ \forall r \in \mathcal{R}$ of activities to be started at the current time t on resource type r with $|\mathcal{E}(r,t) \cup \mathcal{B}^{\text{S}}(r,t)| \leq c_r\}$.

$\mathcal{W}(r,t)$ is the set of activities waiting for resource type r at time t. An activity (i,j) is added to $\mathcal{W}(r_{i\boldsymbol{p}_j},t)$ as soon as all its predecessors have been completed. The definition is based on the general definition by Fernandez et al. [49]. As we have stationary probability distributions we can restrict our considerations without loss of optimality to *stationary* scheduling policies (cf. Puterman [108]) where a decision does not depend on the decision time but only on the available system information at a decision time. In order to reduce the computational complexity and facilitate analysis we assume that a scheduling policy π is

1. *Non-idling* (cf. Meyn [93]) such that $|\mathcal{E}(r,t) \cup \mathcal{B}^{\text{S}}(r,t)| = \min\{c_r, |\mathcal{E}(r,t) \cup \mathcal{W}(r,t)|\}$ if $|\mathcal{E}(r,t)| < c_r$.
2. *Stationary* (cf. Puterman [108]) such that a decision does not depend on the decision time but only on the system information available at a decision time.

Then, decision times are given by the following theorem.

Theorem 2.2.1. *An non-idling and non-preemptive policy π considers the system only on arrival of new projects or completions of an activities.*

Proof. A non-idling policy stops scheduling activities at a decision time t as soon as no further activities can be scheduled. This is the case when for all resource types $r \in \mathcal{R}$ either all resources are busy such that $|\mathcal{E}(r,t) \cup \mathcal{B}^{\text{S}}(r,t)| = c_r$ or no further activities are waiting as $\mathcal{W}(r,t) \backslash \mathcal{B}^{\text{S}}(r,t) = \emptyset$. As no preemptions are allowed, the next time t^{next} where the policy can schedule activities is when at least one of the following two cases occurs.

1. Resources become idle due to completion of activities. Then, activities waiting for resources of the respective types may be scheduled.
2. New activities become ready for execution. This happens on arrivals of new projects or completion of activities if they are the last predecessor activities yet to be completed of their direct successors in the project network. Then, if there are idle resources of the required types activities may be scheduled.

Thus, for making decisions, it is sufficient to consider the system on arrivals of new projects or completions of activities. □

2.2.2 Preemptive Scheduling Problem

If we allow that an activity be preempted (preemptive problem) two cases must be distinguished (cf. Demeulemeester and Herroelen [39]). In the first case activities are *repeated* while in the second case they are *resumed* later. Repeating an activity implies that the work that already has been done is ignored the next time an activity is scheduled. Thus, the distribution of its duration is independent from the amount of work already done. By contrast, resuming an activity implies that work already done is not ignored such that the activity is resumed from the point where its execution has been preempted. Thus, the distribution of its remaining duration depend on the work already done. An exception is the case where the duration is exponentially distributed as the exponential distribution has the *memoryless property*.[1]

In order to keep analysis simple we assume exponentially distributed activity durations for finding optimal policies (cf. Chap. 7). Thus, we do not have to distinguish between repeating or resuming and activity such that we consider both.

Furthermore, we assume that preempting and rescheduling an activity can be done at no cost at decision time.

Next, we give the definition of a scheduling policy for the preemptive problem.

Definition 2.2.2. A **preemptive scheduling policy** π defines decisions at a decision time t and defines a tentative next decision time $\boldsymbol{t}^{\text{next}}$. A decision consists of the sets $\mathcal{B}^{\text{P}}(r,t) \subseteq \mathcal{E}(r,t)\ \forall r \in \mathcal{R}$ of activities to be preempted on resource type $r \in \mathcal{R}$ and the sets $\mathcal{B}^{\text{S}}(r,t) \subseteq \mathcal{W}(r,t)\ \forall r \in \mathcal{R}$ of activities to be started at the current time t on resource type r with $|\mathcal{E}(r,t) \cup \mathcal{B}^{\text{S}}(r,t) \backslash \mathcal{B}^{\text{P}}(r,t)| \leq c_r$.

The definition is based on a general definition by Fernandez et al. [49]. Again, we can restrict our considerations without loss of optimality to *stationary* scheduling policies (cf. Puterman [108]) where a decision does not depend on the decision time but only on available system information. To simplify analysis, we allow preemptive scheduling policies to be idling such that we allow $|\mathcal{E}(r,t) \cup \mathcal{B}^{\text{S}}(r,t)| \leq \min\{c_r, |\mathcal{E}(r,t) \cup \mathcal{W}(r,t)|\}$ if $|\mathcal{E}(r,t)| < c_r$. However, we require that *total idleness* is avoided where no activity is in process at all at any time t as long as $\mathcal{J}(t) \neq \emptyset$.

The fact that activities may be preempted and rescheduled any time leads in principle to an infinite space of scheduling policies. Fortunately, according to the next theorem, we can restrict our considerations without loss of optimality to decision times on arrivals or activity completions.

Theorem 2.2.2. *For exponentially distributed interarrival times and activity durations there exists a globally optimal preemptive scheduling policy π^* that considers the system only on arrival of a new project or completion of an activity.*

Proof. The assertion follows directly from the memory-less property of the exponential distributions for the interarrival times and activity durations. As long as no

[1] Cf. Gross and Harris [54] for the memoryless property of the exponential distribution.

event (arrival of a new project or completion of an activity) occurs the distribution of the remaining time to any event do not change such that there is no benefit of preemptions and rescheduling. □

2.3 Order Acceptance and Capacity Planning Problem

In the context of this problem, we refer to requests from customers before acceptance decisions as *orders* and after acceptance as *projects*. In the following, we describe the relevant aspects of this problem and present assumptions that are made in addition to the assumptions from Sect. 2.1.

2.3.1 Multi-project Environment

For simplicity of the analysis, we consider only a single resource such as the *bottleneck resource* in the system which has capacity 1 in the sense that only one activity can be processed at a time. As at the tactical level no detailed information of the projects are available resources and projects with their activities are considered in an aggregate manner. Thus, resources are typically departments e.g. engineering departments of a company that process larger work packages of a project (cf. Hans [55]).

As information concerning aggregate activities and their precedence relations are obtained from MPP we assume that project types $p \in \mathcal{P}$ become known *after* MPP has been performed.

Furthermore, we assume that for each project only a single activity is to be processed on the bottleneck resource such that, in the following, we refer to projects instead of activities. Furthermore, we again have the assumption that the duration of a project of type $p \in \mathcal{P}$ is exponentially distributed with rate μ_p while the mean duration is $\overline{d}_p = \frac{1}{\mu_p}$. In this context, holding cost w_p per time unit the project is in the system serve as a approximation for the cost incurred due to not meeting due dates or due to a delayed completion.

In addition, we define general project types $\phi \in \Phi$ and assume that each project type p is of one general project type ϕ_p as shown in Fig. 2.2.
We assume that ϕ_p is known *before* MPP has been fully performed. For example, it is known without further analysis that a customer requests to develop a new product from scratch or modify an existing one (cf. Adler et al. [2]). In the worst case, there exists only a single general project type if no information is known about an order at all before MPP is performed.

The arrival rate for general type ϕ is given by

$$\lambda_\phi = \sum_{p \in \mathcal{P}_\phi} \lambda_p \tag{2.9}$$

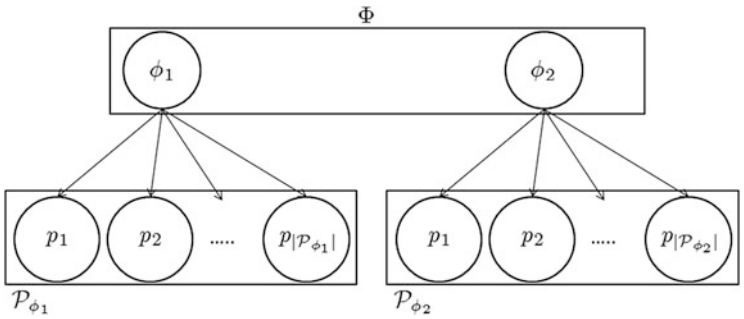

Fig. 2.2 Relationship between general project types and specific project types

where $\mathcal{P}_\phi = \{p \in \mathcal{P} | \phi_p = \phi\}$ is the set of project types $p \in \mathcal{P}$ that are mapped onto a general project type $\phi \in \Phi$.

2.3.2 Order Acceptance Decisions

Any order arriving at the system may be *accepted* or *rejected*. In order to assess whether acceptance is beneficial more detailed information about a project may be taken into account. The information is determined in the MPP step (cf. Hans [55]) subsequent to an arrival. In our model, this corresponds as outlined in Sect. 2.3.1 to determining the specific project type $p \in \mathcal{P}$. As performing MPP requires resources costs such as labor cost are incurred during this step. Furthermore, costs may also involve price reductions due to the fact that a customer has to wait for an acceptance/rejection decision. For simplicity, we assume the costs to be fixed such that a fixed cost $k^{\mathrm{M}}_{\phi_p}$ is incurred for MPP.

We consider the flexibility of doing MPP regularly before OA or postponing MPP to a time after OA. In case of postponed MPP only the general project type ϕ is known for the assessment of a order. However, before making capacity planning decisions, postponed MPP is done at fixed cost $k^{\mathrm{PM}}_{\phi_p}$. As customers typically expect quick responses the MPP step is expected to be crashed in the sense of reducing its processing time at the expense of more resources (cf. Kelley and Walker [68]) if it is performed before OA. Thus, higher costs due to overtime or price reduction to the customer for obtaining a postponed decision may be incurred such that it is realistic to assume that $k^{\mathrm{M}}_{\phi_p} > k^{\mathrm{PM}}_{\phi_p}$. Figure 2.3 illustrates the idea by showing the processes for both types of OA decisions. Below each picture the information (specific project type p or general project type ϕ) that becomes known at different times of each process is shown. For ease of modeling, we assume that we can ignore the time needed for MPP as it is short relative to the duration of the project or the interarrival times. Thus, the option to postpone MPP after to a time after OA helps to save costs for three reasons. Firstly, in case the order is rejected the effort for

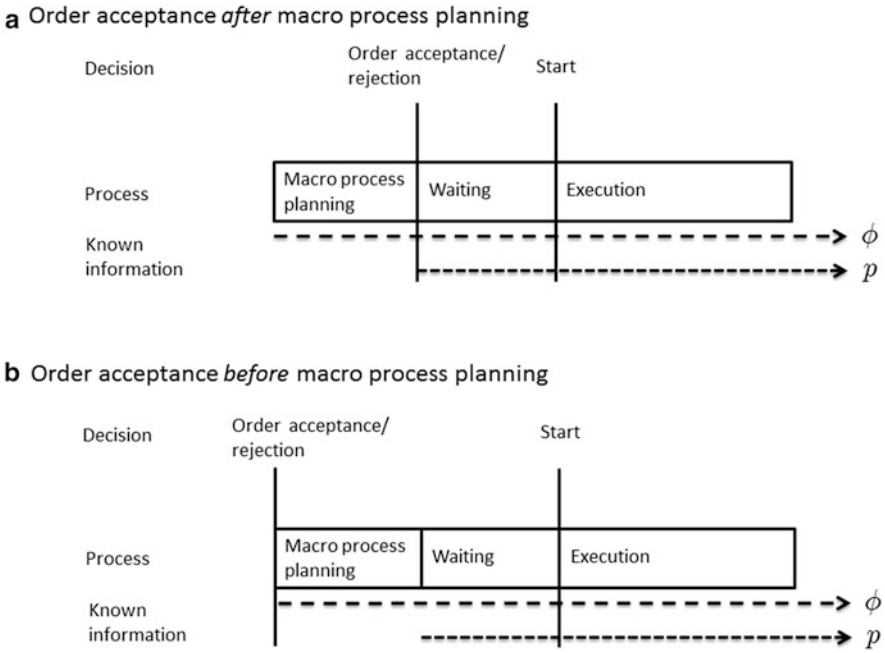

Fig. 2.3 Alternative processes depending on the type of OA decisions

MPP to obtain p is lost. Secondly, MPP can be done using less resources at lower cost. Finally, the customer has a response in less time. This, justifies a lower cost $k_{\phi_p}^{\mathrm{PM}}$ when performing MPP after OA. However, on the downside postponing MPP may lead to worse cost estimates or less reliable due dates. This is implicitly taken into account by the fact that when postponing MPP order acceptance decisions are only based on the general project type known without fully performing MPP. In this thesis, we refer to the first option of doing MPP before making the OA decision as *regular* MPP and the second option as *postponed* MPP.

2.3.3 Resource Allocation Decisions

In order to simplify analysis, we assume that idleness of the resource is not allowed as long as there are waiting projects. Hence, on completion of a project, we schedule a new project for execution from the waiting projects. Furthermore, we assume that only one project may be processed at a time and that the project in process may not be preempted. The latter requirement is realistic as preemptions may have a number of disadvantages which render preemptions detrimental to system performance. *Multi-tasking* may be the consequence which is well known to lead to an increase

of overall flow times of projects (cf. Goldratt [52]). Furthermore, setup times/costs may occur (cf. Anavi-Isakow and Golany [3]).

As we consider the problem at the tactical level, we assume that *non-regular* capacity (cf. Hans [55] or Herbots et al. [57]) is available. Non-regular capacity may be used for example to process multiple projects at a time (cf. Herbots et al. [102]) or to *crash* a project in the sense that it is processed in less time (cf. Kelley and Walker [68]). In this thesis, we consider the second alternative. Be δ^{C} a continuous variable with $0 \leq \delta^{\mathrm{C}} \leq 1$ denoting the share of non-regular capacity used in order to reduce the duration of the current project in process or the project to be scheduled. By increasing the share of non-regular capacity, we assume that the service rate is increased in a linear way such that $\mu_p(1 + z_p\delta_p^{\mathrm{C}})$. z_p is the maximum increase of the service rate in case of full usage of non-regular capacity. If a project is processed using non-regular capacity additional costs $w^{\mathrm{C}}\delta^{\mathrm{C}}$ incurred per time unit a project is processed in a crashed fashion. Crashing costs per time unit that are linear in the usage of non-regular capacity may occur if non-regular capacity comes from overtime. This is common in many areas such as service organizations (cf. Easton and Rossin [44] or McManus [92]). We assume that the usage of non-regular capacity may be dynamically changed any time which is realistic for non-regular capacity resulting from overtime.

Finally, we assume that MPP does not make use of the (bottleneck) resource considered for the execution of projects. Thus, MPP may be performed in a crashed fashion before OA although all non-regular capacity of the resource is in use for crashing the project in process.

Chapter 3
Literature Review

3.1 Dynamic Programming and Approximate Dynamic Programming

Dynamic programming is a general technique for solving sequential problems. The first comprehensive books on the topic have been written by Bellman [13] and Howard [62]. The most important methodologies for determining an optimal policy for a Markov decision process (MDP) are *backward induction*, *value iteration* (VI), *policy iteration* (PI) and *linear programming*. As MDPs are in discrete time where transitions have the same deterministic durations many results and methodologies, such as VI, cannot be directly applied to continuous-time Markov decision processes with exponentially distributed transition times. Thus, to circumvent this issue, Lippman [88] has proposed *uniformization* that involves a transformation of a given continuous-time Markov decision process (CTMDP) into an equivalent CTMDP that corresponds formally to a discrete time MDP.

Another major problem is the fact that the state space becomes so large that applying dynamic programming methodologies becomes intractable due to memory requirements and computational burden. This phenomenon is known as *curse of dimensionality* (cf. Powell [105]), which may, in addition to the state space, also refer to the set of alternative decisions (*action space*) or the set of possible events related to the transition to other system states (*outcome space*).

To remedy the curse of dimensionality, a number of approaches summarized under *approximate dynamic programming* (ADP), have been considered. A central approach to coping with the size of the state space is to approximate the value function via a function having a more compact representation, which is also referred to as *approximation architecture*. Approximation architectures depend on the state variables and have free parameters to be adjusted such that a good approximation of the value function is obtained.

The approaches to determine the free parameters of the approximation architecture can be roughly divided into simulation-based approaches and approaches that

P. Melchiors, *Dynamic and Stochastic Multi-Project Planning*, Lecture Notes in Economics and Mathematical Systems 673, DOI 10.1007/978-3-319-04540-5_3,

are not based on simulation. Simulation based approaches are often referred to as *reinforcement learning* (cf. Sutton and Barto [123]) or *neuro-dynamic programming* (cf. Bertsekas and Tsitsiklis [17]). The basic idea is to simulate the system while observations (e.g. costs or rewards incurred on a sample path) are used to generate observations of the true but unknown value function. Those observations are used to obtain approximations of the value function via stochastic approximation methodologies. A number of algorithms have been proposed which differ, on the one hand, in the way observations of costs or rewards are used in order to generate observations of the value function and, on the other hand, when a policy is improved. In order to obtain observations of the value function the costs or rewards incurred on a sample path may be directly used or combined with the approximation obtained so far, which is done in *temporal difference learning*. Then, one option to obtain an approximation is to perform a least square fit (which corresponds to linear regression in case of linear approximation architectures) of the approximation architecture to the observations of the value function. Furthermore, the approximation may be *learned* in this fashion, without changing the policy, as done in *approximate policy iteration* (cf. Bertsekas and Tsitsiklis [17]). Alternatively, the policy may be changed while the approximation is learned as done in *optimistic approximate policy iteration* (cf. Bertsekas and Tsitsiklis [17]) or *Q-learning* (cf. Sutton and Barto [123]). Typical approximation architectures in this context are neural networks or linear functions, where the approximation architecture depends on the free parameters in a linear fashion. For further details, we refer to Bertsekas and Tsitsiklis [17] or the more recent textbook by Powell [106]. One major advantage of simulation-based approaches is the fact that no model of the system is needed which delivers the distribution of the transition times as well as the transition probabilities. However, on the downside the computational burden for obtaining a good approximation of the value function may be large (cf. Meyn [93]).

From the approaches that are not based on simulation, *approximate linear programming* (ALP) is well known. ALP was originally proposed by Schweitzer and Seidman [115] and is based on the formulation of an MDP as an equivalent *linear programm* (LP). However, the LP still suffers the curse of dimensionality since for each state there is a decision variable representing the value of the value function. Thus, the value function is replaced by a linear approximation architecture such that the number of decision variables is reduced. However, the number of constraints remains large, which may be resolved by constraint reduction (cf. Veach [130]), constraint sampling (cf. De Farias and Van Roy [36]) or column generation on the primal (cf. Veatch and Walker [131]) or on the dual (cf. Adelman [1]) of the LP. De Farias and Van Roy [35] establish theoretical results for the ALP approach with discounted cost and suggest to use *state relevance weights* to control the precision of the approximation at given states. De Farias and Van Roy [34] introduce the ALP approach with state relevance weights for the average cost case.

Another approach which is not based on simulation is the minimization of the *Bellman error* (cf. Bertsekas and Tsitsiklis [17]) where the free parameters of the approximation architecture are determined such that the violation of the evaluation equations is minimized. The approach has been successfully applied especially for

the control of queueing networks such as in Koole and Pot [77] for routing decisions in a call center or in Roubos and Bhulai [113] for the control of admissions or service rates.

3.2 Project Scheduling

We categorize the literature on project scheduling according to the two criteria: Project arrivals and information on project data. With respect to project arrivals, we distinguish between the *static* case, where all projects are available and can be started at the beginning of the time horizon and the *dynamic* case, where projects arrive over time. With respect to the information on project data, we distinguish between the deterministic case where all project data, such as arrival times, durations and resource demands, are deterministic and the stochastic case where (some) project data are stochastic.

3.2.1 Static–Deterministic Project Scheduling

In the static-deterministic multi-project scheduling problem, there are a number of projects which have to be scheduled. Each project is available at the beginning of the planning horizon and all data are deterministic.

Optimal schedules for the static-deterministic case with a single project are mostly based on formulations of the *resource-constrained project scheduling problem* (RCPSP). An early formulation of the problem as a linear program has been proposed by Pritsker et al. [107] for different objective functions such as the minimization of the makespan or, in case of multiple projects, the minimization of the sum of flow times for all projects. If the makespan is used in case of multiple projects, they are considered as a single project such that the makespan corresponds to the longest flow time. This approach is also known as *single project approach*. In order to determine an optimal schedule, which consists of the start times for the activities, a number of more refined models and approaches (e.g. branch and bound algorithms) have been proposed. For a survey, we refer to Brucker et al. [23] or Demeulemeester and Herroelen [39]. However, the size of the problems that can be solved to optimality is limited as finding an optimal schedule has been shown to be NP–complete (cf. Demeulemeester and Herroelen [39]).

Thus, different heuristic approaches have been considered. One of the most popular approaches due to very short computation times is to combine priority rules with a *scheduling scheme* for generating schedules of good quality. For the static and deterministic case, two scheduling schemes are most common. The *serial* scheduling scheme has been proposed by Kelley [67]. It iterates over the activities where in each iteration the first activity from a list sorted according to the priorities is selected and scheduled as early as possible while precedence and

capacity constraints are taken into account. The *parallel* scheduling scheme iterates over the decision times that correspond to completion times of activities already scheduled. In each iteration, as many activities as possible are scheduled at a time. For the single project case, priority rules have been tested, for example, by Davis and Patterson [32], Boctor [20] and Kolisch [74]. Further references can be found in Herroelen et al. [60].

For multi-project scheduling a number of papers are focused on the performance of priority rules. For the objective of minimizing the total weighted project delay, Kurtulus and Davis [81], Kurtulus and Narula [82] and Tsai and Chiu [128] undertook computational studies where they tested the performance of different priority rules. Lawrence and Morton [85] have proposed a family of priority rules that are based on the idea of the *bottleneck dynamics (BD)* approach (cf. Morton and Pentico [96]) for the minimum total weighted tardiness objective. The basic idea is to take into account opportunity cost due to the fact that scheduling activities leads to delay of other activities.

3.2.2 Dynamic–Deterministic Project Scheduling

Dynamic-deterministic multi-project scheduling has been addressed intensively for the special case of the dynamic job shop scheduling problem. For an overview of this, we refer to Ramasesh [109] and Kemppainen [69]. Vepsalainen and Morton [132], Anderson and Nyirenda [4] and Kutanoglu and Sabuncuoglu [83]. They undertook computational studies on the performance of priority rules for minimizing the total weighted tardiness of jobs where the parallel scheduling scheme was used.

3.2.3 Static–Stochastic Project Scheduling

In case of stochastic activity durations, scheduling decisions have to meet the *nonanticipativity constraint* (cf. Fernandez et al. [48]) which states that only information that has become known up to decision time t may be used. Thus, it is not possible to compute a baseline schedule based on the realizations for the activity durations in advance.

For project scheduling under uncertainty, essentially five approaches have been considered in the literature (cf. Herroelen and Leus [59]) that are *proactive scheduling*, *reactive scheduling*, *stochastic scheduling*, *fuzzy scheduling* and *sensitivity analysis*. In case of proactive scheduling a *proactive* baseline schedule is computed where temporal buffers are used to protect it from stochastic variations of activity durations. Reactive scheduling refers to revisions of the proactive schedules in case of unforeseen events such as an activity having a longer duration than planned. In case of stochastic scheduling, *scheduling policies* are used that define for each (not a priori known) decision time the activities which have to be scheduled. Fuzzy

scheduling considers fuzzy start times and completion times of the activities and sensitivity analysis tries to identify a conditions where a given schedule remains optimal.

As already justified in Sect. 1.2, we only consider in this thesis only stochastic scheduling based on scheduling policies.

The literature on the static-stochastic project scheduling problems mostly assumes, for the case of a single project, stochastic activity durations while all other parameters are deterministic. For the case with unlimited resources in the sense that unlimited numbers of activities can be processed at a time, Kulkarni and Adlakha [79] assumed exponentially distributed activity durations and proposed an evaluation model based on a continuous-time Markov chain (CTMC) which allows to numerically compute different moments of the makespan distribution. The approach has been denoted as *Markov program evaluation and review technique networks* (Markov PERT networks). Lee and Suh [86] extended Markov PERT networks to capture more general networks typical for product development processes where precedence relations may be stochastic such that iterations are possible. Obviously, as expected makespan minimization is trivial for unlimited resources (activities are simply started as soon as they become ready according to the precedence constraints) those models do not consider the optimization of scheduling decisions. However, this is no longer the case for the expected net present value (NPV) objective where delaying activities may be beneficial due to discounted costs. At the same time, the payoff obtained at the end of the project may be reduced due to discounting such that there is a tradeoff. Buss and Rosenblatt [24] have shown how the expected NPV can be determined for a Markov PERT network. In addition, they investigated the effect of delaying an activity on the expected NPV. Sobel et al. [120] have presented continuous-time Markov decision processes (CTMDPs) for computing an optimal scheduling policy which maximizes the expected net present value (NPV) of a project given by a Markov PERT network. Creemers et al. [31] have proposed algorithmic improvements allowing the solution of problem instances with larger state spaces.

For the case with limited resources and generally distributed activity durations, a number of classes of scheduling policies have been proposed. As typical objective, the expected makespan has to be minimized. An overview and a classification of the classes scheduling policies is given in Möhring et al. [95] and Stork [122]. The two most widely used scheduling policies for heuristics are *activity–based priority policies* (ABPs) and *resource-based priority policies* (RBPs). Stork [122] has shown that RBPs are from a theoretical point of view disadvantageous because, when interpreted as a function which maps a vector of activity durations into a vector of activity start times, they are neither monotone nor continuous. However, recent computational results have shown that RBPs lead to solutions with a smaller expected makespan (cf. Ballestin and Leus [11] and Ashtiani et al. [5]).

Resource-constrained multi–project scheduling in *static and stochastic* environments has been considered so far mostly in the context of a single resource (or machine) having capacity 1 and projects require exactly one capacity unit. Thus, projects must be processed sequentially. One class of policies that has been

shown to be optimal for many problems in this context is the class of *priority index policies* (cf. Nino-Mora [100]). A priority index policy assigns projects, competing for the resource, a *priority index*, which is used for prioritizing them. The priority index of a project has the property that it depends only on its state at the time of the decision and is independent from the states of other projects. A well known class of problems where priority index policies have been shown to be optimal are *multi–armed bandit problems* (MABs) (see Nino-Mora[100] or Gittins and Jones [51] for a description). Kavadias and Loch [66] consider scheduling of multiple projects at a single bottleneck resource under more general assumptions such that the scheduling problem is no longer a MAB. However, they found that under certain conditions the optimal policy is still a priority index policy. Choi et al. [26] have formulated a discrete time Markov decision process (MDP) for scheduling of chemical engineering projects to be processed on multiple resource types. The MDP takes into account a range of sources of uncertainty. For example, in addition to activity durations, success probabilities of activities may be stochastic. As state spaces become extremely large they use simulation based approaches of approximate dynamic programming (ADP) for obtaining near optimal scheduling policies w.r.t. expected total reward.

3.2.4 Dynamic–Stochastic Project Scheduling

The seminal paper which showed the practical relevance of modeling a dynamic–stochastic multi–project environments as a queueing network is Adler et al. [2] who investigated the performance of R&D–projects in terms of flow times for a chemical company. There, and later in Levy and Globerson [87], a multi-project organization is modeled as a *processing network* where projects of different types arrive over time and each activity of a project has to be processed by a function (resource) of a specific type. Projects of a given type typically share the same precedence relations, interarrival time distributions and distributions for the activity durations (cf. also Chap. 2). Activities are queued in front of the resources of the specified types in order to be processed. Processing networks are closely related to classical queueing networks. However, a major difference to classical queueing networks is the fact that projects or jobs which flow through the network may *fork* in the sense that activities may be processed in parallel or *join* in the sense that multiple activities that may be processed in parallel must have been completed before their common successors may be started. The investigation is done via simulation where, however, only the FCFS–priority policy has been used for scheduling.

From a theoretical point of view such *fork-join–processing networks* have already been subject to research before. A number of papers consider approaches to estimate the distribution of flow times where typical assumptions are that each resource type processes at most one activity type of a project type and a FCFS-priority policy is used for scheduling. Furthermore, there is a single resource of each resource type. Heavy traffic analysis (where the utilization of the resources is close to 1) has been

applied by Nguyen [98] for a single project type in order to obtain approximations e.g. of the expected flow time. Nguyen [99] has extended the analysis to the case of multiple project types. For the case of a single project type with exponentially distributed activity durations, Azaron et al. [6] and Azaron and Modarres [7] developed an approach for determining the flow time distribution. The idea is to transform the original fork-join network in the dynamic-stochastic case referred to as *dynamic PERT network* into a Markov PERT network. For the transformation, the authors have exploited the fact that if there is a single resource or infinitely many resources of a given type the flow time of an activity (waiting time plus processing time) is exponentially distributed. Thus, the flow time distributions of the activities at the resources in the dynamic PERT network replace the distributions of the activity durations in the Markov PERT network. Using phase-type distributions, the idea has been extended to determine approximations for the flow time distribution in case that activity durations follow a general distribution. Although the authors claim that the approach is useful for evaluating scheduling policies no scheduling policies other than FCFS have been considered. Azaron and Tavakkoli-Moghaddam [8] used the approach in order to evaluate decisions on the number of resources which are dedicated to processing an activity type. In their model, the number of resources of a type has an impact on the expected duration of an activity type (the assumption that only one activity may be processed at a time still holds). Yaghoubi et al. [136] have extended the model to the case where CONPIP is used as input control.

One of the few analytical contributions to scheduling in fork-join–processing networks is the paper by Baccelli et al. [9] who have considered non-preemptive as well as preemptive scheduling policies. They introduced the class of *local order preserving policies* and established for its subclasses a number of theoretical results such as the optimality w.r.t. number of projects in the system for FCFS-policies (*not* FCFS-priority policies as considered in this thesis). However, as the classes may contain large numbers of alternative policies optimal policies are provided only for special cases.

A number of simulation studies have assessed different control policies for the framework of Adler et al. [2]. Anavi-Isakow and Golany [3] undertook a simulation study in order to assess the performance of two input control policies, constant number of projects in process (CONPIP) and constant time in process (CONTIP), jointly with priority policies for scheduling activities waiting in the resource queues. Cohen et al. [28] employed the *critical chain* approach originally developed by Goldratt [52] to derive a preemptive priority policy for scheduling and compared the latter with priority policies (Due date modified MINSLK by Dumond and Mabert [43], First Come, First Serve) for minimizing the total flow time. The results show that the due date modified MINSLK–policy gives better results than the critical chain approach. This holds in particular for systems with a high utilization. Although the investigations by Anavi-Isakow and Golany [3] and Cohen et al. [28] deliver some indications on which priority policies should be used, their investigations suffer from the fact that both restrict their considerations to only one toy example.

A special case of multi-project scheduling in a dynamic-stochastic context is scheduling in classical queueing networks where projects do not fork and join. For the special case of queueing networks consisting of a single resource, Cox and Smith [29] have shown that the $c\mu$-policy is optimal for a $M/G/1$-system with different project types and an average cost per unit of time objective. The $c\mu$-policy is a priority index policy that prefers projects with the highest value of $c\mu$ where c denotes the holding cost per unit of time of a project type and μ is the service rate for the respective project. Klimov [72] has shown that an priority index policy is optimal for the more general $M/G/1$-system with feedback. Feedback refers to the fact that projects may revisit the system after being served. Before revisiting the system they may change their type with a given probability,

For more general queueing networks with multiple resource types, optimal scheduling policies can in principle be obtained via *stochastic dynamic programming* (cf. Sennott [118] or Meyn [93]). However, one major obstacle is the *curse of dimensionality* which primarily refers to the size of the state space. One option to remedy this issue is using approximate dynamic programming (ADP) where the value function is approximated by a compact functional form also referred to as *approximation architecture* (for references cf. also Sect. 3.1). Successful applications of ADP to scheduling of queueing networks can be found in De Farias and Van Roy [35], Veach [130] and Vyzas [133] .

The important issue of system stability has been considered by different authors. Recall that stability implies that the long term average number of projects in the system is finite (cf. also Sect. 2.1). However, Kumar and Seidman [80] have shown that instability may occur for some scheduling policies even if the utilization per resource is strictly smaller than one. For further details on stability, we refer to Meyn [93] and to Kumar and Seidman [80].

The only work which considers the computation of optimal policies in the field of dynamic-stochastic multi-project scheduling is the one by Choi et al. [27] who extended the model by Choi et al. [26]. They take into account the arrivals of a limited number of projects in the future. However, no long term optimal policies have been determined.

3.3 Capacity Planning

Models dedicated to tactical *rough cut capacity planning* (RCCP) comprise beside scheduling decisions also decisions related to usage of resource capacities. RCCP in static and deterministic environments have been considered for example by De Boer [33] and Hans [55]. In their models, additional capacities may be used in order to process more project activities at a time or reduce the duration of an activity. The latter usage has been also referred to as *crashing* that has been considered by a different authors such as Kelley and Walker [68], Berman [14] as well as Roemer and Ahmadi [110].

In dynamic-stochastic environments crashing of activities is related to control of services rates. Crabill [30] investigated the optimal control of service rates for an $M/M/1$-system and has found that an optimal policy has multiple thresholds. If a threshold w.r.t. the number of projects in the system is exceeded the service rate is switched to the next higher service rate.

3.4 Order Acceptance

Order acceptance problems have been considered in static as well as in dynamic environments. In static environments, a set of projects not yet started is given at the beginning of the planning horizon. From this set, a subset has to be selected according to an objective w.r.t. a set of constraints. Hence, order acceptance is mostly referred to as *selection* in static environments. Models based on mathematical programming have been proposed by Bard et al. [12] or Loch et al. [90] while Loch and Kavadias [89] have presented a more aggregate model considering the problem from a financial perspective without explicit consideration of projects as discrete items.

Joint optimization of order acceptance and scheduling decisions in the static context has been considered by Slotnick and Morton [119], Talla Nobibon et al. [124] and Talla Nobibon and Leus [125] who have developed exact and heuristic algorithms for the solution.

Order acceptance problems for dynamic environments have been considered in different fields. In multi-project planning, different models exist that are based on the idea of the dynamic stochastic knapsack problem (DSKP). Different variants of the DSKP have been considered by Ross and Tsang [111], Kleywegt and Papastavrou [70] and [71]. The basic idea is as follows. We have a resource of limited capacity and items that dynamically arrive with stochastic interarrival times as well as a demand of varying size for the capacity. On arrival, items may be accepted or rejected. If all resources are in use the item must be rejected such that no queueing is allowed.

Perry and Hartman [102] consider the order acceptance problem with one resource for multiple periods and projects that consist only of a single activity. As the capacity is limited in each period the problem can be considered as a multi-knapsack problem. Herbots et al. [57] extend the model of Perry and Hartman [102] by allowing more complex resource allocation schemes, which are typical for RCCP. Although both models may take into account stochastic interarrival times only deterministic project durations are considered.

Order acceptance problems with stochastic interarrival times and stochastic project durations (projects consist typically only of a single activity) have been considered in queueing theory. One of the very first papers is the one of Naor [97] which considers order acceptance decisions for a $M/M/1$–system with a single project type. A holding cost is incurred per unit of time a project is in the system and payoffs are obtained for each accepted project. The average reward is to be maximized.

Generalizations of the basic model to general distributions and multiple capacity units have been considered by Yechiali [137, 138], Knudsen [73] and Feinberg and Yang [47]. Feinberg and Yang [47] consider a $M/M/c$–system with multiple project types where holding costs, arrival rates and pay offs obtained on completion depend on the project type while all project types have the same expected duration (service rate). For the systems considered, order acceptance policies are monotone policies in the sense that an order of a certain type is accepted until the number of projects in the system exceeds a certain threshold which may depend on the project type. However, scheduling decisions or decisions w.r.t. resource capacities have not been subject to optimization. Scheduling decisions are typically made using FCFS.

Joint optimization of order acceptance decisions and scheduling decisions in queueing theory has been considered by De Serres [40] and [41]. The author considers a $M/M/1$–system with two project types where the expected durations of the projects depend on their types and preemption of projects already in process is allowed. Based on extensive experimental studies, he found that, for many cases, the optimal policies exhibit a monotone structure w.r.t. order acceptance decisions. However, the author failed to provide formal proofs for the structure of optimal policies. Furthermore, the $c\mu$–policy has shown to be optimal in many cases for making scheduling decisions. However, this is not always the case, as a counter example in De Serres [41] shows. Although the work of De Serres comes closest to our problem it neglects important issues relevant for order acceptance in multi–project organizations. Firstly, no decisions w.r.t. usage of resource capacities are taken into account. Secondly, it is assumed that project related information except the duration is fully known on arrival. The cost for performing MPP before OA are neglected.

The joint optimization of order acceptance and scheduling decisions has been investigated on a heuristic basis. Wester et al. [134] and Van Foreest et al. [129] develop and test via simulation different order acceptance and scheduling heuristics for a system consisting of a single resource that processes one project at a time. The heuristics are characterized by different levels of detail concerning the system information used. The case with multiple resource types as well as projects consisting of multiple activities has been considered by Ebben et al. [45] and Ivanescu et al. [64, 65]. Ivanescu et al. [64, 65] extend, in addition, the investigation also to stochastic activity durations.

Chapter 4
Continuous-Time Markov Decision Processes

In this chapter, we review the fundamentals of *continuous-time Markov decision processes* (CTMDPs) as used in this thesis. Our focus is on CTMDPs as the problems presented in Chap. 2 are considered in continuous time. In Sect. 4.1, we outline the general structure of a CTMDP. In order to evaluate existing policies and obtain optimal policies sets of equations denoted as *evaluation equations* and *optimality equations* need to be solved. In Sect. 4.4, we present the general form of the equations. Afterwards, we present in Sect. 4.5 the concept of *uniformization* which is needed before the methods outlined in Sect. 4.6 can be applied for solving the evaluation and optimality equations. Finally, we address in Sect. 4.7 the implementation of a CTMDP as a computer program.

4.1 General Structure

In the following, we present the basic structure of a CTMDP. As shown in Fig. 4.1, a CTMDP is a sequential decision process where the system is considered at decision times for making decisions. In this thesis, we distinguish between *pre-decision states* s and *post-decision states* $\hat{s}$ (cf. Powell [105]). A *pre-decision state* s is the state immediately before a decision (or action) a is made. For making a decision, only the information available in the pre-decision state s and no information about prior states of the system may be used. Making a decision a refers to the selection of a decision from a set $\mathcal{A}(s)$ of alternative decisions.

After making and implementing a decision a, a *post-decision state* $\hat{s} = \hat{s}(s, a)$ is entered. Note that making and implementing a decision is assumed to be done in no time such that $\hat{s}$ is immediately entered. *Post-decision states* are considered explicitly for different reasons. Powell [105], for example, uses *post-decision states* for simulation-based approaches of approximate dynamic programming. We make use of *post-decision states* for simplifying the generation of the transitions as discussed later in Sect. 7.1.1.3.

P. Melchiors, *Dynamic and Stochastic Multi-Project Planning*, Lecture Notes in Economics and Mathematical Systems 673, DOI 10.1007/978-3-319-04540-5_4,

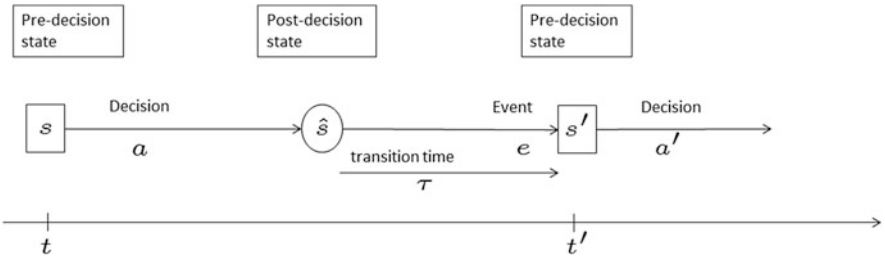

Fig. 4.1 Structure of a CTMDP

Now, from the perspective of the CTMDP the system remains in $\hat{s}$ until an event e occurs which makes a new decision necessary. As a consequence of e, a transition from the system state $\hat{s}$ to a new pre-decision state s' occurs. As events are stochastic the transition from state s to state s' after making decision a occurs with probability $q(s'|s,a)$. Until the occurrence of the event, there is a transition time $\boldsymbol{\tau} = \boldsymbol{\tau}(s,a)$ which is exponentially distributed with rate $\beta(s,a)$. For the transition time from s to s', the fictitious transition s to $\hat{s}$ is not taken into account as it does not need time.

Note that a CTMDP is a special case of a *semi-Markov decision process* (SMDP) (for details cf. Puterman [108]) where the transition times follow a general distribution.

As, for most of the time, we consider pre-decision-states, we refer to pre-decision states s as *system states*. We use the terms pre-decision and post-decision states only where it is necessary to distinguish between both kinds of states.

4.2 Basic Definitions and Relevant Properties

In the following, we give some basic definitions used in the context of MDPs (cf. Puterman[108]).

Definition 4.2.1. The state space is the set $\mathcal{S}$ of all possible system states s.

We will make the definition more specific when discussing the CTMDPs for the dynamic-stochastic multi-project scheduling problem and the problem of order acceptance and capacity planning. In the following, most of our analysis will refer to finite state spaces with $|\mathcal{S}| < \infty$.

As we assume for the multi–project environment that probability distributions are stationary it is sufficient to restrict considerations to *stationary Markov policies*. They are defined as follows.

Definition 4.2.2. A stationary Markov policy π specifies a decision that is independent from decision time t and depends only on the information provided by system state $s \in \mathcal{S}$.

This definition does not imply a loss of optimality when searching for a policy that is globally optimal in the space of all possible policies. It can be shown that there exists a stationary deterministic Markov policy that is globally optimal (cf. Puterman[108]) if the state space $\mathcal{S}$ and the sets of alternative decisions $\mathcal{A}(s)$ $\forall s \in \mathcal{S}$ are finite. Both conditions are met for the cases where we consider optimal policies.

Furthermore, we define the concept of a *sample path* which is used for a part of our analysis.

Definition 4.2.3. A sample path $\omega = (s_1, a_1, \boldsymbol{\tau}_1, s_2, a_2, \boldsymbol{\tau}_2, \ldots, a_{N-1}, \boldsymbol{\tau}_{N-1}, s_N)$ is a sequence of states s_n, decisions a_n and realizations for the transition times $\boldsymbol{\tau}_n$.

Note that on a sample path, it is well possible that a state $s \in \mathcal{S}$ is visited multiple times.

Under a given policy, a CTMDP turns into a *continuous-time Markov chain* (CTMC). In the following, we give some important definitions related to *Markov chains* (not necessarily CTMCs) which can be transferred to the case of MDPs.

Definition 4.2.4. A state $s \in \mathcal{S}$ is recurrent if the time between two visits is bounded with probability 1.

Definition 4.2.5. A state $s \in \mathcal{S}$ is transient if the time between two visits is unbounded with positive probability.

The definition of recurrence can be strengthened by the definition of *positive recurrence*.

Definition 4.2.6. A state $s \in \mathcal{S}$ is positive recurrent if the expected time between two visits is bounded.

The property of recurrence is necessary for the existence of a bounded long term average cost or reward. For finite state spaces ($|\mathcal{S}| \leq \infty$) it is also sufficient (cf. Puterman [108]) whereas for infinite state spaces positive recurrence is needed.

Before we address some properties that are important for the application of the methods used in this work, we give some further definitions characterizing the relationships between states.

Definition 4.2.7. A state $s' \in \mathcal{S}$ is accessible from a state $s \in \mathcal{S}$ ($s \to s'$) if s' can be reached from s with positive probability.

Definition 4.2.8. A state $s' \in \mathcal{S}$ communicates with state $s \in \mathcal{S}$ if $s \to s'$ and $s' \to s$.

Next, we define a closed set (or class) of states.

Definition 4.2.9. A set $\mathcal{C} \subseteq \mathcal{S}$ is closed if no state $s' \in \mathcal{S} \backslash \mathcal{C}$ is accessible from any state $s \in \mathcal{C}$.

Definition 4.2.10. A closed set $\mathcal{C} \subseteq \mathcal{S}$ is irreducible if no subset of $\mathcal{C}$ is closed.

For a finite state space $\mathcal{S}$, it is possible that the set of recurrent states decomposes into a finite number of irreducible closed sets, while, for infinite $\mathcal{S}$, the number of closed sets is possibly infinite. This leads to the following definitions that are relevant for the application of solution methods.

Definition 4.2.11. A MC is unichain if there exists a single irreducible class of recurrent states plus some transient states.

Definition 4.2.12. A MC is multichain if there exist multiple irreducible classes of recurrent states plus some transient states.

Finally, we transfer the definitions of unichain and multichain to MDPs (not necessarily CTMDPs).

Definition 4.2.13. An MDP is unichain if under every deterministic stationary policy there exists a single irreducible class of recurrent states plus some transient states.

Definition 4.2.14. An MDP is multichain if there exists a deterministic stationary policy for which there exist multiple irreducible classes of recurrent states plus some transient states.

The definitions of unichain and multichain have the following implications for the case of CTMDPs. If the CTMC under a given policy is unichain the long term average cost or reward is independent from the starting state $s \in \mathcal{S}$ of a sample path. By contrast, if the CTMC under a given policy is multichain the average cost or reward may vary depending on the starting state $s \in \mathcal{S}$ of a sample path. Therefore, solution methodologies become more involved for multichain CTMDPs. Fortunately, for the problems considered in this thesis, we can restrict our considerations to unichain CTMDPs.

4.3 Objective Function

As we consider the long term average cost or profit per project (cf. Chap. 2) the relevant objective functions for the CTMDP are the average cost per time unit or the average reward per time unit. As the analysis for both objectives is very similar we refer in the general presentation of the MDP related theory only to the average cost objective. Then, the optimal average cost g^* is given by

$$g^* = \min_{\pi \in \Pi} \liminf_{N \to \infty} g(\pi) = \frac{E\left[\sum_{n=1}^{N-1} (c(s_n, \pi(s_n))\boldsymbol{\tau}(s_n, \pi(s_n)) + k(s_n, \pi(s_n), s_{n+1}))\right]}{E\left[\sum_{n=1}^{N} \boldsymbol{\tau}(s_n, \pi(s_n))\right]} \tag{4.1}$$

where $g(\pi)$ is the average cost when policy π is followed. $c(s,a)$ is the cost rate incurrent per time unit subsequent to decision a in system state $c(s,a)$ and $k(s,a,s')$ is the fixed cost incurred on the transition from system state s to system state s' subsequent to decision a.

Recall that if the CTMDP is unichain g^* or $g(\pi)$ are independent from the starting state s_0 of a sample path (cf. Puterman [108]).

4.4 Evaluation and Optimality Equations

We use a CTMDP, discussed so far, for two purposes in the scope of our investigations.

1. Evaluation of a given policy π.
2. Finding an optimal policy π^*.

Evaluation of a given policy π involves determining the value $g(\pi)$ of the objective function. This can be done either by simulation or by finding a solution for the following set of evaluation equations (also referred to as *Poisson's equations*).

$$h(s) = \frac{c(s,\pi(s)) - g(\pi)}{\beta(s,\pi(s))} + \sum_{s' \in \mathcal{S}} q(s'|s,\pi(s)) \left(k(s,\pi(s),s') + h(s')\right) \tag{4.2}$$

For the existence of a unique solution of the unknowns g and $h(s)$ we must have a *unichain* CTMDP with a *finite* state space where $k(s,a,s')$, $c(s,a)$ and $\beta(s,a) > 0$ are bounded. Then, the unique solution can be obtained by setting $h(s'') = 0$ for some state $s'' \in \mathcal{S}$ (cf. Puterman [108]).

The values $h(s)$ are also denoted as *relative value function*. One possible interpretation is that they are the long term difference between the expected total costs when the system is started at state s and the expected total costs when the system is started at state s'' (cf. Tijms [127] or Bertsekas [15]). Thus, $h(s)$ is also denoted as *relative cost* (cf. Bertsekas [16]).

While Eqs. (7.27) are sufficient for the evaluation of a stationary policy π an optimal stationary policy $\pi^* = \underset{\pi \in \Pi}{\operatorname{argmin}}\, g(\pi)$ must deliver a feasible solution for the following set of optimality equations (also referred to as *Bellman equations*).

$$h(s) = \min_{a \in \mathcal{A}(s)} \left\{ \frac{c(s,a) - g^*}{\beta(s,a)} + \sum_{s' \in \mathcal{S}} \left[q(s'|s,a) \left(k(s,a,s') + h(s')\right)\right] \right\} \tag{4.3}$$

4.5 Uniformization

When considering a CTMDP, there is the problem that the transition times after making decision $a \in \mathcal{A}(s)$ in state $s \in \mathcal{S}$ are exponentially distributed with varying expected durations $\frac{1}{\beta(s,a)}$ while many results and methods e.g. value iteration (VI) have been developed for discrete time MDPs with transitions having the same deterministic duration (typically the duration is one time unit). Therefore, we apply *uniformization* originally developed by Lippman [88] (cf. also Puterman [108]). It transforms the original CTMDP into an equivalent CTMDP where the *expected* transition times are equal for all states and decisions. Thus, the uniformized CTMDP corresponds formally to a discrete time MDP although it is still a CTMDP. Note that for the uniformized CTMDP only the expected transition times are equal while the transition times are still exponentially distributed random variables.

As a first step, we fix the *uniformization constant* c such that

$$\sup_{s \in \mathcal{S}, a \in \mathcal{A}(s)} \beta(s,a) < c < \infty \tag{4.4}$$

which leads to $c \geq \max\{\beta(s,a) \forall s \in \mathcal{S}, a \in \mathcal{A}(s)\}$. Next, we consider the transitions after making decision $a \in \mathcal{A}(s)$ in state $s \in \mathcal{S}$. We add to the transitions subsequent to decision a in state s a fictitious transition which directs back to state s. The fictitious transition is related to a fictitious (dummy) event which occur at rate $c - \beta(s,a)$. Thus, the new total rate is c. As the fictitious event does not have any effect on the system state the *uniformized* CTMDP is equivalent to the original one (a formal proof can be found in Tijms [127]). Figure 4.2 shows an extract of the CTMDP for illustrating the idea.For system state s, the additional transitions go from the post-decision state $\hat{s}(s,a)$ (for simplicity of presentation we have dropped the parameters s and a) back to system state s.

Furthermore, we obtain new transition probabilities as given by

$$\tilde{q}(s'|s,a) = q(s'|s,a)\frac{\beta(s,a)}{c} \ \forall s, s' \in \mathcal{S} \tag{4.5}$$

We set

$$\tilde{g}(\pi) = \frac{g(\pi)}{c} \tag{4.6}$$

$$\tilde{g}^* = \frac{g^*}{c} \tag{4.7}$$

$$\tilde{c}(s,a) = \frac{c(s,a)}{c} \tag{4.8}$$

and obtain the following evaluation equations for a $\pi \in \Pi$ which correspond formally to the evaluation equations of a discrete time MDP.

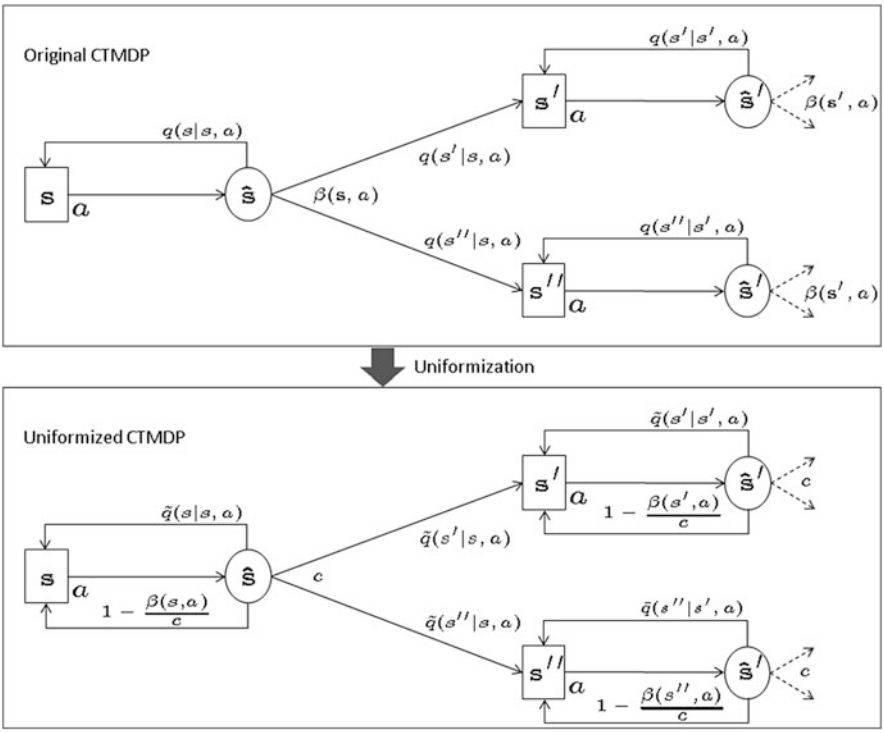

Fig. 4.2 Idea of uniformization

$$h(s) = \tilde{c}(s, \pi(s)) - \tilde{g}(\pi) + \sum_{s' \in \mathcal{S}} \tilde{q}(s'|\pi(s), s) \left(k(s, \pi(s), s') + h(s') \right) + \left(1 - \frac{\beta(s, \pi(s))}{c} \right) h(s) \ \forall s \in \mathcal{S} \quad (4.9)$$

The optimality equations for the uniformized CTMDP are given by

$$h(s) = \min_{a \in \mathcal{A}(s)} \left\{ \tilde{c}(s, a) - \tilde{g}^* + \sum_{s' \in \mathcal{S}} \tilde{q}(s'|a, s) \left(k(s, a, s') + h(s') \right) + \left(1 - \frac{\beta(s, a)}{c} \right) h(s) \right\} \quad \forall s \in \mathcal{S} \quad (4.10)$$

4.6 General Solution Methodologies

For the discussion of the solution methodologies, we use the generic form of the evaluation equations given by

$$h(s) = \frac{c(s,\pi(s)) - g}{\beta(s,\pi(s))} + \sum_{s' \in \mathcal{S}} q(s'|s,\pi(s)) \left(k(s,\pi(s),s') + h(s') \right) \quad \forall s \in \mathcal{S} \tag{4.11}$$

and the optimality equations given by

$$h(s) = \min_{a \in \mathcal{A}(s)} \left\{ \frac{c(s,a) - g^*}{\beta(s,a)} + \sum_{s' \in \mathcal{S}} q(s'|s,a) \left(k(s,a,s') + h(s') \right) \right\} \quad \forall s \in \mathcal{S} \tag{4.12}$$

as a starting point.

4.6.1 Value Iteration

Value iteration (VI) (cf. Puterman [108]) is one of the most commonly used algorithms to solve infinite horizon MDPs. The algorithm iterates over the entire state space $s \in \mathcal{S}$ and updates a set of values $V_n(s)$ being numerical estimates for the value function for all $s \in \mathcal{S}$ in iteration n. As VI requires a discrete time MDP we have to consider the *uniformized* CTMDP. The procedure is given by Algorithm 1. As the CTMDP is *unichain*, we can apply value iteration without modifications. Note that in order to avoid numerical problems, we subtract in Step 11 the value of empty state V_n^0. Hence, the values $V_n(s)$ remain bounded and converge to the relative value function $h(s)$. Therefore, the algorithm is known as *relative value iteration*. Note that standard value iteration and relative value iteration obtain the same policy on convergence.

The numerical estimate g' for g^* is obtained from

$$g' = \frac{c}{2} \left(\max_{s \in \mathcal{S}} (V_n(s) - V_{n-1}(s)) + \min_{s \in \mathcal{S}} (V_n(s) - V_{n-1}(s)) \right) \tag{4.13}$$

which is known to converge to g^* for $n \to \infty$.

4.6.2 Policy Iteration

The idea of *policy iteration* (PI) is to start with given policy π_0 and to perform multiple iterations in order to obtain an optimal policy. Each iteration n comprises an *evaluation step* and an *improvement step*. The evaluation step determines a

Algorithm 1 Unichain relative value iteration

Require: $\mathcal{S}$

for $s \in \mathcal{S}$ **do**

  $V_0(s) \leftarrow 0$

end for

$n \leftarrow 1$

Do

  for $s \in \mathcal{S}$ **do**

$$V_n(s) \leftarrow \min_{a \in \mathcal{A}(s)} \{\tilde{c}(s,a) + \sum_{s' \in \mathcal{S}} \tilde{q}(s'|s,a)(k(s,a,s') + V_{n-1}(s')) + \left(1 - \frac{\beta(s,a)}{c}\right) V_{n-1}(s)\} \quad \forall s \in \mathcal{S}$$

  end for

  $V_n^0 \leftarrow V_n(s^0)$

  for $s \in \mathcal{S}$ **do**

    $V_n(s) \leftarrow V_n(s) - V_n^0$

  end for

  $n \leftarrow n + 1$

Until $\max_{s \in \mathcal{S}}(V_n(s) - V_{n-1}(s)) - \min_{s \in \mathcal{S}}(V_n(s) - V_{n-1}(s)) < \epsilon$

for $s \in \mathcal{S}$ **do**

$$\pi(s) \leftarrow \underset{a \in \mathcal{A}(s)}{\operatorname{argmin}} \{\tilde{c}(s,a) + \sum_{s' \in \mathcal{S}} \tilde{q}(s'|s,a)(k(s,a,s') + V_{n-1}(s')) + \left(1 - \frac{\beta(s,a)}{c}\right) V_{n-1}(s)\} \quad \forall s \in \mathcal{S}$$

end for

solution for the set of evaluation equations (4.12). This can effectively be done using VI where decisions are made based on a given policy. Alternatively, we may also use algorithms for the efficient solution of linear equations e.g. Gauss-Seidel (cf. Defregger [37]) or the Power Series Algorithm (cf. Koole and Pot [78]).

The improvement step (4.14) also known as *one step policy improvement* (cf. Tijms [127]) determines a new policy π_n given the value function $h(\pi_{n-1}, s)$ $\forall s \in \mathcal{S}$ and average cost $g(\pi_{n-1})$ for policy π_{n-1} from the previous iteration by

$$\pi_n(s) = \underset{a \in \mathcal{A}(s)}{\operatorname{argmin}} \left\{ \frac{c(s,a) - g(\pi_{n-1})}{\beta(s,a)} + \sum_{s' \in \mathcal{S}} q(s'|s,a) \left(k(s,a,s') + h(\pi_{n-1}, s')\right) \right\} \quad \forall s \in \mathcal{S} \quad (4.14)$$

$h(\pi_{n-1}, s)$ is the relative function when policy π_{n-1} is followed in state s and afterwards. Thus, the right-hand side of the equation can be interpreted as the relative cost when decision a is selected and policy π_{n-1} is followed afterwards. For the new policy, we must have $g(\pi_n) \leq g(\pi_{n-1})$ (for a proof cf. Tijms [127]).

The idea of policy improvement will also be central in the approximate dynamic programming algorithms discussed in Sect. 7.4. Policy iteration aborts as soon as no further improvement of a policy can be obtained such that $g(\pi_n) = g(\pi_{n-1})$. The procedure is given by Algorithm 2.

The motivation to consider policy iteration is threefold. Firstly, it has advantages in terms of run time and memory requirements. For example, finding the optimal decision for each system state can be very time-consuming but needs to be performed less frequently than in VI. Secondly, it can be implemented more efficiently in terms of run time and memory requirements (for details we refer to Sect. 4.7.2). Thirdly, it serves as a basis for the algorithms of ADP in Sect. 7.4.

Algorithm 2 Unichain policy iteration

Require: $\mathcal{S}, \pi_0$

$n \leftarrow 0$

Do

$n \leftarrow n + 1$

Policy evaluation: Obtain the average cost $g(\pi_{n-1})$ and the relative value function $h(\pi_{n-1}, s)$ $\forall s \in \mathcal{S}$ by solving equations:

$$h(s) = \frac{c(s,a) - g(\pi_{n-1})}{\beta(s, \pi_{n-1}(s))} + \sum_{s' \in \mathcal{S}} q(s'|s, \pi_{n-1}(s)) \left(k(s, \pi_{n-1}(s), s') + h(s')\right) \qquad \forall s \in \mathcal{S}$$

Policy improvement: Chose π_{n+1} to satisfy:

$$\pi_n(s) = \underset{a \in \mathcal{A}(s)}{\operatorname{argmin}} \left\{ \frac{c(s,a) - g(\pi_{n-1})}{\beta(s,a)} + \sum_{s' \in \mathcal{S}} q(s'|s,a) \left(k(s,a,s') + h(\pi_{n-1}, s')\right) \right\} \qquad \forall s \in \mathcal{S}$$

Until $g(\pi_n) = g(\pi_{n-1})$

$\pi^* = \pi_n$

4.7 Implementation

4.7.1 Generation of the State Space

The implementation of the CTMDP and the corresponding solution methodologies has been done using JAVA.

At first we briefly address the data structure, before we briefly address the procedure for generating the state space.

4.7.1.1 Data Structures

The state space $\mathcal{S}$ is represented using an array of objects representing the system states. An object for system state s contains an array for tuples $(\sigma, n(\sigma, s))$ for all project states $\sigma \in \Sigma$ where $n(\sigma, s) > 0$ (for details on the definition of project states σ and system states s we refer to Sect. 7.1). Furthermore, we maintain an array containing objects that represent project states $\sigma \in \Sigma$. Each object representing a project state has multiple arrays with indices for waiting activities, activities in process and the project type.

For a fast retrieval of system states $s \in \mathcal{S}$ and project states $\sigma \in \Sigma$, we use hash tables (cf. Sedgewick [117]) of which the elements are addressed via *hash keys*. In the following, we briefly explain the usage and computation of hash keys only for system states s (for project states σ we proceed in a similar way). A hash key for a system state s is an integer value obtained via *hash function* $H(s)$ which lies in the range between 1 and the size of the hash table. The computation of $H(s)$ is based on the content of a system state or project state object. Figure 4.3 shows the hash table and the array of objects for system states $s \in \mathcal{S}$. A hash key $H(s)$ is used as an index in the hash table of a fixed size containing references. Then, each reference points to the first system state object in the array of system states which has a given hash key e.g. for system state s_1 we store in the hash table 1 being the position of s_1 in the array of system state objects. If no such system state object exists yet as system state s may not have been added yet to the state space we store -1. Note that the hash keys have a smaller length, e.g. 16 bit, than the total set of variables used for representing a system state s (the set of tuples $(\sigma, n(\sigma, s))$) such that there may be more possible combinations of values for the variables of a system state than possible hash keys (in case of a length of 16 bit we have at most 65,536 hash keys). Thus, collisions are possible in the sense that two states may have the same hash key. In the example, we have for s_1 and s_3 $H(s_1) = H(s_2)$. In order to cope with collisions, state objects having the same hash key are linked by storing for each system state s the position $Succ(s)$ of the next state having the same hash key in the array. If there exists no further state with the same hash key we set $Succ(s) = -1$. For example, s_1 is stored at position 1 and system state s_3 at position 3. Thus, we store for s_1 the location of s_3, by setting $Succ(s_1) = 3$, as the next system state with

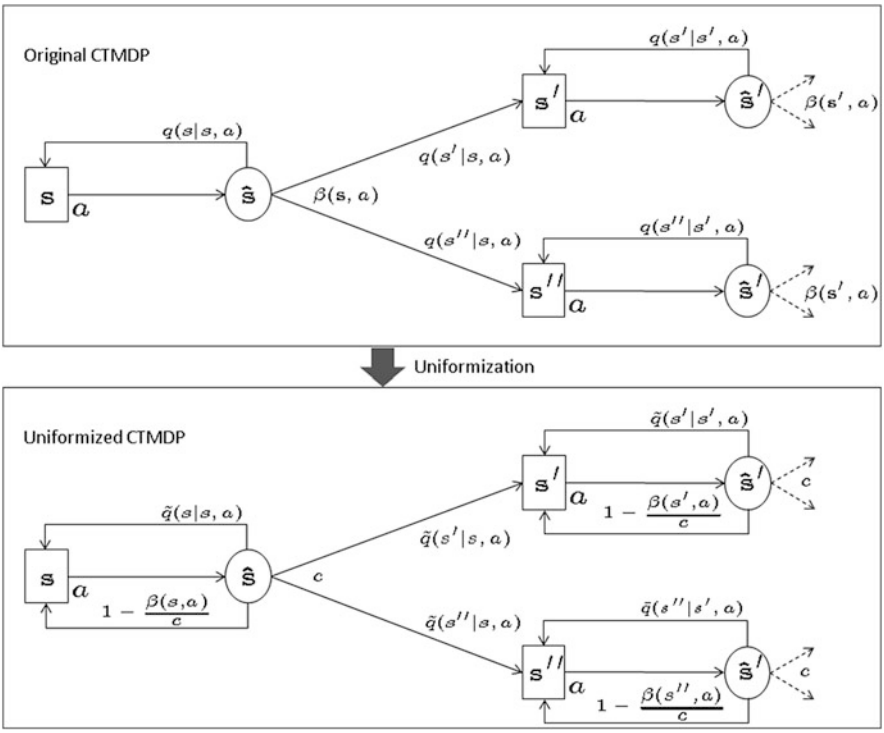

Fig. 4.3 Hash table and array of system state objects

$H(s_3) = H(s_1)$. As there is no further system state object with the same hash key we set $Succ(s_3) = -1$.

Thus, to check whether a given system state s already exists, we proceed along the chain of system states having the same hash key as a given system state s.

4.7.1.2 Generation Procedure

The generation procedure for the state space $\mathcal{S}$ is as follows. At first, we add system state s^0, where the system is empty. Then, we generate all feasible decisions and transitions to successor states of s^0 in the CTMDP. If a successor state does not exist yet it is added to the state space by storing its object after the last object in the array of system state objects and updating the hash table information. Then, we process the array of system state objects in an increasing order and stop as soon as, after considering a system state, no further system states are in the array of system state objects. This is the case if all system states have been checked for successors and no successors not yet in the state space could be generated.

4.7.2 Solution Methodologies

The run time required by VI strongly depends on Step 7. In each iteration n and for each state $s \in \mathcal{S}$ all decisions $a \in \mathcal{A}(s)$ including the transitions to the successor states must be generated and evaluated. Therefore, Step 7 may consume a lot of computational time if a full enumeration of alternative decisions is necessary. The computational burden can be reduced in two ways.

1. Generation and storage of decisions and transitions in advance. However memory consumptions may be tremendously increased.
2. The structure of the Bellman equations can be exploited to avoid full enumeration and evaluation of all decisions $a \in \mathcal{A}(s)$. This has been done for the preemptive scheduling problem in Sect. 7.1.2 or for the order acceptance and capacity planning problem in Chap. 8. The computation burden for generating transitions might be higher than in the first case. However, less computational effort for evaluating decisions is needed and memory consumption is reduced as no decisions and transitions need to be stored.

In order to reduce the computational burden of PI we store the decisions of the current policy π_n as well as the corresponding transitions. Thus, we have, as an advantage of PI, lower memory consumption than for VI as, for each state $s \in \mathcal{S}$, we only have to store the transitions for one decision instead of all decisions. Furthermore, the step of determining optimal decisions $a^* \in \mathcal{A}(s)\ \forall s \in \mathcal{S}$ is done less frequently.

Chapter 5
Generation of Problem Instances

For testing scheduling methodologies in a systematic way, it is important to have a set of problem instances that is representative for the entire space of problem instances. For example, the set of instances of Patterson [101] for the static-deterministic single project scheduling problem has been used by many authors for testing. However, Kolisch et al. [76] have found that the set does not contain cases that are difficult to solve and that problem parameters may have a strong impact on problem complexity in terms of computation times.

Thus, for static-deterministic single–project scheduling problems, also referred to as resource-constrained project scheduling problem (RCPSP), a number of generators for systematic creation of problem instances according to given parameters have been developed by Kolisch and Sprecher [75], Schwindt [116] and Demeulemeester et al. [38]).

For static-deterministic multi-project scheduling problems, a generator has been recently developed by Browning and Yassine [22].

However, for dynamic-stochastic multi-project scheduling problems, as considered in this thesis, no generator is available so far. Most instances (cf. Adler et al. [2] and Anavi-Isakow and Golany [3]) have either been taken from the real world or have been created as toy examples.

Thus, we have developed a procedure by which problem instances of the dynamic-stochastic multi–project scheduling problem can be generated systematically based on specified problem parameters.

As a first step, we address in Sect. 5.1 the generation of project networks, composed of nodes, representing activity types, and arcs, representing precedence relations. Those networks serve as skeletons for the project types in the generation procedure which is presented in Sect. 5.2.

P. Melchiors, *Dynamic and Stochastic Multi-Project Planning*, Lecture Notes in Economics and Mathematical Systems 673, DOI 10.1007/978-3-319-04540-5_5,

5.1 Generation of Project Networks

In this thesis, *activity-on node* (AoN) networks (cf. Demeulemeester and Herroelen [39]) are considered where nodes represent activities and arcs represent precedence relations. The generation is done using the generator *ProgenMax*, which has originally been developed by Schwindt [116] for generating instances of a generalized RCPSP. As we only need nodes and arcs from those problem instances only two parameters are of interest–the *order strength* (*OS*) and the *number of nodes* $|\mathcal{V}|$.

OS is a [0; 1]-normalized summary measure characterizing the network structure and is defined as follows (cf. Schwindt [116]).

$$OS = \frac{\sum_{i \in \mathcal{V}} \sum_{i' \in \mathcal{V}} \delta_{ii'} - |\mathcal{V}|}{|\mathcal{V}|(|\mathcal{V}| - 1)/2} \tag{5.1}$$

$\mathcal{V}$ is the set of nodes and $\delta_{ii'}$ is a binary variable indicating that there exists a path in the AoN–network from node i to i'. In words, *OS* is the fraction of the number of precedence relations (including the transitive ones) between all nodes $i, i' \in \mathcal{V}$ and the maximum number of possible precedence relations.

At the boundaries of the range of values for *OS*, a value of 0 reflects a parallel network without any precedence relations at all (dummy nodes and precedence relations to dummy nodes are ignored) and 1 a serial network with precedence relations between all nodes. Note that *OS* originally has been proposed by Thesen [126] as an easy to compute approximation for the *restrictiveness* of a network. For each combination of *OS* and $|\mathcal{V}|$, we generate N^{nw} sample networks. Table 5.1 summarizes the parameters for the network generation.

5.2 Generation Procedure

The procedure is composed of multiple steps and uses general parameters (cf. Table 5.2), resource type related parameters (cf. Table 5.3) and project type related parameters (cf. Table 5.4).

The steps of the procedure are outlined in Fig. 5.1. Note that for each project type $p \in \mathcal{P}$ an AoN–network $(\mathcal{V}_p(OS_p, n_p^{\mathrm{nw}}), \mathcal{A}_p(OS_p, n_p^{\mathrm{nw}}))$ is given that has been generated in advance according to Sect. 5.1. For each problem instance (of the N^{PI} instances to be generated), the following steps are carried out.

5.2.1 Step 1: Assignment of Activity Types to Resource Types

In the first step, we determine, for each resource type $r \in \mathcal{R}$, the set $\mathcal{V}_{pr} = \{i \in \mathcal{V}_p | r_{ip} = r\}$ of activity types (i, p) to be processed by resource type r. We randomly

Table 5.1 General parameters for base instance generation

Parameter	Definition
OS	Order strength
$\|\mathcal{V}\|$	Number of nodes
N^{nw}	Number of networks to be sampled for given OS and $\|\mathcal{V}\|$

Table 5.2 General parameters for base instance generation

Parameter	Definition
$\lambda^{\max}$	Maximum total arrival rate
N^{PI}	Number of problem instances to be generated
sd	Seed for the random numbers used in the generation process

Table 5.3 Resource type related parameters for base instance generation

Parameter	Definition
c_r	Number of resources of type r
$CV^{\overline{d},\min}$	Minimum value for the coefficient of variation related to expected durations
$CV^{\overline{d},\max}$	Maximum value for the coefficient of variation related to expected durations
u	Utilization per resource

Table 5.4 Project type related parameters for base instance generation

Parameter	Definition
α_p	Percent of tardy projects of type p
a_p	Fraction of the total arrival rate
ϵ^{wf}	Tolerance parameter for workload proportions
OS_p	Order strength of the network of project type p
n_p^{nw}	Number of the network sample of project type p
ν_r	Utilization multiplier of resource type r
$\|\mathcal{V}_{pr}\|$	Number of activity types (i, p) to be processed by resource type r
wf_p	Workload coefficient of project type p
w_p	Weight of project type p

select a node $i \in \mathcal{V}_p$ that has not been assigned to some resource type yet and set $r_{ip} = r$ until the specified cardinality $|\mathcal{V}_{pr}|$ is reached for $\mathcal{V}_{pr}$.

5.2.2 *Step 2: Determination of Expected Durations of the Activity Types*

In the second step, we determine for all activity types (i, p) the expected duration $\overline{d}_{ip}$. At first, we set the total arrival rate $\lambda = \lambda^{\max}$ where we have, for the

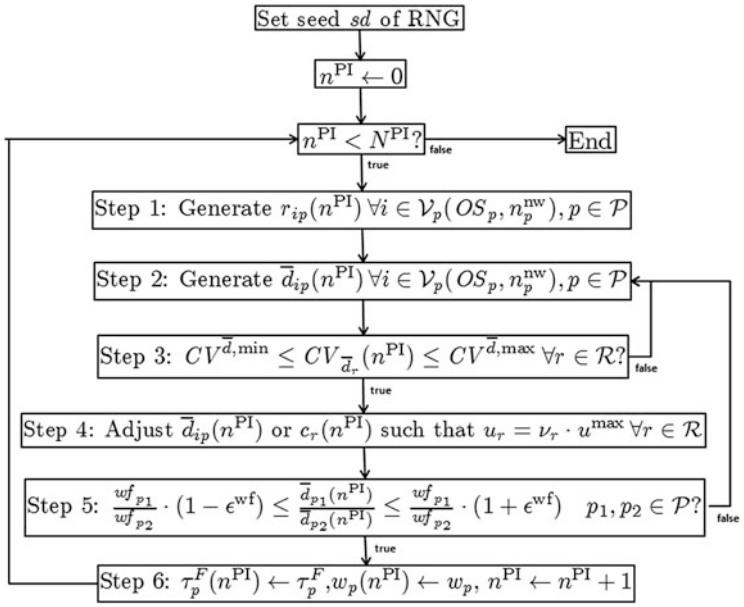

Fig. 5.1 Generation procedure

arrival rate of any project type $p \in \mathcal{P}$, $\lambda_p = \lambda^{\max} \cdot a_p$. a_p is the fraction of the total arrival rate that is due to project type p. For the generation of expected activity durations, we make the preliminary assumption that all resources have the same level of utilization such that

$$u_r = \frac{\rho_r}{c_r} = 1 \ \forall r \in \mathcal{R} \tag{5.2}$$

ρ_r refers to the traffic intensity as defined by (2.3) which can be interpreted as the amount of work that arrives per time unit at resource type r (cf. Brémaud [21]). Later, we discuss two options to achieve utilizations that are specific to resource types.

Next, we draw, for each activity type (i, p), a random real number $\tilde{d}_{ip}$ from $[0; 1]$. In order to obtain the values for $\overline{d}_{ip}$ we must scale the values for $\tilde{d}_{ip}$ as follows. As $u_r = 1 \ \forall r \in \mathcal{R}$ the traffic intensities are given by $\rho_r = c_r$. Be $\rho_{ip} = \lambda_p \cdot \overline{d}_{ip}$ the traffic intensity due to activity type (i, p). Then, for all activity types (i, p) with $i \in \mathcal{V}_{pr}$, we must have

$$\sum_{p \in \mathcal{P}} \sum_{i \in \mathcal{V}_{pr}} \rho_{ip} = \rho_r = c_r \tag{5.3}$$

Be $\tilde{\rho}_r$ and $\tilde{\rho}_{ip}$ the traffic intensities when we replace $\overline{d}_{ip}$ by $\tilde{d}_{ip}$. By scaling $\tilde{\rho}_{ip}$ such that $\rho_{ip} = \tilde{\rho}_{ip}\frac{\rho_r}{\tilde{\rho}_r} = \tilde{\rho}_{ip}\frac{c_r}{\tilde{\rho}_r}$ condition (5.3) is met. From the definition of ρ_{ip}, we obtain the expected durations by

$$\overline{d}_{ip} = \tilde{d}_{ip}\frac{c_r}{\tilde{\rho}_r} \tag{5.4}$$

To attain the desired level of utilization, such that $u_r = u \ \forall r \in \mathcal{R}$, we set $\lambda = \lambda^{\max} \cdot u$ and $\lambda_p = \lambda \cdot a_p$.

5.2.3 Step 3: Variation Check of the Expected Activity Durations

In the third step, we check, for each resource type $r \in \mathcal{R}$, the variation of the expected durations of the activities to be processed. In order to measure the variation we introduce random variable $\overline{\boldsymbol{d}}_r$ for the expected duration of any activity arriving (becoming ready for execution) at resource type r. Then, the variation of the expected durations is measured by the coefficient of variation $CV_{\overline{\boldsymbol{d}}_r}$ of $\overline{\boldsymbol{d}}_r$. For the computation of $CV_{\overline{\boldsymbol{d}}_r}$, we make the simplifying assumption that the types of activities arriving at resource type r are *identically and independently* distributed.[1] Thus, the probability that any arriving activity at resource type r is of type (i, p) is given by $\frac{a_p}{a_r}$ where $a_r = \sum_{p\in\mathcal{P}} a_p|\mathcal{V}_{pr}|$. a_r can be interpreted as the expected number of activities to be processed by resource type r due to a project of any type having entered the system. Now, $E[\overline{\boldsymbol{d}}_r]$ and $Var[\overline{\boldsymbol{d}}_r]$ are given by

$$E[\overline{\boldsymbol{d}}_r] = \frac{1}{a_r}\sum_{p\in\mathcal{P}} a_p \sum_{i\in\mathcal{V}_{pr}} \overline{d}_{ip} \tag{5.5}$$

$$Var[\overline{\boldsymbol{d}}_r] = \frac{1}{a_r}\sum_{p\in\mathcal{P}} a_p \sum_{i\in\mathcal{V}_{pr}} \overline{d}_{ip}^2 - E[\overline{\boldsymbol{d}}_r]^2 \tag{5.6}$$

Thus, we obtain for the squared coefficient of variation $CV^2_{\overline{\boldsymbol{d}}_r}$

$$CV^2_{\overline{\boldsymbol{d}}_r} = \frac{Var[\overline{\boldsymbol{d}}_r]}{E[\overline{\boldsymbol{d}}_r]^2} \tag{5.7}$$

[1] Note that the assumption is normally not met due to the impact of scheduling policies and interdependencies between activities coming from the same project. For example, multiple activities from the same project may arrive after completion of their common predecessors.

$$= \frac{\frac{1}{a_r} \sum_{p \in \mathcal{P}} a_p \sum_{i \in \mathcal{V}_{pr}} \overline{d}_{ip}^2 - E[\overline{\boldsymbol{d}}_r]^2}{\left(\frac{1}{a_r} \sum_{p \in \mathcal{P}} a_p \sum_{i \in \mathcal{V}_{pr}} \overline{d}_{ip} \right)^2} \tag{5.8}$$

$$= a_r \frac{\sum_{p \in \mathcal{P}} a_p \sum_{i \in \mathcal{V}_{pr}} \overline{d}_{ip}^2}{\left(\sum_{p \in \mathcal{P}} a_p \sum_{i \in \mathcal{V}_{pr}} \overline{d}_{ip} \right)^2} - 1 \tag{5.9}$$

Finally, we get

$$CV_{\overline{\boldsymbol{d}}_r} = \sqrt{a_r \frac{\sum_{p \in \mathcal{P}} a_p \sum_{i \in \mathcal{V}_{pr}} \overline{d}_{ip}^2}{\left(\sum_{p \in \mathcal{P}} a_p \cdot \sum_{i \in \mathcal{V}_{pr}} \overline{d}_{ip} \right)^2} - 1} \tag{5.10}$$

Note that the reason for using $CV_{\overline{\boldsymbol{d}}_r}$ instead of using $Var[\overline{\boldsymbol{d}}_r]$ for controlling the variation related to the expected durations becomes obvious when we insert (5.4) in (5.6) and (5.10) such that we obtain

$$Var[\overline{\boldsymbol{d}}_r] = \left(\frac{c_r}{\lambda^{\max} \sum_{p \in \mathcal{P}} a_p \sum_{i \in \mathcal{V}_{pr}} \tilde{d}_{ip}} \right)^2 \left(\frac{1}{a_r} \sum_{p \in \mathcal{P}} \sum_{i \in \mathcal{V}_{pr}} \tilde{d}_{ip}^2 - \left(\frac{1}{a_r} \sum_{p \in \mathcal{P}} a_p \sum_{i \in \mathcal{V}_{pr}} \tilde{d}_{ip} \right)^2 \right) \tag{5.11}$$

$$CV_{\overline{\boldsymbol{d}}_r} = \sqrt{a_r \frac{\sum_{p \in \mathcal{P}} a_p \sum_{i \in \mathcal{V}_{pr}} \tilde{d}_{ip}^2}{\left(\sum_{p \in \mathcal{P}} a_p \cdot \sum_{i \in \mathcal{V}_{pr}} \tilde{d}_{ip} \right)^2} - 1} \tag{5.12}$$

We observe that $Var[\overline{\boldsymbol{d}}_r]$ depends on $\lambda^{\max}$ and c_r while $CV_{\overline{\boldsymbol{d}}_r}$ is independent of the two parameters. This can be explained by the fact that the two parameters do not affect the relative variation between the expected durations but only the absolute level of their mean and variation. Thus, we can measure and control the variation of the expected durations independently from the values of other parameters.

In order to make sure that the expected activity durations attain the desired level of variation for each resource type $r \in \mathcal{R}$ we check the following condition.

$$CV^{\overline{d},\min} \leq CV_{\overline{\boldsymbol{d}}_r} \leq CV^{\overline{d},\max} \quad \forall r \in \mathcal{R} \tag{5.13}$$

If (5.13) is not met the expected durations $\overline{d}_{ip}$ are determined again by repeating the steps starting from sampling the values for $\tilde{d}_{ip} \in [0; 1]$ in Step 2.

5.2.4 Step 4: Adjustments to Resource Type Specific Utilizations

In the fourth step, we take into account that the utilization per resource of type $r \in \mathcal{R}$ may be lower than u. Therefore, we use a *utilization multiplier* $\nu_r \in [0; 1]$ denoting the fraction of the actual utilization of the maximum utilization u such that $u_r = u \cdot \nu_r$. Now, we have two options for taking into account ν_r in the generation process.

1. Increasing c_r: With $\rho_r = u \cdot c_r$ we obtain

$$c_r(\nu_r) = \left\lceil \frac{\rho_r}{u \cdot \nu_r} \right\rceil = \left\lceil \frac{c_r}{\nu_r} \right\rceil \tag{5.14}$$

   Hence $u_r = \frac{u \cdot c_r}{c_r(\nu_r)} \leq u \cdot \nu_r$.
2. Scaling down the expected durations of all activity types (i, p) with $r_{ip} = r$ gives

$$\overline{d}_{ip}(\nu_r) = \overline{d}_{ip}\nu_r \forall\, p \in \mathcal{R}, i \in \mathcal{V}_{pr} \tag{5.15}$$

   Thus, we have

$$u_r = \frac{\nu_r \lambda \sum\limits_{p \in \mathcal{P}} a_p \sum\limits_{i \in \mathcal{V}_{pr}} \overline{d}_{ip}}{c_r} = \frac{\rho_r}{c_r}\nu_r = u \cdot \nu_r \tag{5.16}$$

Note that both options have no effect on $CV_{\overline{\boldsymbol{d}}_r}$.

5.2.5 Step 5: Check of Project Type Workloads

In the fifth step, we check whether the proportions of the expected work loads per project for the different project types correspond to the proportions of the workload indices wi_p. For a project type $p \in \mathcal{P}$ the expected workload per project is given by

$$\overline{d}_p = \sum_{i \in \mathcal{V}_p} \overline{d}_{ip} \tag{5.17}$$

Now, we check whether for any two project types $p_1, p_2 \in \mathcal{P}$ the following condition is met.

$$\frac{wi_{p_1}}{wi_{p_2}} \cdot (1 - \epsilon^{\text{wi}}) \leq \frac{\overline{d}_{p_1}}{\overline{d}_{p_2}} \leq \frac{wi_{p_1}}{wi_{p_2}} \cdot (1 + \epsilon^{\text{wi}}) \tag{5.18}$$

ϵ^{wi} is a tolerance parameter as it is difficult to meet the condition exactly due to sampled expected activity durations from a continuous uniformly distributed interval. If (5.18) is not met for some $p_1, p_2 \in \mathcal{P}$ the procedure is repeated starting with sampling of the values for $\tilde{d}_{ip} \in [0; 1]$ in Step 2.

5.2.6 Step 6: Storage of Additional Parameters

In the sixth step, additional parameters are simply stored with the other data of the problem instance. Firstly, we store for each project type $p \in \mathcal{P}$ the holding cost w_p per time unit. Secondly, we store α_p. α_p denotes the percentage of tardy projects of type p when fixed maximum flow time $\overline{D}_p^{\max}$ is set and the set of activities to be scheduled $\mathcal{B}_r^{\text{S}}(t)$ is determined randomly. Both are simply stored with the other data of the problem instance.

Further problem related parameters (e.g. for the probability distributions for activity durations and interarrival times) may be added. However, when we investigate mostly instances with exponentially distributed interarrival times and activity durations no further distribution parameters are needed.

The procedure is repeated until the specified number of sampled instances N^{PI} is generated. In order to obtain the same problem instances (for a given set of parameters), each time the procedure is run, we initialize the random number generator (RNG) using a given seed *sd*. As long as the same seed *sd* is used, sampled expected durations $\overline{d}_{ip}$ and resources types r_{ip} are independent from non-relevant parameters, such as OS_p or α_p. Thus, changing OS_p or α_p does not have an impact on the sampled values of $\overline{d}_{ip}$ and r_{ip}.

Chapter 6
Scheduling Using Priority Policies

In this chapter, we present a simulation study carried out to investigate the performance of priority policies in dynamic-stochastic environments.

As in most related studies in the project scheduling as well as the job shop literature preemptions are not allowed, we restrict our considerations to this case.

The study has three objectives. Firstly, to determine whether it is beneficial to use priority policies. Secondly, to show the impact of problem parameters on the performance of priority policies. Thirdly, to identify conditions, in terms of problem parameters, under which specific priority policies can be recommended.

In Sect. 6.1, we give a description of the priority policies considered in this study. They have been selected according to their performance for related problems, especially for those with a weighted tardiness objective, in the multi-project scheduling and job shop scheduling literature.

Section 6.2 presents in detail, the additional assumptions made for the simulation study, and the problem parameters considered.

In Sect. 6.3, we investigate the main effects of the problem parameters and in Sect. 6.4 we analyze in detail the conditions under which specific priority policies can be recommended.

6.1 Priority Policies

For stochastic project scheduling problems, various classes of policies have been proposed (cf. for example Möhring et al. [95] and Stork [122]). Our focus is on the class of *resource-based priority policies* (cf. Stork [122]) that are defined as follows.

Definition 6.1.1. At decision epoch t a **resource-based priority policy** (RBP) π^{prio} orders the activities in $\mathcal{W}(r,t)$ based on priorities obtained by a **priority rule** and selects activities according to this order.

P. Melchiors, *Dynamic and Stochastic Multi-Project Planning*, Lecture Notes in Economics and Mathematical Systems 673, DOI 10.1007/978-3-319-04540-5_6,

Resource based priority policies combine priority rules with the *parallel schedule generation scheme* (cf. Kolisch [74]), which is compatible with the *nonanticipativity constraint* (cf. Ballestin [10]).

An alternative to RBPs are *activity based priority policies* (ABPs) (cf. Stork [122]) where the additional constraint exists that an activity with a lower priority (w.l.o.g. we prefer activities with higher priorities) may not be scheduled before an activity with a higher priority. From a theoretical point of view ABPs have the advantage that *Graham anomalies* (cf. Graham [53]) are avoided. Graham anomalies refer to the fact that reducing the duration of an activity may lead to an increase of the project completion time which is a consequence of the non-idling property of RBSs. By the additional constraint ABPs allow idleness and thus avoid Graham anomalies. Activities with lower priorities are ignored, although they could be scheduled, as soon as an activity could not be scheduled. However, some preliminary experiments have shown that ABPs lead in many cases to a bad performance or even instable behavior with in an unbounded long term average number of projects in the system. This is due to idleness of resources that occurs quite frequently such that the busy periods of the resources are not sufficient to cope with the workload associated with the projects arriving at the system.

After a short discussion of how parameters of priority rules are computed, we state the priority rules to be investigated in formal terms. The selection of the rules has been based on their performance for similar problems especially with weighted tardiness or a weighted flow time objectives (cf. Lawrence and Morton [85], Kurtulus and Narula [82] and Kutanoglu and Sabuncuoglu [83]). Furthermore, rules have been selected where information can be easily obtained for example weights such that their implementation is less costly. For the rest of the chapter, we refer to a *priority policy* or only policy when we refer to a RBP that is based on a given priority rule such as FCFS.

6.1.1 Computation of Rule Parameters

In this section, we discuss two parameters used by several priority rules.

The first parameter is the expected length $\overline{D}_p^{\mathrm{CP}}$ of the critical path, being the longest path in the project network when limited resources are neglected, of project type p. The second parameter is an estimate for the expected latest start time $\overline{l}_{ij}$ for activities (i, j) in order to meet the due date of project j. One major difficulty in determining both parameters is their dependence on the current system state (in terms of sets of waiting activities and activities in process) and future scheduling decisions. As those dependencies partially result from scarce resources we use simple estimates for $\overline{D}_p^{\mathrm{CP}}$ and $\overline{l}_{ij}$ instead that are computed without consideration of limited resources and only using expected durations as follows. At first, we estimate the expected length of the critical path $\overline{D}_p^{\mathrm{CP}}$ by computing the longest path in the network if the expected durations of the activity types are used as if they were

deterministic durations. Then, for each project j we obtain the estimates for $\overline{\boldsymbol{l}}_{ij}$ by the backward recursion proposed by Lawrence and Morton [85] as given by

$$\overline{\boldsymbol{l}}_{ij} = \begin{cases} \boldsymbol{t}_j^{\mathrm{a}} + \max\{\boldsymbol{D}_j^{\max}, D_{\boldsymbol{p}_j}^{\mathrm{CP}}\} - \overline{d}_{i\boldsymbol{p}_j} & \mathcal{V}_{\boldsymbol{p}_j}^{\mathrm{Succ}}(i) = \emptyset \\ \min\limits_{i'}\{(\overline{\boldsymbol{l}}_{i'j} - \overline{d}_{i\boldsymbol{p}_j}) : i' \in \mathcal{V}_{\boldsymbol{p}_j}^{\mathrm{Succ}}(i)\} & \text{otherwise} \end{cases} \tag{6.1}$$

The computation of latest start times based on deterministic durations is expected to be typical procedure of practitioners although durations may be stochastic since scheduling is mostly based on deterministic models. Thus, we consider priority policies based on information that is typically used.

6.1.2 Priority Rules

6.1.2.1 Bottleneck Dynamics Rules

A family of rules summarized under the *bottleneck dynamics* approach has been proposed by Lawrence and Morton [85] and Morton and Pentico [96]. In particular, these rules have been designed for the weighted tardiness objective. The generic form of the rules is

$$\max_{(i,j)\in\mathcal{W}(r,t)} \frac{w_{\boldsymbol{p}_j}}{\boldsymbol{\pi}_{ij}(t)} \boldsymbol{U}_{ij}(t) \tag{6.2}$$

The basic idea of the approach is to balance the cost $w_{\boldsymbol{p}_j}\boldsymbol{U}_{ij}(t)$ of delaying an activity for one time unit against the marginal opportunity cost $\boldsymbol{\pi}_{ij}(t)$ associated with starting its execution one time unit earlier. The term $w_{\boldsymbol{p}_j}\boldsymbol{U}_{ij}(t)$ for the delay cost has two components: The weight (holding costs per time unit) $w_{\boldsymbol{p}_j}$ reflects the maximum possible delay costs incurred when the project tardiness increases by one time unit. As activities may have positive total slack $\overline{\boldsymbol{l}}_{ij} - t$, the urgency factor $\boldsymbol{U}_{ij}$, $0 \leq \boldsymbol{U}_{ij}(t) \leq 1$, is used for discounting $w_{\boldsymbol{p}_j}$. It is given by

$$\boldsymbol{U}_{ij}(t) = \exp\left(-\frac{(\overline{\boldsymbol{l}}_{ij} - t)^+}{\kappa \overline{\overline{d}}_{r_{i\boldsymbol{p}_j}}(t)}\right) \tag{6.3}$$

where $\overline{\overline{d}}_{r_{i\boldsymbol{p}_j}}(t)$ is the average expected duration of activities waiting in front of resource type r at time t and κ is a scaling factor denoted as *lookahead parameter*. By increasing κ, the fact that waiting times consume slack and thus increase the urgency is taken into account (cf. Lawrence and Morton [85]). Figure 6.1 shows the shape of the urgency factor for different values for κ and $\overline{\overline{d}}_{r_{i\boldsymbol{p}_j}}(t) = 1$. For negative slack, the urgency factor is 1 and it decreases exponentially for positive slack. The higher the look ahead parameter κ, the less rapid the decrease of the urgency

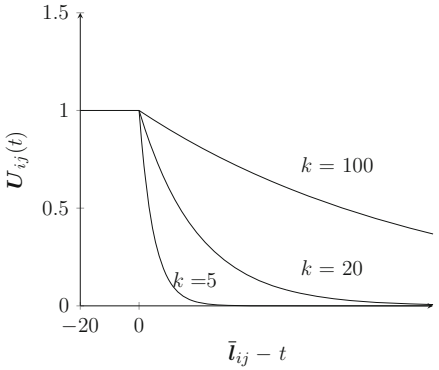

Fig. 6.1 Function of the urgency factor $\boldsymbol{U}_{ij}(t)$

factor. In order to estimate opportunity cost $\boldsymbol{\pi}_{ij}(t)$ Lawrence and Morton [85] propose two methods, namely *myopic activity costing* (MC) and *global activity costing* (GC). Myopic activity costing considers only activity (i, j) to be scheduled on resource type $r_{i\boldsymbol{p}_j}$ which gives opportunity cost

$$\boldsymbol{\pi}_{ij}(t) = \frac{\boldsymbol{\pi}_{r_{i\boldsymbol{p}_j}}(t)}{c_{r_{i\boldsymbol{p}_j}}} \overline{d}_{i\boldsymbol{p}_j} \tag{6.4}$$

The rationale is that scheduling activity (i, j) seizes fraction $\frac{1}{c_{r_{i\boldsymbol{p}_j}}}$ of the resources of type $r_{i\boldsymbol{p}_j}$ for $\overline{d}_{i\boldsymbol{p}_j}$ time units. Each time unit is worth $\boldsymbol{\pi}_{r_{i\boldsymbol{p}_j}}(t)$, being the estimated opportunity cost for postponing the activities waiting in front of resource type $r_{i\boldsymbol{p}_j}$ by one time unit. As we have $c_{r_{i\boldsymbol{p}_j}} = c_r$ and $\boldsymbol{\pi}_{r_{i\boldsymbol{p}_j}}(t) = \boldsymbol{\pi}_r(t)$ for all activities $(i, j) \in \mathcal{W}_r(t)$ waiting in front of resource type r, (6.4) reduces to $\boldsymbol{\pi}_{ij}(t) = \overline{d}_{i\boldsymbol{p}_j}$.

For global activity costing (GC), the opportunity costs of all unscheduled activities of project j from the set $\mathcal{U}_j^{\mathrm{S}}(t)$, including activity (i, j), are considered. Morton and Pentico [96] make the assumption that to expedite project j all unscheduled activities (i', j) with $i' \in \mathcal{U}_j^{\mathrm{S}}(t)$ of project j have to be started immediately as soon as they become ready. Thus, we obtain the opportunity cost by

$$\boldsymbol{\pi}_{ij}(t) = \sum_{m \in \mathcal{U}_j^{\mathrm{S}}(t)} \left(\frac{\boldsymbol{\pi}_{r_{m\boldsymbol{p}_j}}(t)}{c_{r_{m\boldsymbol{p}_j}}} \overline{d}_{m\boldsymbol{p}_j} \right) \tag{6.5}$$

where resource prices $\boldsymbol{\pi}_r(t)$ are set either by *uniform resource pricing* in Eq. (6.6) (cf. Lawrence and Morton [85]) or by *dynamic resource pricing* in Eq. (6.7) (cf. Morton and Pentico [96]).

$$\boldsymbol{\pi}_r^{\mathrm{U}}(t) = 1 \quad \forall r \in \mathcal{R} \tag{6.6}$$

$$\boldsymbol{\pi}_r^{\mathrm{D}}(t) = \sum_{(i,j) \in \mathcal{W}_r(t)} w_{\boldsymbol{p}_j} \cdot \boldsymbol{U}_{ij}(t) \tag{6.7}$$

Combining the two activity costing methods, myopic and global, with the resource pricing schemes, uniform and dynamic, we obtain the following three bottleneck dynamic (BD) priority rules.

BD-MC: BD with Myopic Activity Costing (MC)

$$\max_{(i,j)\in\mathcal{W}_r(t)} \frac{w_{\boldsymbol{p}_j}}{\overline{d}_{i\,\boldsymbol{p}_j}} \cdot \boldsymbol{U}_{ij}(t) \tag{6.8}$$

BD-GC-U: BD with Global Activity Costing (GC) and Uniform Resource Pricing (U)

$$\max_{(i,j)\in\mathcal{W}_r(t)} \frac{w_{\boldsymbol{p}_j}}{\sum\limits_{m\in\mathcal{U}_j^{\mathrm{S}}(t)} \overline{d}_{m\boldsymbol{p}_j} \frac{\boldsymbol{\pi}_r^{\mathrm{U}}(t)}{c_{r_{m\boldsymbol{p}_j}}}} \cdot \boldsymbol{U}_{ij}(t) \tag{6.9}$$

BD-GC-D: BD with Global Activity Costing (GC) and Dynamic Resource Pricing (D)

$$\max_{(i,j)\in\mathcal{W}_r(t)} \frac{w_{\boldsymbol{p}_j}}{\sum\limits_{m\in\mathcal{U}_j^{\mathrm{S}}(t)} \overline{d}_{m\boldsymbol{p}_j} \cdot \frac{\boldsymbol{\pi}_r^{\mathrm{D}}(t)}{c_{r_{m\boldsymbol{p}_j}}}} \cdot \boldsymbol{U}_{ij}(t) \tag{6.10}$$

Note that the priority rules given by (6.8)–(6.10) are similar to the well known WSPT-rule. The BD-MC-rule multiplies, for each waiting activity, the priority from WSPT with urgency factor $\boldsymbol{U}_{ij}(t)$ whereby due dates are taken into account. Accordingly, the two rules based on global activity costing, BD-GC-U and BD-GC-D, are not only considering the duration of the activity under consideration but of all unscheduled activities of the project the activity belongs to.

6.1.2.2 Further Rules

Next to the three bottleneck dynamic rules, we investigate the following rules (for the references we refer the reader to end of this section).

First-Come, First-Served (FCFS)

$$\min_{(i,j)\in\mathcal{W}_r(t)} \boldsymbol{t}_{ij}^{\mathrm{a}} \tag{6.11}$$

where $\boldsymbol{t}_{ij}^{\mathrm{a}}$ refers to the arrival time of activity (i,j) at resource type $r_{i\,\boldsymbol{p}_j}$.

Maximum Penalty (MAXPEN)

$$\max_{(i,j)\in\mathcal{W}_r(t)} w_{\boldsymbol{p}_j} \tag{6.12}$$

Due Date Modified Shortest Activity from Shortest Project (SASP-DD)

$$\min_{(i,j)\in\mathcal{W}_r(t)} \begin{cases} \overline{\boldsymbol{l}}_{ij} - t & \overline{\boldsymbol{l}}_{ij} - t < 0 \\ \overline{D}^{\mathrm{CP}}_{\boldsymbol{p}_j} + \overline{d}_{i\boldsymbol{p}_j} & \text{otherwise} \end{cases} \tag{6.13}$$

Weighted Earliest Due Date (WEDD)

$$\min_{(i,j)\in\mathcal{W}_r(t)} \begin{cases} w^{-1}_{\boldsymbol{p}_j} \cdot (\boldsymbol{t}^a_j + \boldsymbol{D}^{\max}_j - t) & \boldsymbol{t}^a_j + \boldsymbol{D}^{\max}_j - t \geq 0 \\ w_{\boldsymbol{p}_j} \cdot (\boldsymbol{t}^a_j + \boldsymbol{D}^{\max}_j - t) & \boldsymbol{t}^a_j + \boldsymbol{D}^{\max}_j - t < 0 \end{cases} \tag{6.14}$$

Weighted Minimum Slack (WMINSLK)

$$\min_{(i,j)\in\mathcal{W}_r(t)} \begin{cases} w^{-1}_{\boldsymbol{p}_j} \cdot (\overline{\boldsymbol{l}}_{ij} - t) & \overline{\boldsymbol{l}}_{ij} - t \geq 0 \\ w_{\boldsymbol{p}_j} \cdot (\overline{\boldsymbol{l}}_{ij} - t) & \overline{\boldsymbol{l}}_{ij} - t < 0 \end{cases} \tag{6.15}$$

Note that we modified WEDD and WMINSLK such that in case of negative remaining time until the due date ($\boldsymbol{t}^a_j + \boldsymbol{D}^{\max}_j - t > 0$) or negative total slack ($\overline{\boldsymbol{l}}_{ij} - t$), the priority value is still positive.

Weighted Shortest Processing Time (WSPT)

$$\max_{(i,j)\in\mathcal{W}_r(t)} \frac{w_{\boldsymbol{p}_j}}{\overline{d}_{i\boldsymbol{p}_j}} \tag{6.16}$$

Weighted Critical Ratio and Shortest Processing Time (W(CR+SPT))

$$\max_{(i,j)\in\mathcal{W}_r(t)} \frac{w_{\boldsymbol{p}_j}}{\overline{d}_{i\boldsymbol{p}_j} \cdot \max\left\{1, \frac{\boldsymbol{t}^a_j + \boldsymbol{D}^{\max}_j - t}{D^{\mathrm{CP}}_{\boldsymbol{p}_j} - \overline{l}^{\mathrm{CP}}_{i\boldsymbol{p}_j}}\right\}} \tag{6.17}$$

The W(CR+SPT)-rule has originally been designed for minimizing a weighted tardiness objective in job shop scheduling. We have adapted it according to the idea of Tsai and Chiu [128] to the multi-project scheduling problem. Instead of the expected remaining workload of a job the expected length of longest path

between an activity type (i, p) and the end of the project of type p is used. For the computation of the expected latest start times $\overline{\boldsymbol{l}}_{ij}$, we do not take into account waiting times such that $E\left[D_p^{\mathrm{CP}} - l_{ip}^{\mathrm{CP}}\right]$ is a constant for all projects of type p.

Random (RAN)

This policy randomly selects activities for scheduling.

The formulas for FCFS, WEDD, WMINSLK and WSPT have been taken from Lawrence and Morton [85]. W(CR+SPT) has been proposed for dynamic job shop scheduling problems by Kutanoglu and Sabuncuoglu [83] based on the CR+SPT-rule by Andersson and Nyirenda [4]. SASP-DD has been taken from Dumond and Mabert [43]. Finally we note that ties are broken randomly.

6.2 Experimental Design

Firstly, we discuss more specific assumptions for the simulation study. Secondly, we outline the relevant parameters for the generation of problem instances and simulation. For details of the generation procedure, we refer to Chap. 5.

6.2.1 Preliminaries

At first we assume that preemptions are not allowed such that we consider non-preemptive RBPs.

In addition, the following specific assumptions are made. For each project type $p \in \mathcal{P}$, we assume a Poisson arrival process with rate λ_p. Activity durations are assumed to be exponentially distributed with mean $\overline{d}_{ip}$ for activity type (i, p). Maximum flow times $\boldsymbol{D}_j^{\max}$ are drawn from an uniform distributed interval $\left[(1-\beta)\overline{D}_{\boldsymbol{p}_j}^{\max}; (1+\beta)\overline{D}_{\boldsymbol{p}_j}^{\max}\right]$. The number of projects in the system is restricted to $K^{\max}$ because, as addressed in Sect. 2.1, scheduling policies (especially if they are not optimal) may lead to instable system behavior where not steady state is reached (cf. Kumar and Seidman [80] or Meyn [93]) such that simulation run times may become very large. $K^{\max}$ and $\overline{D}_p^{\max} \forall\, p \in \mathcal{P}$ are determined by two simulation runs where activities are scheduled using the RAN-policy. In the first run, we have not restricted the number of projects $|\mathcal{J}(t)|$ at any time t. Then, we have set $K^{\max}$ to the number of projects which is not exceeded 99 % of the time. As a consequence, the system is nearly an open system without limitation on the number of projects. In the second run, taking into account $K^{\max}$, as determined above, we have determined $\overline{D}_p^{\max}$ such that the share of completed projects from all accepted projects of type

$p \in \mathcal{P}$ with a flow time $\boldsymbol{F}_j \leq \overline{D}_p^{\max}$ amounts to $1 - \alpha_p$. Parameter α_p serves to control due date tightness and corresponds to the share of tardy projects of type p when using the RAN-policy. Employing α_p, we address the problem that the difficulty to meet due dates depends on multiple factors, amongst others the utilization of resources (cf. Ramasesh [109]). As α_p is a statistical measure, defined on realizations of project flow times, it allows to control the tightness independently from problem parameters. Furthermore, as RAN does not use any information, due date tightness is controlled independently from the information used by specific scheduling policies.

Finally, we discuss a normalization of the results. Instead of using the objective values $\boldsymbol{Z}(\pi)$, obtained when using policy π for a problem instance, we use normalized results given by

$$\boldsymbol{Z}^n(\pi) = \frac{\boldsymbol{Z}(\pi)}{\boldsymbol{Z}(\text{RAN})} \tag{6.18}$$

where $\boldsymbol{Z}^n(\pi)$ is the average weighted tardiness obtained for a policy π relative to the average weighted tardiness obtained using RAN. The normalization has the advantage that, when averaging over the results from a set of problem instances, the values lie in a similar range such that they have a similar impact on the mean.

6.2.2 Generation of Problem Instances

The generation of problem instances is controlled by two set of parameters, system parameters and project parameters.

6.2.2.1 System Parameters

The *number of resource types* $|\mathcal{R}|$ is set to 1, 3, 5 ,10 ,15 and 20 with $c_r = 1$ $\forall r \in \mathcal{R}$.

As for $c_r = 1 \quad \forall r \in \mathcal{R}$ each resource type corresponds to a resource, we use for the rest of the chapter, the terms resource types and resources interchangeably.

The *utilization* u_r of resource $r \in \mathcal{R}$ is assumed to be equal for all resources $r \in \mathcal{R}$ such that $u_r = u \; \forall r \in \mathcal{R}$. To obtain a given level of u expected durations of the activity types are set such that for $\lambda = \lambda^{\max} = 0.1333$ the utilization amounts to 100 %. Then, we attain a given level of u by setting the total arrival rate to $\lambda = \lambda^{\max} \cdot u$ where u is considered at the levels of 70 and 90 %. For simplicity, the project type related arrival rates λ_p have equal shares of λ such that $a_p = \frac{1}{|\mathcal{P}|}$.

The variation of the expected durations of the activities processed on a resource (type) r is controlled by their coefficient of variation $CV_{\overline{\boldsymbol{d}}_r}$ where $\overline{\boldsymbol{d}}_r$ is a random

Table 6.1 Values for the system parameters

Parameter	Value
$\lvert\mathcal{R}\rvert$	1, 3, 5, 10, 15, 20
u	0.7, 0.9
$\left[CV^{\overline{d},\min}; CV^{\overline{d},\max}\right]$	[0; 0.4], [0.8; 1,000]

variable for the expected duration of any activity to be processed on resource r. For details, we refer to Chap. 5. For our experiment, we consider two intervals $\left[CV^{\overline{d},\min}; CV^{\overline{d},\max}\right]$ from which $CV_{\overline{d}_r}$ may come from. The first interval $[0; 0.4]$ refers to small variations of the expected durations and second interval $[0.8; 1.0]$ to large variations of the expected durations. Table 6.1 summarizes the system parameters and the levels at which they have set within the experimental design.

6.2.2.2 Project Type Parameters

We investigate problem instances with two project types ($\mathcal{P} = \{1, 2\}$) having weights $w_1 = 1$ and $w_2 = 2$. As we intend to obtain general insights into the effects of problem parameters we consider similar project types in the sense that they have the same values for their parameters except the holding cost w_p. We consider two levels of due date tightness $\alpha = 20, 80\,\%$ with $\alpha_p = \alpha \;\forall p \in \mathcal{P}$.

The parameter β for the variation of the maximum flow time $\boldsymbol{D}_j^{\max}$ is set to $50\,\%$.

For controlling the networks of the project types, we have set the following parameters.

The number of activity types for each project type $p \in \mathcal{P}$ has been set to $|\mathcal{V}_p| = 20 \;\forall p \in \mathcal{P}$.

The network structure of each project type is controlled using the order strength (OS). Recall that OS is a $[0, 1]$-normalized summary measure for the number of all precedence relations, including the transitive ones. A value of 0 indicates a purely parallel network where there are no precedence relations between the activities. A value of 1 indicates a serial network where activities have to be executed in a strict sequence. A formal definition of OS is given by Schwindt [116]. We have set OS_p to 0, 0.2 , 0.4, 0.6, 0.8 and 1.0 for both project types $p \in \mathcal{P}$. For each level of OS_p, four sample networks are generated such that we can form two pairs of different networks.

Finally, the workload indices wi_p are set to one for all $p \in \mathcal{P}$ with a tolerance parameter $\epsilon^{\text{wi}} = 0.15$ in order to have project types with an approximately equal expected workload $\overline{d}_p = \sum_{i \in \mathcal{V}_p} \overline{d}_{ip}$. The requirement that project types must have an approximately equal workload is made in order to avoid hidden effects of the workloads on the performance of priority policies. Hidden effects may result from the fact that some policies (e.g. BD-GC-U) consider the total expected workload from unscheduled activities of the projects such that large differences between the

Table 6.2 $|\mathcal{V}_{pr}|$ for all $p \in \mathcal{P}$ and $r \in \mathcal{R}$

$\lvert\mathcal{R}\rvert$	$\lvert\mathcal{V}_{pr}\rvert$
1	20
3	7 7 6
5	4 4 4 4 4
10	2 2 2 2 2 2 2 2 2 2
15	2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
20	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Table 6.3 Values of the project type parameters

Parameter	Value
α (%)	80, 20
β (%)	50
$(\lvert\mathcal{V}_1\rvert, \lvert\mathcal{V}_2\rvert)$	(20, 20)
$(\overline{w}_1, \overline{w}_2)$	(1, 2)
$(\mathrm{wi}_1, \mathrm{wi}_2)$	(1, 1)
ϵ^{wi}	0.15
OS	0, 0.2, 0.4, 0.6, 0.8, 1.0

expected workloads of the project types may lead to a better performance of such policies.

For each combination of parameters and pair of networks $N^{\mathrm{PI}} = 2$ instances are generated as outlined in Chap. 5. The number of activity types $|\mathcal{V}_{pr}|$ to be processed by each resource $r \in \mathcal{R}$ are specified in Table 6.2. In our design each resource should process nearly the same number of activity types. Thus, for fixed $|\mathcal{V}_p| = 20 \; \forall \, p \in \mathcal{P}$ the values for $|\mathcal{V}_{pr}|$ depend on $|\mathcal{R}|$.

Table 6.3 summarizes the parameters related to project types. The total number of problem instances results from the product over the number of levels (given in parentheses) w.r.t. $|\mathcal{R}|$ (6), u (2), $CV_{\overline{\boldsymbol{d}}_r}$ (2), α (2), OS_p (6), the number of generated pairs of networks for each level of OS_p (2) and the number of the instances N^{PI} (2) for a given combination of parameters such that we obtain 1,152 instances. For each problem instance we simulate each priority policy such that we have 12,672 combinations of priority policies and problem instances.

6.2.3 Simulation Set Up

For the simulation of the problem instances, we have implemented a discrete event simulation tool using JAVA. When running the simulation, we employed a warm up period of 10,000 projects to reach a steady state. The length of the warm up period has been determined using Welch's procedure (cf. Law [84]) for problem instances with $|\mathcal{R}| = 20$ and $u = 0.9$ where a high number of projects is expected to be in the system. By this, we have obtained a longer warm up period than necessary for most of our instances, ensuring that we are on the safe side. The length of the observation period has been set to 20,000 projects. As variance reduction technique,

we have employed *common random numbers* (see Law [84]) such that, for each problem instance, all priority policies are applied on the same sample of projects characterized by realizations for type, interarrival times and activity durations.

For each combination of problem instance and priority policy, we have run 10 replications such that we could verify that for most ínstances and policies the resulting paired t-confidence intervals (cf. Law [84]) do not contain 0 at a confidence level of 95 %. Furthermore, the width of the confidence interval is at most 10 % of Z_π^n.

For the BD-policies the lookahead parameter κ is set to 1 as for this value good results have been obtained in preliminary experiments.

6.3 Main Effects of Problem Parameters

Let us first discuss the main effects of the problem parameters α, $|\mathcal{R}|$, OS_p, $CV_{\overline{d}}$ and u on the performance of the priority policies. In order to obtain the main effect of a problem parameter, we average over sets of problem instances where a parameter has a given value. Then, the differences between the means obtained for different values of the problem parameter are computed which indicate an effect (cf. Law [84]). We have verified that, despite interactions between parameters, the effects can also be observed for subsets of the problem instances. In the following $\overline{Z}^n(\pi)$ denotes the mean of the $Z^n(\pi)$ values for a set of problem instances with given value for a problem parameter.

6.3.1 Due Date Tightness

Figure 6.2 shows the impact of due date tightness α controlled via the percentage of tardy projects. For low due date tightness ($\alpha = 0.2$), policies employing due date information such as slack or critical ratio exhibit a superior performance. For high due date tightness ($\alpha = 0.8$), the performance of policies using due date information is still better, but to a much smaller extend. This can be explained by the fact that, for $\alpha = 0.2$, many activities have ample expected total slack $\overline{l}_{ij}(t) - t$. In this case, it is sufficient to identify the activities with small total slack and schedule them first. Activities with enough total slack need no further consideration. However, for $\alpha = 0.8$ the total slack of many activities is small and hence is not as good for discriminating activities anymore. Other information such as weights, expected activity durations or resource prices becomes more important.

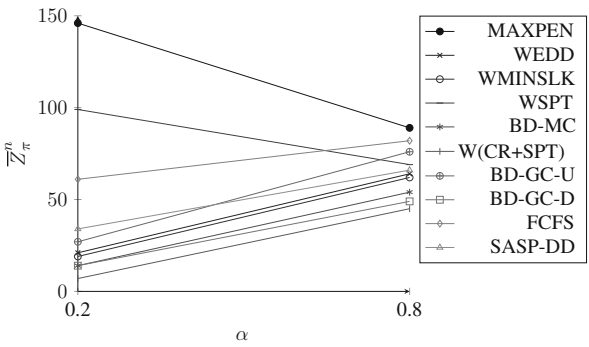

Fig. 6.2 Impact of the tightness factor

6.3.2 Number of Resources

Figure 6.3 shows the performance for different numbers of resources $|\mathcal{R}|$ at the two levels of due date tightness ($\alpha = 0.2$ and $\alpha = 0.8$). For a clear representation, we separate the priority policies according to the type of performance function into two figures.

For both levels of tightness, we observe that for increasing $|\mathcal{R}|$, the performance of the priority policies depicted on the left, BD-GC-D, BD-GC-U, MAXPEN, WEDD, WMINSLK and SASP-DD, deteriorates. Contrarily, the performance of some policies depicted on the right, namely WSPT and W(CR+SPT), improves slightly. The performance of the remaining policies on the right is varying and partially depends on the level of due date tightness while the performance of FCFS is constant for high due date tightness and slightly improving for low due date tightness. BD-MC, independently of due date tightness, exhibits an improved performance until $|\mathcal{R}| = 5$ resources and deteriorates afterwards.

Altogether, we observe that for low due date tightness, policies using due date information give a better performance which is almost identical. This can be explained using the argument from Sect. 6.3.1. Finally, we note that W(CR+SPT) exhibits the best performance for higher $|\mathcal{R}|$.

Next, we analyze in more depth the reason for the good performance of BD-GC-U, BD-GC-D, WEDD WMINSLK and SASP for small $|\mathcal{R}|$. As a starting point, we consider the case where $|\mathcal{R}| = 1$ with unbounded $K^{\max}$ and generally distributed activity durations. If we set $\alpha = 100\,\%$ the objective function turns into the minimization of the average weighted flow time. For this specific case, the optimal scheduling policy is a *priority index policy* (cf. Nino-Mora [100]), as given by

Theorem 6.3.1. *The optimal scheduling policy w.r.t. expected weighted flow time is a priority index policy where an activity is selected according to*

$$\max_{(i,j)\in\mathcal{W}(r,t)} \frac{w_{p_j}}{\sum\limits_{m\in\mathcal{U}_j(t)} \overline{d}_{mp_j}} \tag{6.19}$$

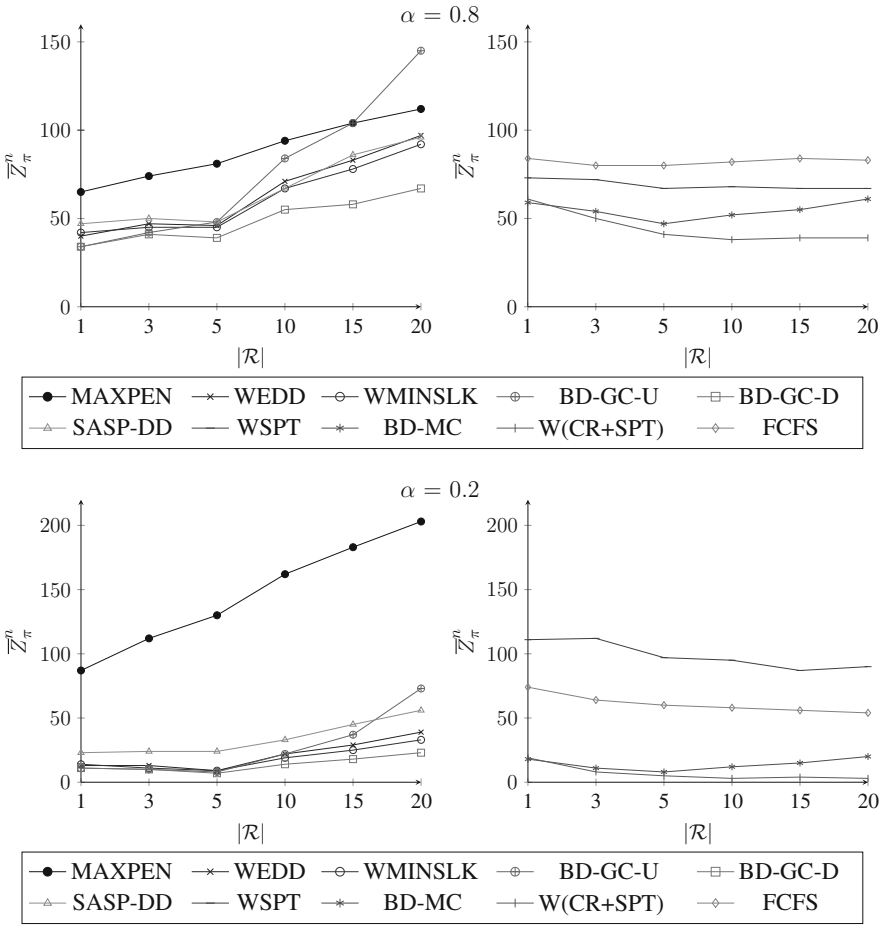

Fig. 6.3 Impact of $|\mathcal{R}|$

As a tie breaker among the activities of a project and between the projects $j \in \mathcal{J}(t)$ in case of equal priority any policy can be used.

Recall that $\mathcal{U}_j(t)$ denotes the set of unfinished activities which is equivalent to $\mathcal{U}^S_j(t)$ at decision times since no activity is in process. The proof of Theorem 6.3.1 can be found in Sect. 7.2 and is based on Klimov's [72] result for optimally scheduling multiple job classes waiting in front of a queueing system with a single resource. We observe that (6.19) is equivalent to BD-GC-U given by (6.9) with an urgency factor $\boldsymbol{U}_{ij}(t) = 1$. Furthermore, we observe that similarly to BD-GC-U the optimal policy is a generalization of WSPT where a project is considered as one job. As the completion of an activity leads to a reduction of the expected remaining workload $\sum_{m \in \mathcal{U}_j(t)} \overline{d}_{mp(j)}$ the priority of a project increases. Hence, the activities of

a project are processed as long as no project with a higher priority arrives. For this case, the project in process is preempted on completion of the activity in process. This behavior reflects the basic approximation assumption (cf. Sect. 6.1.2) of the BD-approach, which states that in order to expedite a project (reduce its tardiness) all its unscheduled activities (including the activity currently considered) must be expedited. Resource prices are not needed as all activity types are processed by the same resource.

This suggests that, for small $|\mathcal{R}|$ where each resource type processes many activity types, the performance of BD-GC-U and the BD-GC-D is less sensitive to resource prices. For larger $|\mathcal{R}|$, the number of activity types processed per resource type decreases such that more resource types are involved for processing a project type. Hence, the benefit of scheduling an activity is affected by system variables such as sets of waiting activities and activities in process at other resource types. This explains the deteriorating performance of the two policies as well as the observation that BD-GC-D with dynamic resource price estimates performs better than BD-GC-U.

For explaining the observation that SASP-DD, WEDD and WMINSLK exhibit a similar performance as BD-GC-U and BD-GC-D, we note that due dates are positively correlated with project arrival times such that projects having earlier due dates tend to have arrived earlier. Thus, projects with earlier due dates tend to have more completed and scheduled activities. As SASP-DD, WMINSLK and WEDD tend to prefer activities from projects with earlier due dates they implicitly prefer projects having less expected workload from unscheduled activities, similarly to BD-GC-D and the BD-GC-U.

Finally, we explain the performance of WSPT, W(CR+SPT) and BD-MC. Note that the three policies are based on the same principle and consider each activity as an isolated job such that all other unfinished activities of a project are ignored. Thus, the policies are expected to perform well if the activities belong to separate projects and the successors of the waiting activities do not interfere each other by demanding the same resources. Obviously, both conditions are more likely to be met for large $|\mathcal{R}|$. The deteriorating performance of BD-MC for large $|\mathcal{R}|$ may be explained by two issues related to $\boldsymbol{U}_{ij}(t)$. Firstly, the lookahead parameter κ of the BD-MC-policy is not always optimally set. Secondly, the critical ratio employed by W(CR+SPT) may be a more appropriate urgency measure than $\boldsymbol{U}_{ij}(t)$.

6.3.3 Order Strength

Figure 6.4 shows the results for different values of order strength (*OS*). Again, we present the results for $\alpha = 0.8$ and $\alpha = 0.2$ separately. For high due date tightness, we observe that, for all policies except WSPT and W(CR+SPT), the performance deteriorates at higher values of *OS*. By contrast, the performance of WSPT slightly improves when increasing *OS* while for low due date tightness the performance of WSPT deteriorates. The performance of W(CR+SPT) is nearly

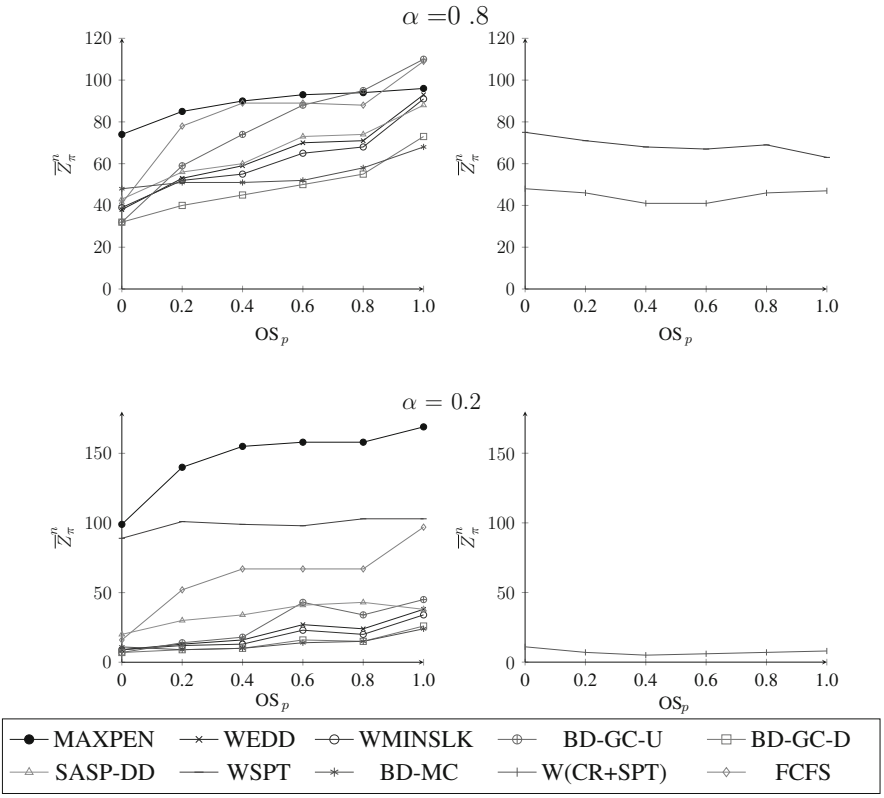

Fig. 6.4 Impact of OS_p on policy performance

stable for both levels of due date tightness. Finally, we observe that FCFS shows a good performance for $OS = 0$.

We first explain the performance of BG-GC-D and BD-GC-U. At low levels of *OS*, there are only few precedence relations such that the activities of a project that are processed by each resource can be considered as nearly independent sub-projects. On the level of such sub-projects, we can argue using Theorem 7.2.1 that activities from sub-projects of a resource having less expected remaining workload and high delay cost should be preferred. At the same time, at the level of entire projects, expediting a subproject is only beneficial if all parallel sub-projects are expedited as well. As all sub-projects must be finished in order to finish the entire project the advantage of expediting a subproject would be reduced by parallel sub-projects not expedited. We observe that both aspects of good scheduling decisions are implicitly taken into account by the consideration of the expected workload from unscheduled activities of a project (being equal to sum of the expected remaining workload over all sub-projects).

The similar performance of SASP-DD, WMINSLK and WEDD can be explained again by the positive correlation between due dates and expected workload from unscheduled activities as discussed in Sect. 6.3.2.

To explain the performance of WSPT for low due date tightness, we note that the latter does not take into account due date information and hence has an inferior performance. However, the effect of less advantageous scheduling decisions is weaker for small values of OS. Activities are less critical in the sense that a delay is less likely to increase project tardiness as more parallel activities are likely to be completed afterwards.

The performance of FCFS for $OS_p = 0$ can be explained by the fact that the arrival times of activities $\boldsymbol{t}_{ij}^{\mathrm{a}}$ belonging to the same project are identical to the project arrival times such that $\boldsymbol{t}_{ij}^{\mathrm{a}} = \boldsymbol{t}_{j}^{\mathrm{a}}$. Thus, activities from projects having arrived earlier are preferred which results in a similar behavior as policies based on due date information such as SASP-DD or WMINSLK.

6.3.4 Variation of Expected Activity Durations

Figure 6.5 shows the results for the two ranges of $CV_{\overline{d}_r}$. The results are shown separately as the performance of the priority policies follows the same pattern for the two levels of due date tightness. We observe that the policies based on the WSPT-principle, that are namely BD-MC and W(CR+SPT) perform better for a higher $CV_{\overline{d}_r}$ level.

This can intuitively be explained by the fact that minimizing the average weighted flow time of the activities at each resource becomes more effective when the variation between the expected durations of the activities is larger. Note that a WSPT-policy (or $c\mu$-policy) is well known to minimize the average weighted flow time of projects consisting of a single activity at a single resource (cf. Pinedo [103]) for static as well as dynamic problems. Furthermore, short average weighted flow times of activities at the resources tend to have a positive effect on the average weighted tardiness on the level of projects.

6.3.5 Utilization per Resource

Figure 6.6 shows the effect of the *utilization per resource* u. Obviously, the effect is small. This contrasts the findings in the literature where increasing utilization leads to a higher effectiveness of policies aimed at flow time related objectives such as WSPT (cf. Kutanoglu and Sabuncuoglu [83] or Vepsalainen and Morton [132]). An explanation is delivered by the fact that the tightness is controlled by the statistical measure *percentage of tardy projects* α. Hence, the effect of utilization is implicitly taken into account when setting the mean maximum durations $\overline{D}_p^{\max} \forall\, p \in \mathcal{P}$. This leads to a due date tightness that is roughly at the same level for different values

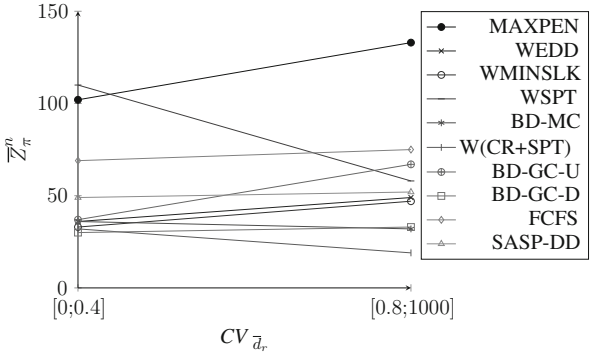

Fig. 6.5 Impact of $CV_{\overline{d}_r}$ on policy performance

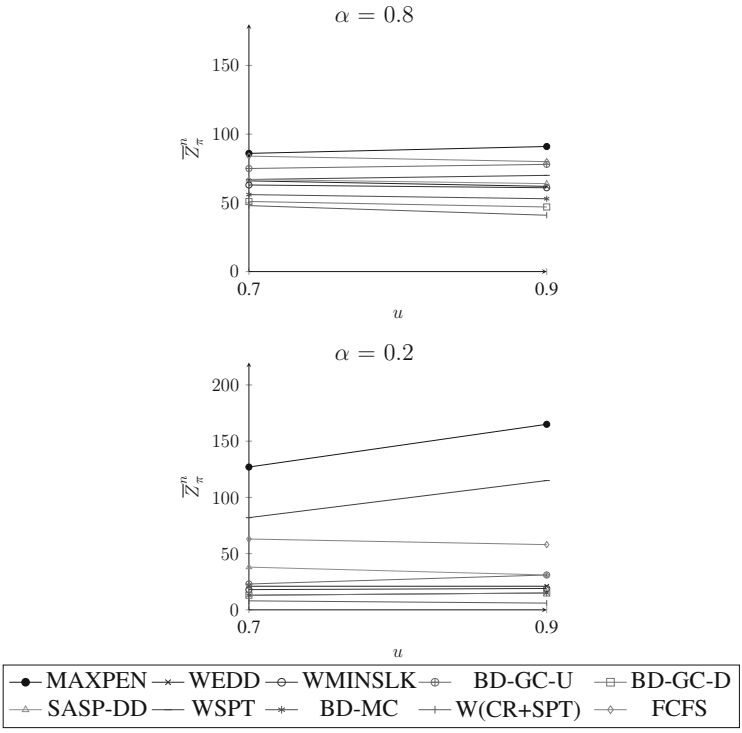

Fig. 6.6 Impact of the number of *OS* on policy performance

of u. By contrast, in the literature due dates are set without taking into account resource utilization. As a consequence increasing utilization leads in many cases to an increasing tightness such that observations similar to Sect. 6.3.1 can be made.

Furthermore, we observe that when increasing u the gaps between the objective values slightly increase. This can be explained by increasing queue lengths amplifying the effect of the priority policies.

6.3.6 *Observations for Problem Instances with a Single Project Type*

We also have investigated instances with a single project type with $w_1 = 1$. Essentially, the effects observed for two project types could be confirmed also for the case of a single project type. However, we have found that in many cases BD-GC-U performs much better such that the impact of different resource pricing schemes (uniform and dynamic pricing) is weaker. A possible explanation is the fact that projects have a more similar flow through the system and have the same weights/costs per time unit when the due date is not met. Thus, judgements whether it is beneficial to expedite an activity is more determined by the remaining activities to be scheduled.

6.4 Detailed Analysis

In the following, we consider the results in more detail with the objective to identify cases where certain policies can be recommended. Primarily, the recommendations are based on the performance of the policies. However, we additionally pay attention to policies having less need for information such that they are easier to implement.

When presenting the results, we show, in addition to $\overline{Z}^n(\pi)$, the mean ranks $\overline{R}(\pi)$. This helps to verify that low (high) $\overline{Z}^n(\pi)$ are not due to exceptional low (high) values $Z^n(\pi)$ for some instances.

The analysis is composed of two parts. At first, we consider two extreme cases where recommendations w.r.t. priority policies to be used are especially clear. The first case comprises problem instances with only one resource ($|\mathcal{R}| = 1$). The second case comprises problem instances with purely parallel networks ($OS_p = 0 \; \forall p \in \mathcal{P}$).

In the second part, we analyze the results for the remaining problem instances.

6.4.1 *Performance for Special Cases*

6.4.1.1 Single Resource

Table 6.4 shows the results for the case with one resource processing all activity types ($|\mathcal{R}| = 1$). For this case, BD-GC-D and BD-GC-U perform best with WEDD

Table 6.4 Mean weighted tardiness values and mean ranks for the case with 1 resource

Policy	$\overline{Z}_{\pi}^{n}$ (%)	$\overline{R}_{\pi}$
BD-GC-U	22.6	1.4
BD-GC-D	22.6	1.3
WEDD	26.6	3.0
WMINSLK	28.2	4.1
SASP-DD	35.3	6.2
BD-MC	38.4	5.8
W(CR+SPT)	40.0	6.7
MAXPEN	76.1	8.3
FCFS	78.8	8.6
WSPT	91.8	9.3

Table 6.5 Mean weighted tardiness values and mean ranks for the case with $OS = 0$

Policy	$\overline{Z}_{\pi}^{n}$ (%)	$\overline{R}_{\pi}$
BD-GC-U	19.6	1.7
BD-GC-D	19.5	1.3
WEDD	23.2	3.7
WMINSLK	23.9	4.6
FCFS	28.2	6.0
BD-MC	29.3	5.6
W(CR+SPT)	29.3	5.8
SASP-DD	31.3	7.1
WSPT	81.9	9.7
MAXPEN	86.4	9.3

and WMINSLK being the best followers. These observations are consistent with the theoretical insights discussed in Sect. 6.3.2.

6.4.1.2 Parallel Networks

For the case of parallel networks without precedence relations ($OS_p = 0 \ \forall p \in \mathcal{P}$), BD-GC-U and BD-GC-D perform best with WEDD and WMINSLK being the best followers. Furthermore, FCFS may be an easy-to-implement alternative having only a modest need for information (Table 6.5).

6.4.2 Performance for the Remaining Problem Instances

For ease of presentation, we number the intervals for $CV_{\overline{d}_r}$ as follows – 1 for interval [0; 0.4] and 2 for interval [0.8; 1,000].

As a first step, we have grouped the problem instances by due date tightness. As a second step, we have identified, for each of the two groups, five subsets (10 subsets in total) covering different ranges of parameter values. Table 6.6 shows $\overline{Z}_{\pi}^{n}$ and $\overline{R}_{\pi}$

Table 6.6 Mean average weighted tardiness values and ranks for high tightness

Subset	1		2		3		4		5	
$\lvert\mathcal{R}\rvert$	3–5		3–5		3–5		10–20		10–20	
OS	0.2–0.4		0.6–1		0.6–1		0.2–1		0.2–1	
$CV_{\overline{d}}$	1–2		1		2		1		2	
	$\overline{Z}_\pi^n$ (%)	$\overline{R}_\pi$	$\overline{Z}_\pi^n$ (%)	$\overline{R}_\pi$	$\overline{Z}_\pi^n$ (%)	$\overline{R}_\pi$	$\overline{Z}_\pi^n$ (%)	$\overline{R}_\pi$	$\overline{Z}_\pi^n$ (%)	$\overline{R}_\pi$
BD-GC-U	40.7	2.7	52.1	3.2	56.8	5.5	89.7	6.9	161.8	8.5
BD-GC-D	37.2	1.4	49.0	1.5	43.4	3.0	62.4	2.2	66.9	3.9
BD-MC	49.8	6.1	62.2	5.5	40.6	3.0	65.6	3.8	49.2	2.7
W(CR+SPT)	47.1	4.9	58.9	4.1	29.7	1.1	56.6	1.7	17.2	1.0
WMINSLK	40.6	3.3	52.8	3.2	51.2	5.6	66.1	3.6	105.0	6.6
WEDD	43.1	4.5	56.5	5.1	52.4	6.3	72.2	5.7	110.0	7.7
WSPT	71.9	8.7	86.1	8.5	47.0	5.4	93.2	8.6	37.2	2.9
MAXPEN	76.1	8.8	79.4	8.3	85.5	9.0	90.8	8.5	124.4	8.2
FCFS	83.2	9.4	91.6	9.5	96.5	9.6	83.3	7.3	96.2	6.6
SASP-DD	45.3	5.3	60.7	6.2	51.1	6.3	75.6	6.7	103.0	6.8

Table 6.7 Mean average tardiness values and ranks for low tightness

Subset	6		7		8		9		10	
$\lvert\mathcal{R}\rvert$	3–5		3–5		10–20		10–20		10–20	
OS	0.2–1		0.2–1		0.2–1		0.2–0.4		0.6–1	
$CV_{\overline{d}}$	1		2		1		2		2	
	$\overline{Z}_\pi^n$(%)	$\overline{R}_\pi$	$\overline{Z}_\pi^n$(%)	$\overline{R}_\pi$	$\overline{Z}_\pi^n$(%)	$\overline{R}_\pi$	$\overline{Z}_\pi^n$(%)	$\overline{R}_\pi$	$\overline{Z}_\pi^n$(%)	$\overline{R}_\pi$
BD-GC-U	10.6	3.0	8.3	3.6	13.7	4.2	42.2	5.5	121.5	7.2
BD-GC-D	10.1	1.9	7.9	2.9	12.0	3.0	15.6	3.4	38.9	4.2
BD-MC	11.3	4.2	7.5	3.1	11.4	2.6	12.2	2.4	30.0	3.0
W(CR+SPT)	7.9	2.1	3.8	1.4	4.9	1.0	1.4	1.0	0.5	1.0
WMINSLK	11.6	4.2	9.5	5.0	13.9	4.4	24.6	5.5	57.6	5.9
WEDD	13.5	5.6	10.2	5.6	17.6	5.8	27.9	6.2	67.2	7.0
WSPT	144.4	9.7	70.5	8.5	149.8	9.4	40.1	6.9	23.7	3.9
MAXPEN	122.0	9.3	136.0	9.9	148.7	9.6	236.4	10.0	243.1	9.9
FCFS	70.6	8.0	73.2	8.6	57.4	7.7	53.9	7.9	79.8	7.3
SASP-DD	32.2	7.0	18.6	6.5	47.9	7.3	39.6	6.3	59.2	5.8

for the different policies and subsets for high due date tightness and Table 6.7 for low due date tightness.

For both levels of tightness, we make the general observation that policies based on due date information outperform, for most cases, policies without due date information, such as FCFS, MAXPEN. Furthermore, we observe that WMINSLK outperforms WEDD such that we conclude that slack information, although being only a rough cut estimate, has a benefit over the due date.

In the following, we discuss the more specific observations made for each level of due date tightness.

6.4.2.1 High Due Date Tightness

For low values of $|\mathcal{R}|$ and *OS* (subset 1), low $|\mathcal{R}|$, low $CV_{\overline{d}}$ and higher values for *OS* (subset 2), BD-GC-D, BD-GC-D and WMINSLK are the best alternatives.

Furthermore, for the remaining subsets (subset 3–5) with higher values for $|\mathcal{R}|$ and $CV_{\overline{d}_r}$, W(CR+SPT) performs best. Furthermore, BD-GC-D and BD-MC perform well. For high numbers of resources and high variation of expected activity durations (subset 5), WSPT is a good easy-to-implement policy.

6.4.2.2 Low Due Date Tightness

W(CR+SPT) performs best for almost all subsets (except for subset 1 where its average rank is slightly higher than that of BD-GC-D). As alternatives, BD-GC-D and BD-MC exhibit a reasonably good performance while BD-GC-U can only be recommended for small $|\mathcal{R}|$. WMINSLK can be considered a good alternative as long as we do not have high values for $|\mathcal{R}|$ and $CV_{\overline{d}_r}$ (subsets 6–8).

Finally, WSPT is a good alternative for high $|\mathcal{R}|$, $CV_{\overline{d}_r}$ and *OS* (subset 10) where it outperforms all policies but W(CR+SPT).

Chapter 7
Optimal and Near Optimal Scheduling Policies

This chapter is dedicated to models and methodologies for the computation of optimal and near optimal policies.

To simplify analysis, we ignore due dates and restrict our consideration to flow times of projects (due dates are set equal to the arrival times of the projects). Furthermore, we assume that activity durations are exponentially distributed.

As a starting point, we model in Sect. 7.1 the non-preemptive and the preemptive scheduling problem outlined in Chap. 2 as Markov decision processes. Furthermore, we show how problem structure can be exploited to reduce the computational burden and give complexity results in terms of state space cardinality. Finally, a numerical example is presented in order to discuss structural properties that are typical for optimal policies.

Section 7.2 considers the special case where the system consists of a single resource. We have found that a priority index policy is optimal. Indices can be obtained without much computational effort.

The focus of Sect. 7.3 is on the class of *project state ordering policies* (POPs), which helps to considerably reduce state space cardinality. At the same time the policies remain optimal for many cases. Basic structural results are provided for the preemptive and the non-preemptive problem. For the preemptive problem, we quantify more precisely the state space reduction via an equivalence of the state space obtained for the preemptive scheduling problem to the state space of a queueing network.

In Sect. 7.4, we consider a number of approximate dynamic programming (ADP) approaches to remedy the curse of dimensionality and extend the computation of optimal policies to open systems with unbounded $K^{\max}$.

The performance optimal policies and the methodologies for their computation are investigated in Sect. 7.5 based on an extensive computational study.

P. Melchiors, *Dynamic and Stochastic Multi-Project Planning*, Lecture Notes in Economics and Mathematical Systems 673, DOI 10.1007/978-3-319-04540-5_7,

7.1 Models as a Markov Decision Process

In this section, we formulate, as a first step, in Sect. 7.1.1 the CTMDP for the *non-preemptive scheduling problem* according to the assumptions made in the Chap. 2. Secondly, we present in Sect. 7.1.2 the CTMDP for the scheduling problem where preemptions are allowed (*preemptive scheduling problem*).

7.1.1 Non-preemptive Scheduling Problem

The framework for the description has been taken from Powell [106].

7.1.1.1 Markov Decision Process

State variables A system state $s(t)$ at time t is characterized by the states of the projects in the system. The state $\sigma(j,t)$ of a project $j \in \mathcal{J}(t)$ is given by

$$\sigma(j,t) = (p, \mathcal{W}(j,t), \mathcal{E}(j,t)) \tag{7.1}$$

The index p is needed to address information depending on project type p, such as precedence relations, expected activity durations. $\mathcal{W}(j,t)$ is the set of waiting activities and $\mathcal{E}(j,t)$ the set of activities in process of project j at time t. In the following, we use $p(\sigma)$, $\mathcal{W}(\sigma)$ and $\mathcal{W}(\sigma)$ to refer to the three elements of project state σ. Based on $\mathcal{W}(\sigma)$ and $\mathcal{E}(\sigma)$ we define $\mathcal{U}(\sigma)$ the set of unfinished activities of a project in state σ which comprises $\mathcal{W}(\sigma) \cup \mathcal{E}(\sigma)$ and all direct and indirect successors. Note that $\mathcal{W}(\sigma) \cup \mathcal{E}(\sigma)$ form an *antichain* being a subset of $\mathcal{V}_{p(\sigma)}$ without precedence relations between the nodes. Furthermore, a project state σ is analogous to the concept of an *uniformly directed cut* (UDC) as used in Kulkarni and Adlakha [79]. A UDC is in the context of *activity-on-arc*-networks a set of arcs representing activities without precedence relations preventing parallel execution. The set of states that a project may adopt between arrival and completion is bounded by two special project states: The initial state σ_p^{I} immediately entered after the arrival where no activities can be in process or be completed (such that $\mathcal{U}(\sigma_p^{\mathrm{I}}) = \mathcal{V}_p$) and the absorbing state $\sigma_p^{\mathrm{F}} = (p, \emptyset, \emptyset)$ entered on completion of the last activity where $\mathcal{U}(\sigma_p^{\mathrm{F}}) = \emptyset$.

By Σ, we refer to the set of all feasible project states for projects being in the system where a project state σ is feasible if no precedence or resource constraints are violated. The finite number of activity types (i, p) imply a finite set Σ. As projects entering project state σ_p^{F} leave immediately the system, σ_p^{F} is not element of Σ.

The number of state variables can be reduced by exploiting that it is not necessary to distinguish between individual projects being in the same project state σ. All projects in the same state are identical in terms of duration distributions and resource

types of activities, precedence relations, sets of waiting activities and activities in process such that the projects possess the same stochastic properties essential for the CTMDP. Hence, we define variables $n(\sigma, t)$ for the number of projects in project state σ at time t. Then, a system state at time t is given by

$$s(t) = \mathbf{n}(t) = \big(n(\sigma_1, t), n(\sigma_2, t), \ldots, n(\sigma_{|\Sigma|}, t)\big) \tag{7.2}$$

As a special system state, we define $s^0 = (0, \ldots, 0)$ where the system is empty. As we consider stationary policies we drop the time index in the following. Using $n(\sigma, s)$, we refer, in the following, to the number of projects in project state σ while the system is in state s.

We define

$$\Sigma(p, s) = \{\sigma \in \Sigma | p(\sigma) = p \wedge n(\sigma, s) > 0\} \tag{7.3}$$

to be the subset of project states $\sigma \in \Sigma$ with $p(\sigma) = p$ and $n(\sigma, s) > 0$.

The set $\mathcal{S}$ containing all *feasible* system states s where decisions are to be made (pre-decision states) defines the *state space*. Feasibility of a state s is given by the following definition.

Definition 7.1.1. A system state s is *feasible* if there exists a policy under which it is *accessible* from system state s^0.

Finally, we define $K(s)$ to be the number of projects in the system for system state s as given by

$$K(s) = \sum_{\sigma \in \Sigma} n(\sigma, s) \tag{7.4}$$

Note that the post-decision states are considered explicitly for the construction of the transitions. However, a post-decisions state is only in the state space $\mathcal{S}$ if there are occasions where it is also pre-decision state.

Decision variables For the purpose of generating alternative decisions, we number the projects in the system by $k = 1, \ldots, K(s)$ where project states σ are ordered lexicographically according to the following criteria.

1. Project type $p(\sigma)$.
2. Set of waiting activities $\mathcal{W}(\sigma)$.
3. Set of activities in process $\mathcal{E}(\sigma)$.

Between projects in the same state σ, ties can be broken arbitrarily as they are equivalent. Then, $\sigma(k, s)$ refers to the state of project k in system state s and tuple (i, k) refers to activity i of project k to be scheduled. Furthermore, $\mathcal{B}^{\mathrm{S}}(r, s)$ is the set of activities given by tuples (i, k) to be scheduled in system state s on resource type $r \in \mathcal{R}$. Then, a decision a is given by a tuple of sets $\mathcal{B}^{\mathrm{S}}(r, s)$ as follows

$$a = \left(\mathcal{B}^{\mathrm{S}}(r_1, s), \ldots, \mathcal{B}^{\mathrm{S}}(r_{|\mathcal{R}|}, s)\right) \tag{7.5}$$

By $\mathcal{B}^{\mathrm{S}}(r, s, a)$, we refer to the set of activities to be scheduled by decision a on resource type $r \in \mathcal{R}$. The set $\mathcal{A}(s)$ of alternative decisions for system state s is generated in three steps.

1. Determine the set $\mathcal{W}(r, s)$ of activities waiting for resource type r in system state s as given by

$$\mathcal{W}(r, s) = \bigcup_{k=1}^{K(s)} \left\{(i, k) | i \in \mathcal{W}(\sigma(k, s)) \wedge r_{ip(\sigma(k,s))} = r\right\} \tag{7.6}$$

   The definition of $\mathcal{W}(r, s)$ states that it consists of waiting activities from the projects $k = 1, \ldots, K(s)$ where the resource type $r_{ip(\sigma(k,s))}$ required for processing corresponds to the considered resource type r.
2. Determine the set $\mathcal{A}(r, s)$ of all feasible sets $\mathcal{B}^{\mathrm{S}}(r, s)$. Note that if $|\mathcal{W}(r, s)| > f(r, s)$ only subsets $\mathcal{B}^{\mathrm{S}}(r, s) \subset \mathcal{W}(r, s)$ can be scheduled at a decision time. Then, a set $\mathcal{A}(r, s)$ is given by

$$\mathcal{A}(r, s) = \left\{\mathcal{B}^{\mathrm{S}}(r, s) \subseteq \mathcal{W}(r, s) | \; |\mathcal{B}^{\mathrm{S}}(r, s)| = \min\left\{f(r, s), |\mathcal{W}(r, s)|\right\}\right\} \tag{7.7}$$

   where $f(r, s)$ refers to the number of idle resources of type r in system state s. If $\mathcal{W}(r, s) = \emptyset$, an empty set $\mathcal{B}^{\mathrm{S}}(r, s) = \emptyset$ is added to $\mathcal{A}(r, s)$. The constraint $|\mathcal{B}^{\mathrm{S}}(r, s)| = \min\left\{f(r, s); |\mathcal{W}(r, s)|\right\}$ on the cardinality of the sets $\mathcal{B}^{\mathrm{S}}(r, s)$ states that either all waiting activities may be selected or just as many as there are idle resources of type r. As the constraint requires to schedule as many activities as possible, only decisions being in line with a *non-idling* policy are generated.
3. Determine $\mathcal{A}(s)$ by the cartesian product of the sets $\mathcal{A}(r, s)$ as given by

$$\mathcal{A}(s) = \bigtimes_{r \in \mathcal{R}} \mathcal{A}(r, s) \tag{7.8}$$

   For the case that no activities are available to be scheduled, a single decision consisting of empty sets is obtained. Thus, the decision has no effect on the system. This may occur in different situations. Firstly, if there are no idle resources of the required types for processing any of the waiting activities. This may be the case on arrivals of new projects. Secondly, if there are no waiting activities since all activities are in process and successors of the activity just completed have some yet unfinished predecessors. Finally, if the system is empty.

Exogenous information process Exogenous information arrives when an event occurs while the system is in a post-decision state $\hat{s}$. For our problem, we identify two kinds of events.

1. Arrival of a new project of type p – This kind of event occurs at a rate of λ_p. By means of project type p, we know the network structure, the expected durations and the required resource types for processing the activities.
2. Completion of activity i of a project being in project state σ – As such an even may occur for any of the $n(\sigma, \hat{s})$ projects being in project state σ this event occurs at a rate $n(\sigma, \hat{s}) \cdot \mu_{ip(\sigma)}$.

Transition function The transitions from system state $s \in \mathcal{S}$ to another system state $s' \in \mathcal{S}$ when having selected decision $a \in \mathcal{A}(s)$ can be constructed as follows. At first, we describe the effect of a decision a leading to a *post-decision state* $\hat{s}(s, a)$. Afterwards, we present in detail the effects of events leading to transitions to subsequent system states $s' \in \mathcal{S}$.

Before going into detail, we give some definitions.

- σ_p^{I} is the initial project state when a new project arrives at the system. σ_p^{I} only contains activities i in $\mathcal{W}(\sigma_p^{\mathrm{I}})$ having no predecessors such that $(i', i) \notin \mathcal{A}_p \ \forall i \in \mathcal{W}(\sigma_p^{\mathrm{I}}), i' \in \mathcal{V}_p$. Furthermore, no activities are in process such that $\mathcal{E}(\sigma_p^{\mathrm{I}}) = \emptyset$.
- $\sigma^{\mathrm{C}}(i, \sigma)$ is the project state subsequent to the completion of activity i of a project in state σ (all other activities in $\mathcal{E}(\sigma)$ continue being in process). Each direct successor i' of i (such that $(i, i') \in \mathcal{A}_{p(\sigma)}$) is checked whether it is ready in the sense that all its predecessors have been completed. In the positive case i' is added to $\mathcal{W}(\sigma^{\mathrm{R}}(i, \sigma))$ being the set of direct successors that become ready for being scheduled on completion of activity i in project state σ. $\mathcal{W}(\sigma^{\mathrm{R}}(i, \sigma))$ is given by

$$\mathcal{W}^{\mathrm{R}}(i, \sigma) = \{i' \in \mathcal{U}(\sigma) \backslash \{i\} | \forall (i'', i') \in \mathcal{A}_p : i'' \notin \mathcal{U}(\sigma) \backslash \{i\} \wedge (i, i') \in \mathcal{A}_p\} \tag{7.9}$$

  Then, we have

$$\sigma^{\mathrm{C}}(i, \sigma) = \left(p(\sigma), \mathcal{W}(\sigma) \cup \mathcal{W}^{\mathrm{R}}(i, \sigma), \mathcal{E}(\sigma) \backslash \{i\}\right) \tag{7.10}$$

- $\sigma^{\mathrm{S}}(\mathcal{B}, \sigma)$ is the project state subsequent to scheduling all activities given by index set $\mathcal{B}$ of a project in state σ. It is given by

$$\sigma^{\mathrm{S}}(\mathcal{B}, \sigma) = (p(\sigma), \mathcal{W}(\sigma) \backslash \mathcal{B}, \mathcal{E}(\sigma) \cup \mathcal{B}) \tag{7.11}$$

In order to obtain the post-decision state $\hat{s}(s, a)$ subsequent to pre-decision state $s \in \mathcal{S}$ we compute the index sets

$$\mathcal{B}^{\mathrm{S}}(k, s, a) = \{i | (i, k) \in \bigcup_{r \in \mathcal{R}} \mathcal{B}^{\mathrm{S}}(r, s, a)\} \tag{7.12}$$

of all activities of project $k \in \{1, \ldots, K(s)\}$ scheduled by decision a in system state s.

Then, we define

$$\hat{n}(\sigma, s, a) = \sum_{k=1}^{K(s)} \mathbb{1}\left\{\sigma^{\mathrm{S}}(\mathcal{B}^{\mathrm{S}}(k, s, a), \sigma(k, s)) = \sigma\right\} \tag{7.13}$$

to be the number of projects in state σ when the system is in the post-decision state $\hat{s}(s, a)$. If $\mathcal{B}^{\mathrm{S}}(k, s, a) = \emptyset$ then we clearly have $\sigma^{S}(\mathcal{B}^{\mathrm{S}}(k, s, a), \sigma(k, s)) = \sigma(k, s)$. Now, the post-decision state $\hat{s}(s, a)$ is given by

$$\hat{s}(s, a) = \left(\hat{n}(\sigma_1, s, a), \ldots, \hat{n}(\sigma_{|\Sigma|}, s, a)\right) \tag{7.14}$$

The total rate $\beta(s, a)$ of the exponentially distributed transition time is obtained by

$$\beta(s, a) = \sum_{\sigma \in \Sigma} \sum_{i \in \mathcal{E}(\sigma)} \mu_{ip(\sigma)} \cdot n(\sigma, \hat{s}(s, a)) + \sum_{p \in \mathcal{P}} \lambda_p \tag{7.15}$$

Next, to obtain a system state $s' \in \mathcal{S}$ subsequent to post-decision state $\hat{s}(s, a)$, we develop the transitions due to the two kinds of events (arrivals and activity completions).

1. Arrival of a new project of type $p \in \mathcal{P}$ – Arrivals may occur as long as $K(s) < K^{\max}$. The pre-decision state $s^{\mathrm{A}}(s, a, p)$ that is entered on arrival and acceptance of a project of type $p \in \mathcal{P}$ subsequent to making decision a in pre-decision state s is given by

$$s^{\mathrm{A}}(s, a, p) = \hat{s}(s, a) + \mathbb{1}\{K(s) < K^{\max}\}\, \boldsymbol{e}(\sigma_p^I) \tag{7.16}$$

   where $\boldsymbol{e}(\sigma)$ is a unity vector having a value of 1 at the position of σ. We observe that $s^{\mathrm{A}}(s, a, p) = \hat{s}(s, a)$ if $K(s) = K^{\max}$. Thus, $\hat{s}(s, a)$ is also a pre-decision state such that $\hat{s}(s, a) \in \mathcal{S}$. The probability of a transition related to an arrival event is given by

$$q^{\mathrm{A}}(s, a, p) = \frac{\lambda_p}{\beta(s, a)} \tag{7.17}$$

   Furthermore, we have a fixed cost of the transition

$$k^{\mathrm{A}}(s, p) = \mathbb{1}\{K(s) = K^{\max}\} \cdot y_p \tag{7.18}$$

   which is incurred only if a project must be rejected as $K(s) = K^{\max}$.
2. Completion of activity i of a project in state σ – The pre-decision state $s^{\mathrm{C}}(s, a, i, \sigma)$, entered on completion of activity $i \in \mathcal{E}(\sigma)$ of a project in state σ if decision a was selected before in state s, is given by

$$s^{\mathrm{C}}(s,a,i,\sigma) = \hat{s}(s,a) - \mathbb{1}\{n(\sigma,\hat{s}(s,a)) > 0\}\,\boldsymbol{e}(\sigma) + \mathbb{1}\left\{n(\sigma,\hat{s}(s,a)) > 0 \wedge \sigma^{C}(i,\sigma) \neq \sigma_p^F\right\}\boldsymbol{e}(\sigma^{C}(i,\sigma)) \quad (7.19)$$

Note that if $n(\sigma,\hat{s}(s,a)) = 0$ we have $s^{\mathrm{C}}(s,a,i,\sigma) = \hat{s}(s,a)$. Furthermore, if a project of type p enters its absorbing state σ_p^F, on completion of its last activity, no increment of the number of projects in state σ_p^F takes place as the project leaves the system.

The probability of a transition related to the completion of activity $i \in \mathcal{E}(\sigma)$ of a project in state σ is given by

$$q^{\mathrm{C}}(s,a,i,\sigma) = \frac{\mu_{ip(\sigma)} \cdot n(\sigma,\hat{s}(s,a))}{\beta(s,a)} \quad (7.20)$$

Cost function The *total holding cost per time unit* subsequent to a pre-decision state s is given by

$$c(s) = \sum_{\sigma\in\Sigma} w_{p(\sigma)} \cdot n(\sigma,s) \quad (7.21)$$

Note that the holding cost per time unit only depends on the pre-decision state as the set of projects in the system does not change when making scheduling decisions. However, the expected holding cost until the next pre-decision state s is given by $\frac{c(s)}{\beta(s,a)}$ and hence also depend on the decision.

Furthermore, we have fixed cost $k(s,a,s')$ which may be incurred on a transition between two pre-decision states s and s'. On arrival of a new project of type p, we have $k(s,a,s') = k^{\mathrm{A}}(s,p)$. On completion of an activity, we have $k(s,a,s') = 0$.

Objective function In order evaluate a given policy π, we consider the long term average cost per time unit as given by

$$g(\pi) = \liminf_{N\to\infty} g(\pi) = \frac{E\left[\sum_{n=0}^{N-1}\left(c(s_n)\boldsymbol{\tau}(s_n,\pi(s_n)) + k(s_n,a_n,s_{n+1})\right)\right]}{E\left[\sum_{n=1}^{N}\boldsymbol{\tau}(s_n,\pi(s_n))\right]} \quad (7.22)$$

n denotes the number of the visited state and $\boldsymbol{\tau}(s,a)$ denotes the transition time being exponentially distributed with rate $\beta(s,a)$. Then, an optimal policy is given by

$$\pi^* = \arg\min_{\pi\in\Pi}\{g(\pi)\} \quad (7.23)$$

Thus, $g^* = g(\pi^*)$ is the minimum cost yielded under an optimal policy π^*. The equivalence of the average cost per project and the average cost per time unit becomes obvious when considering a sample path associated with an arrival stream of J projects. Be $N(J)$ the number of transitions of the sample path.

We assume that the sample path starts at an empty system $s_0 = s^0$ and ends at an empty system $s_{N(J)} = s^0$. Then, we have for the rejection cost

$$E\left[\sum_{j=0}^{J} \mathbb{1}\left\{|\mathcal{J}(\boldsymbol{t}_j^a)| = K^{\max}\right\} \cdot y_{\boldsymbol{p}_j}\right] = E\left[\sum_{n=1}^{N(J)} k(s_n, a_n, s_{n+1})\right] \tag{7.24}$$

and for the holding costs

$$E\left[\sum_{j=1}^{J} w_{\boldsymbol{p}_j} \boldsymbol{F}_j\right] = E\left[\sum_{n=1}^{N(J)} c(s_n)\boldsymbol{\tau}_n(s_n, a_n)\right] \tag{7.25}$$

where $w_{\boldsymbol{p}_j} \boldsymbol{F}_j$ is the total holding cost of project j. Thus, we obtain

$$\begin{aligned} &E\left[\sum_{j=1}^{J}\left(\mathbb{1}\left\{|\mathcal{J}(t_j^a)| = K^{\max}\right\} \cdot y_{\boldsymbol{p}_j} + w_{\boldsymbol{p}_j} \boldsymbol{F}_j\right)\right] \\ &\quad = \left[\sum_{n=1}^{N(J)} \left(k(s_n, a_n, s_{n+1}) + c(s_n)\boldsymbol{\tau}(s_n, a_n)\right)\right] \end{aligned} \tag{7.26}$$

Whether we divide the right or the left-hand side by the number of projects J for obtaining the cost per project or by the expected period length $E\left[\sum_{n=1}^{N(J)} \boldsymbol{\tau}(s_n, \pi(s_n))\right] > 0$ for obtaining the cost per time unit clearly has no impact on the optimization. Hence, the two objectives are equivalent.

Finally, we remark that the restriction to non-idling policies helps to reduce computational complexity by the fact that the decision sets $\mathcal{A}(s)$ as well as the state space $\mathcal{S}$ become smaller. If idleness were allowed we may have more system states where some of the resources idle. At the same time, as we see later, the state space $\mathcal{S}$ for the problem without preemptions may grow very large, even for small problems. Unfortunately, we were not able to quantify in more precisely the cardinality of $\mathcal{S}$. However, we will quantify more precisely the state space cardinality for the preemptive problem. Furthermore, we will see in Sect. 7.4.2 that the state space cardinality for the preemptive problem with preemptions is a lower bound of the state space cardinality for the non-preemptive problem.

7.1.1.2 Evaluation and Optimality Equations

Firstly, we show that under any stationary policy π there exists, independently from the starting state of a sample path, a single value $g(\pi)$ for the long term average

cost. This property is important for the formulation of evaluation and optimality equations below and for the application of solution methodologies as presented in Chap. 4. Recall from Chap. 4 that the existence is guaranteed if a CTMDP is *unichain* (cf. Puterman [108]) such that there exist under any stationary policy a single *recurrent* class of states and a (possibly) empty set of transient states. The following Theorem gives that the CTMDP is unichain for the dynamic-stochastic scheduling problem.

Theorem 7.1.1. *Under any stationary scheduling policy π that avoids total idleness, the CTMDP for the dynamic stochastic multi-project scheduling problem is unichain.*

Proof. At first, we note that by avoiding total idleness at least one activity is processed on any resource type except that the system is empty ($K(s) = 0$). Hence, there is always at least one resource busy as long as $K(s) > 0$. Furthermore, with non-zero probability no new project arrives for the entire period until the system becomes empty such that system state $s^0 \in \mathcal{S}$ is *accessible* from any system state. Conversely, under every stationary policy π, a subset $\mathcal{S}(\pi) \subseteq \mathcal{S} \backslash s^0$ is *accessible* from s^0 such that all $s \in \mathcal{S}(\pi)$ communicate with s^0 and $\mathcal{S}(\pi) \cup \{s^0\}$ is a *recurrent* class of states. Since s^0 is accessible from all *transient* states $s \in \mathcal{S} \backslash \mathcal{S}(\pi)$ it is the only recurrent class. □

As non-idling policies always avoid total idleness the Theorem applies to the CTMDP for the non-preemptive problem. Then, the set of evaluation equations (Poisson's equations) is given by

$$h(s) = \frac{c(s) - g}{\beta(s, \pi(s))} + \sum_{p \in \mathcal{P}} \left[q^{\mathrm{A}}(s, \pi(s), p) \left(k^{\mathrm{A}}(s, p) + h(s^{\mathrm{A}}(s, \pi(s), p)) \right) \right] + \sum_{\sigma \in \Sigma} \sum_{i \in \mathcal{E}} q^{\mathrm{C}}(s, \pi(s), i, \sigma) h(s^{\mathrm{C}}(s, \pi(s), i, \sigma)) \quad \forall s \in \mathcal{S} \tag{7.27}$$

and the set of optimality equations (Bellman equations) is given by

$$h(s) = \min_{a \in \mathcal{A}(s)} \left\{ \frac{c(s) - g}{\beta(s, a)} + \sum_{p \in \mathcal{P}} \left[q^{\mathrm{A}}(s, a, p) \left(k^{\mathrm{A}}(s, p) + h(s^{\mathrm{A}}(s, a, p)) \right) \right] + \sum_{\sigma \in \Sigma} \sum_{i \in \mathcal{E}} q^{\mathrm{C}}(s, a, i, \sigma) h(s^{\mathrm{C}}(s, a, i, \sigma)) \right\} \quad \forall s \in \mathcal{S} \tag{7.28}$$

For the solution, we set $h(s^0) = 0$ as s^0 is in the recurrent class under any stationary policy π that avoids total idleness.

7.1.1.3 Elimination of Scheduling Decisions

A subproblem is finding an optimal decision $a^* \in \mathcal{A}(s)$ for all system states $s \in \mathcal{S}$. As $|\mathcal{A}(s)|$ may be large the solution of the subproblem by fully enumerating and evaluating all alternative decisions $a \in \mathcal{A}(s)$ may slow down the procedure. Thus, we eliminate scheduling decisions by truncating for any system state $s \in \mathcal{S}$ the number of projects in project state σ. We define the set $\mathcal{R}(\sigma)$ of resource types which are demanded by at least one waiting activity of a project in state σ.

$$\mathcal{R}(\sigma) = \{r \in \mathcal{R} | \exists i \in \mathcal{W}(\sigma) \wedge r_{ip(\sigma)} = r\} \tag{7.29}$$

In order to eliminate decisions we observe that in the worst case we assign to each idle resource of type $r \in \mathcal{R}(\sigma)$ no more than one activity from a project in state σ such that any two activities $i, i' \in \mathcal{W}(\sigma)$ are not from the same project. Hence, we need consider at most as many projects in state σ as there are idle resources of the resource types $r \in \mathcal{R}(\sigma)$ such that we can truncate the number of projects in state σ to

$$n^{\mathrm{Tr}}(\sigma, s) = \min \left\{ \sum_{r \in \mathcal{R}(\sigma)} f(r, s); n(\sigma, s) \right\} \leq n(\sigma, s) \tag{7.30}$$

Then, for the generation of decisions as outlined in Sect. 7.1.1.1, we consider a system state $s^{\mathrm{Tr}}(s) = \left(n^{\mathrm{Tr}}(\sigma_1, s), \ldots, n^{\mathrm{Tr}}(\sigma_{|\Sigma|}, s)\right)$ with truncated numbers of projects such that less alternative decisions exist.

Example 7.1.1. We have for system state s $n(\sigma, s) = 4$ and $n(\sigma', s) = 0$ $\forall \sigma' \in \Sigma \neq \sigma$. This implies that we have four projects $k_1, k_2, k_3, k_4 \in [1, \ldots, K(s)]$ being in the same project state $\sigma(k_1, s) = \sigma(k_2, s) = \sigma(k_3, s) = \sigma(k_4, s) = \sigma$. Furthermore, we have two waiting activities ($\mathcal{W}(\sigma) = \{i_1, i_2\}$) which are to be processed on resource type r ($r_{i_1 p(\sigma)} = r_{i_2 p(\sigma)} = r$) which has 1 idle resource ($f(r, s) = 1$). Obviously, no more than one activity can be scheduled. Thus, consideration of scheduling alternatives can be restricted to project k_1 such that $n^{\mathrm{Tr}}(\sigma, s) = \min\{1; 4\} = 1$.

7.1.2 Preemptive Scheduling Problem

In this section, we assume that activities may be preempted and rescheduled at no cost if an event occurs.

In Sect. 7.1.2.1, we discuss the CTMDP for the preemptive problem as an extension of the CTMDP for the non-preemptive problem. Based on this formulation, we exploit further structural properties to derive in Sect. 7.1.2.2 a simplified CTMDP without explicit consideration of post-decision states. Finally, for the simplified

CTMDP, we present in Sect. 7.1.2.3 an efficient procedure for determining an optimal scheduling decision.

7.1.2.1 Extension of the CTMDP for the Non-preemptive Problem

Generally, for preempting and rescheduling activities, we have to determine the sets $\mathcal{B}^{\mathrm{P}}(r,t)$ and $\mathcal{B}^{\mathrm{S}}(r,t)$ for all $r \in \mathcal{R}$ at a decision time t.

Exploiting the assumption that no cost is incurred due to preemptions and rescheduling, we simply shift back all activities in process to the set of waiting activities such that $\mathcal{B}^{\mathrm{P}}(r,t) = \mathcal{E}(r,t)$. Thus, only the sets $\mathcal{B}^{\mathrm{S}}(r,t)$ have to be optimized.

Furthermore, due to Theorem 2.2.2, decision times can be restricted without loss of optimality to times where activity completions or project arrivals occur.

To obtain the CTMDP for the preemptive problem, we define the following operation which shifts activities in process of a project in state σ back to the set of waiting activities

$$\sigma^{\mathrm{P}}(\sigma) = (p(\sigma), \mathcal{W}(\sigma) \cup \mathcal{E}(\sigma), \emptyset) \tag{7.31}$$

In order to adapt the CTMDP for the non-preemption problem (cf. Sect. 7.1.1), such that it becomes a CTMDP for the preemptive problem, we define the following operation which maps any state $s \in \mathcal{S}$ onto a state $s^{\mathrm{P}}(s) \in \mathcal{S}^{\mathrm{P}}$ where all activities are waiting. It is given by

$$s^{\mathrm{P}}(s) = \left(n^{\mathrm{P}}(\sigma_1, s), \ldots, n^{\mathrm{P}}(\sigma_{|\Sigma|}, s)\right) \tag{7.32}$$

where $n^{\mathrm{P}}(\sigma, s)$ is given by

$$n^{\mathrm{P}}(\sigma, s) = \begin{cases} \sum\limits_{\substack{\sigma' \in \Sigma \\ \sigma^{\mathrm{P}}(\sigma') = \sigma}} n(\sigma', s) & \sigma \in \Sigma^{\mathrm{P}} \\ 0 & \text{otherwise} \end{cases} \tag{7.33}$$

Σ^{P} is the set of feasible project states which may occur in the pre-decision states of the preemptive problem, where we have $\mathcal{E}(\sigma) = \emptyset \ \forall \sigma \in \Sigma^{\mathrm{P}}$.

By $\mathcal{S}^{\mathrm{P}}$, we refer to the space of all feasible pre-decision states for the preemptive problem. Then, for the transitions as outlined in Sect. 7.1.1.1 $s^{\mathrm{A}}(s,a,p)$ and $s^{\mathrm{C}}(s,a,i,\sigma)$ are replaced by $s^{\mathrm{P}}(s^{\mathrm{A}}(s,a,p))$ and $s^{\mathrm{P}}(s^{\mathrm{C}}(s,a,i,\sigma))$. The surjective mapping $s^{\mathrm{P}}(s)$ suggests that the state space $\mathcal{S}^{\mathrm{P}}$ of the preemptive problem is smaller than $\mathcal{S}$. In pre-decision states we no longer have to account for the activities being in process as they are all in the sets of waiting activities.

Furthermore, the preemptive problem can be considered as a *relaxation* of the non-preemptive problem as the activities being in process at a decision time represent a constraint on the decision sets $\mathcal{A}(s)$.

As, by assumption, total idleness is avoided such that at least one activity must be in process except the system is empty (in state s^0) Theorem 7.1.1 applies also to the preemptive problem.

7.1.2.2 Simplified CTMDP

The structure of the preemptive problem can exploited in order to obtain a simplified CTMDP where no explicit consideration of post-decision states is needed anymore. Furthermore, decisions have a simpler representation which leads to an efficient approach for determining the optimal decision without full enumeration and evaluation of all scheduling alternatives. Details are given in Sect. 7.1.2.3.

In order to see how the CTMDP, presented in Sect. 7.1.2.1, can be simplified, let us consider the effect of a decision $a \in \mathcal{A}(s)$ that schedules the set $\mathcal{B}^{\mathrm{S}}(k,s,a)$ of activities of project $k \in [1,\ldots,K(s)]$. Hence, for the post-decision state $\hat{s}(s,a)$, we have

$$\sigma^{\mathrm{S}}(\mathcal{B}^{\mathrm{S}}(k,s,a),\sigma) = \left(\mathcal{W}(\sigma)\backslash\mathcal{B}^{\mathrm{S}}(k,s,a), \mathcal{B}^{\mathrm{S}}(k,s,a), p(\sigma)\right) \tag{7.34}$$

Next, let us assume that the next event refers to the completion of activity (i,k) with $i \in \mathcal{B}^{\mathrm{S}}(k,s,a)$ such that the subsequent project state in system state s' is given by

$$\sigma^{\mathrm{P}}\left(\sigma^{\mathrm{C}}\left(i,\sigma^{\mathrm{S}}\left(\mathcal{B}^{\mathrm{S}}(k,s,a),\sigma\right)\right)\right) = \left(p(\sigma), \mathcal{W}(\sigma)\backslash\{i\} \cup \mathcal{W}^{R}(i,\sigma), \emptyset\right) \tag{7.35}$$

We observe that, except the completion of activity (i,k), the decision to schedule activities in $\mathcal{B}^{\mathrm{S}}(k,s,a)$ does not carry over to the next project state as we shift all activities not completed back to the set of waiting activities. Furthermore, for any other project $k' \in [1,\ldots,K(s)] \neq k$, we have

$$\sigma^{\mathrm{P}}\left(\sigma^{\mathrm{S}}\left(\mathcal{B}^{\mathrm{S}}\left(k',s,a\right),\sigma\right)\right) = \sigma \tag{7.36}$$

such that scheduling activities in $\mathcal{B}^{\mathrm{S}}(k',s,a)$ does not carry over as well.

Hence, for obtaining the transition related to the completion of an activity it is sufficient to know the activity index i and the state $\sigma(k,s)$ of the project $k \in [1,\ldots,K(s)]$ covered by a scheduling decision. The system state subsequent to the completion of activity i of a project in state σ is given by

$$s^{\mathrm{CP}}(s,i,\sigma) = s - \mathbf{e}(\sigma) + \mathbf{e}\left(\sigma^{\mathrm{CW}}(i,\sigma)\right) \tag{7.37}$$

where

$$\sigma^{\mathrm{CW}}(i,\sigma) = \left(p(\sigma), \mathcal{W}(\sigma)\backslash\{i\} \cup \mathcal{W}^{R}(i,\sigma), \emptyset\right) \tag{7.38}$$

Note, as $\sigma^{\mathrm{C}}(i,\sigma)$ requires $i \in \mathcal{E}(\sigma)$ we have introduced $\sigma^{\mathrm{CW}}(i,\sigma)$ which requires $i \in \mathcal{W}(\sigma)$.

On arrival of a new project of type $p \in \mathcal{P}$, all activities in process are shifted back to the sets of waiting activities. Hence, we have $n(\sigma_p^{\mathrm{I}}, s') = n(\sigma_p^{\mathrm{I}}, s) + 1$ and $n(\sigma, s') = n(\sigma, s)\ \forall \sigma \neq \sigma_p^{\mathrm{I}}$. In vector notation a transition subsequent to an arrival of a new project of type $p \in \mathcal{P}$ is given by

$$s^{\mathrm{AP}}(s, p) = s + \mathbb{1}\{K(s) < K^{\max}\}\,\mathbf{e}\left(\sigma^I(p)\right) \tag{7.39}$$

Recall that in case of the non-preemption problem, for each project $k \in [1, \ldots, K(s)]$, the effect of a scheduling decision, represented by the sets $\mathcal{B}^{\mathrm{S}}(k, s, a)$, carries over to the subsequent system states. This is due to the fact that all activities scheduled by decision a and that are not completed remain in process. This explains why scheduling decisions must be referred to individual projects which makes the generation of alternative decisions and transitions more involved.

By contrast, for the preemptive problem the only effect of a scheduling decision in system state s that carries over to subsequent states is the completion of activity i of a project in state σ. All other activities are preempted such that the fact they were scheduled is "forgotten".

As a consequence, we only have to distinguish between *groups of activities* identified by the tuples (i, σ) such that, for a scheduling decision, we simply have to specify a number $n(i, \sigma)$ of activities of group (i, σ) to be scheduled. A decision a can now be defined as a vector of the variables $n(i, \sigma)$.

$$a = \left(n(i, \sigma)\ \forall \sigma \in \Sigma^{\mathrm{P}}, i \in \mathcal{W}(\sigma)\right) \tag{7.40}$$

Then, rate $\beta(s, a)$ is given by

$$\beta(s, a) = \sum_{\sigma \in \Sigma^{\mathrm{P}}} \sum_{i \in \mathcal{W}(\sigma)} \mu_{ip(\sigma)} n(i, \sigma, a) + \sum_{p \in \mathcal{P}} \lambda_p \tag{7.41}$$

By $n(i, \sigma, a)$, we address the number of activities of group (i, σ) which are scheduled by decision a. Based on a decision, we obtain for the probability that an activity of group (i, σ) is completed

$$q^{\mathrm{CP}}(s, a, i, \sigma) = \frac{n(i, \sigma, a)\mu_{ip(\sigma)}}{\beta(s, a)} \tag{7.42}$$

and for the probability that a project of type $p \in \mathcal{P}$ arrives

$$q^{\mathrm{AP}}(s, a, p) = \frac{\lambda_p}{\beta(s, a)} \tag{7.43}$$

The evaluation equations (Poisson's equations) are then given by

$$h(s) = \frac{c(s) - g(\pi)}{\beta(s, \pi(s))} + \sum_{p \in \mathcal{P}} \left[q^{\mathrm{AP}}(s, \pi(s), p) \left(k^{\mathrm{A}}(s, p) + h(s^{\mathrm{AP}}(s, p)) \right) \right] + \sum_{\sigma \in \Sigma^{\mathrm{P}}} \sum_{i \in \mathcal{E}} q^{\mathrm{CP}}(s, \pi(s), i, \sigma) h(s^{\mathrm{CP}}(s, i, \sigma)) \quad \forall s \in \mathcal{S} \tag{7.44}$$

and the optimality equations (Bellman equations) by

$$h(s) = \min_{a \in \mathcal{A}(s)} \left\{ \frac{c(s) - g^*}{\beta(s, a)} + \sum_{p \in \mathcal{P}} \left[q^{\mathrm{AP}}(s, a, p) \left(k^{\mathrm{A}}(s, p) + h(s^{\mathrm{AP}}(s, p)) \right) \right] + \sum_{\sigma \in \Sigma^{\mathrm{P}}} \sum_{i \in \mathcal{E}} q^{\mathrm{CP}}(s, a, i, \sigma) h(s^{\mathrm{CP}}(s, i, \sigma)) \right\} \quad \forall s \in \mathcal{S} \tag{7.45}$$

7.1.2.3 Efficient Procedure for Determining Optimal Decisions

While the state space for the preemptive problem becomes smaller the decision sets $\mathcal{A}(s)$ may become larger as all resources idle and more scheduling alternatives exist. However, for the simplified CTMDP optimal scheduling decision can be efficiently determined without enumeration and evaluation of all decisions $a \in \mathcal{A}(s)$ as given by the following theorem.

Theorem 7.1.2. *An optimal decision $a^* \in \mathcal{A}(s)$ is given by the optimal solution of the following linear program (LP).*

$$\min \quad \sum_{\sigma \in \Sigma^P} \sum_{i \in \mathcal{W}(\sigma)} n(i, \sigma) Q(s, i, \sigma) \tag{7.46}$$

where

$$Q(s, i, \sigma) = \mu_{ip(\sigma)} \left(h(s^{CP}(s, i, \sigma)) - h(s) \right) \tag{7.47}$$

subject to

$$\sum_{\sigma \in \Sigma^P} \sum_{i \in \mathcal{W}(\sigma)} \mathbb{1} \left\{ r_{ip(\sigma)} = r \right\} n(i, \sigma) \leq c_r \quad \forall r \in \mathcal{R} \tag{7.48}$$

$$n(i, \sigma) \leq n(\sigma, s) \quad \forall \sigma \in \Sigma^P, i \in \mathcal{W}(\sigma) \tag{7.49}$$

$$n(i, \sigma) \in \mathbb{N}_0 \quad \forall \sigma \in \Sigma^P, i \in \mathcal{W}(\sigma) \tag{7.50}$$

The problem can be classified as a bounded multi-dimensional knapsack problem (cf. Martello and Toth [91] and Fréville [50]) where the values $Q(s, i, \sigma)$ can be

interpreted as the change of the long term expected total cost by the completion of activity i of a project in state σ. Thus, if $Q(s,i,\sigma) < 0$ activity $i \in \mathcal{W}(\sigma)$ from a project in state σ is scheduled if enough resources are available. Obviously, as the values $Q(s,i,\sigma)$ have the role of state dependent priorities when scheduling activities, activity groups (i,σ) can also be interpreted as *priority classes*. The first set of constraints (7.48) states that at most c_r activities may be scheduled on resource type r. The second set of constraints (7.49) states that the number of activities $n(i,\sigma)$ of group (i,σ) to be scheduled may not exceed the number of waiting activities of the respective group. As each activity of type (i,p) with $i \in \mathcal{V}_p$ occurs at most once in the set of waiting activities $\mathcal{W}(\sigma)$ of a project state $\sigma \in \Sigma^{\mathrm{P}}$, the number of waiting activities of group (i,σ) equals the number of projects in this project state $n(\sigma,s)$. The third set of constraints (7.50) establishes integrality of the solution. Note that for the theoretically possible case that no activity is selected as $Q(s,i,\sigma) > 0\ \forall \sigma \in \Sigma^{\mathrm{P}}, i \in \mathcal{W}(\sigma)$ a single activity of the group (i^*,σ^*) with the smallest $Q(s,i,\sigma) > 0$ must be selected ($n(i^*,\sigma^*) = 1$) such that total idleness is avoided.

Now, we give the proof of Theorem 7.1.2.

Proof. At first, we uniformize the CTMDP as outlined in Sect. 4.5 in order to replace the rates $\beta(s,a)$ by the uniformization constant c, which is independent of a decision. Thus, the optimality equations of the *uniformized* CTMDP are given by

$$\begin{aligned} h(s) = \min \Bigg\{ & \frac{c(s) - g^*}{c} + \sum_{\sigma \in \Sigma^{\mathrm{P}}} \sum_{i \in \mathcal{W}(\sigma)} \frac{n(i,\sigma)\mu_{ip(\sigma)}}{c} \left(h(s^{\mathrm{CP}}(s,i,\sigma)) \right) \\ & + \sum_{p \in \mathcal{P}} \frac{\lambda_p}{c} \left(k^{\mathrm{A}}(s,p) + h(s^{\mathrm{AP}}(s,p)) \right) \\ & + \left(1 - \sum_{\sigma \in \Sigma^{\mathrm{P}}} \sum_{i \in \mathcal{W}(\sigma)} \frac{n(i,\sigma)\mu_{ip(\sigma)}}{c} - \sum_{p \in \mathcal{P}} \frac{\lambda_p}{c} \right) h(s) \Bigg\} \quad \forall s \in \mathcal{S}^{\mathrm{P}} \end{aligned} \tag{7.51}$$

Rearranging and pulling constant terms out of the minimization gives

$$\begin{aligned} h(s) = \min \Bigg\{ & \sum_{\sigma \in \Sigma^{\mathrm{P}}} \sum_{i \in \mathcal{W}(\sigma)} \frac{n(i,\sigma)\mu_{ip(\sigma)}}{c} \left(h(s^{\mathrm{CP}}(s,i,\sigma)) - h(s) \right) \Bigg\} \\ + \frac{c(s) - g^*}{c} + h(s) + \sum_{p \in \mathcal{P}} \frac{\lambda_p}{c} & \left(k^{\mathrm{A}}(s,p) h(s^{\mathrm{AP}}(s,p)) - h(s) \right) \quad \forall s \in \mathcal{S}^{\mathrm{P}} \end{aligned} \tag{7.52}$$

Now, we denote the value of an activity being scheduled by

$$Q(s,i,\sigma) = \mu_{ip(\sigma)} \left(h(s^{\mathrm{CP}}(s,i,\sigma)) - h(s) \right) \tag{7.53}$$

such that we can write

$$h(s) = \min \left\{ \sum_{\sigma \in \Sigma^{\mathrm{P}}} \sum_{i \in \mathcal{W}(\sigma)} \frac{n(i,\sigma)}{c} Q(s,i,\sigma) \right\}$$
$$+ \frac{c(s) - g^*}{c} + h(s) + \sum_{p \in \mathcal{P}} \frac{\lambda_p}{c} \left(k^{\mathrm{A}}(s,p) h(s^{\mathrm{AP}}(s,p)) - h(s) \right) \quad \forall s \in \mathcal{S}^{\mathrm{P}} \tag{7.54}$$

Now, we see that the right-hand side is an *affine* mapping of the decision variables $n(i,\sigma)$ on the value function $h(s)$ of state s. This property is also referred to as *affine expectation property* (cf. Moallemi et al. [94]).

As the constant part of the function as well the uniformization constant are not relevant for the minimization, it is sufficient to minimize

$$\min \sum_{\sigma \in \Sigma^{\mathrm{P}}} \sum_{i \in \mathcal{W}(\sigma)} n(i,\sigma) Q(s,i,\sigma) \tag{7.55}$$

The minimization is subject to the constraints (7.48) and (7.49). □

The problem can now be solved in two ways.

(i) We can solve a Linear Program (LP) obtained when relaxing the integrality constraint (7.50) of the decision variables $n(i,\sigma)$. The optimal solution of the LP is integral as the right-hand sides of the constraints are integral while the matrix of coefficients for the decision variables is *totally unimodular*. This can be seen as follows. The matrix of the coefficients for the left-hand sides of the constraints has two entries (one for each set of constraints) in each column being 1 while the other entries are 0. Thus, total unimodularity follows (cf. Williams [135]).

(ii) We can decompose the problem into $|\mathcal{R}|$ bounded knapsack problems as each activity group (i,σ) is entirely processed by one resource type. The $|\mathcal{R}|$ knapsack problems are very easy to solve as all items (activities) have the same size (1). The solution procedure is given by Algorithm 3.

Before we state the solution procedure we define

$$\Sigma(s) = \{\sigma \in \Sigma | n(\sigma, s) > 0\} \tag{7.56}$$

the subset of project states $\sigma \in \Sigma$ with $n(\sigma,s) > 0$. We break ties between activity groups (i,σ) having the same $Q(s,i,\sigma)$ according to the following criteria.

1. Project type $p(\sigma)$.
2. Set of waiting activities $\mathcal{W}(\sigma)$.
3. Index i.

Algorithm 3 Computing the optimal decision for the preemptive case

Require: $s \in \mathcal{S}^{\mathrm{P}}$
1:
2: **for** $\sigma \in \Sigma(s)$ **do**
3: **for** $i \in \mathcal{W}(\sigma)$ **do**
4: $n(i,\sigma) \leftarrow 0$
5: **end for**
6: **end for**
7: **for** $r \in \mathcal{R}$ **do**
8: Compute set $\mathcal{C} = \{(i,\sigma) | \sigma \in \Sigma(s) \wedge i \in \mathcal{W}(\sigma) \wedge r_{ip(\sigma)} = r\}$
9: $f \leftarrow 0$
10: **while** $\mathcal{C} \neq \emptyset$ and $f < c_r$ **do**
11: Get $(i,\sigma) \in \mathcal{C}$ with smallest $Q(s,i,\sigma)$
12: **if** $c_r - f > n(\sigma,s)$ **then**
13: $n(i,\sigma) \leftarrow n(\sigma,s)$
14: **else**
15: $n(i,\sigma) \leftarrow c_r - f$
16: **end if**
17: $f \leftarrow f + n(i,\sigma)$
18: $\mathcal{C} \leftarrow \mathcal{C} \backslash (i,\sigma)$
19: **end while**
20: **end for**

The run time of the procedure is characterized as follows.

Theorem 7.1.3. *The run time of Algorithm 3 is in*

$$O\left(\sum_{r\in\mathcal{R}} |\mathcal{C}(r)| + \sum_{r\in\mathcal{R}} |\mathcal{C}(r)| \log |\mathcal{C}(r)|\right).$$

Proof. For an efficient implementation, we store for each system state only the tuples $(\sigma, n(\sigma))$ such that $\Sigma(s)$ is already given (cf. Sect. 4.7). The run time for the initialization is in $O(\sum_{r\in\mathcal{R}} |\mathcal{C}(r)|)$ as $\bigcup_{r\in\mathcal{R}} \mathcal{C}(r) = \{(i,\sigma) | \sigma \in \Sigma(s), i \in \mathcal{W}(\sigma)\}$. For each set $\mathcal{C}(r)$, we can use a sorted list which can be obtained in $O(|\mathcal{C}(r)| \log |\mathcal{C}(r)|)$. using for example *Quicksort* (cf. Sedgewick [117]). Then, getting (i,σ) as well as the remaining steps in the inner loop for each resource type $r \in \mathcal{R}$ is done in constant time. Thus, the total run time of the inner loop is in $O\left(\sum_{r\in\mathcal{R}} |\mathcal{C}(r)|\right)$, and the entire procedure has a run time of $O\left(\sum_{r\in\mathcal{R}} |\mathcal{C}(r)| + \sum_{r\in\mathcal{R}} |\mathcal{C}(r)| \log |\mathcal{C}(r)|\right)$. □

Finally, we remark that the procedure is similar to the parallel scheduling scheme as used by a resource-based priority policy (cf. Sect. 6.1) based on the state dependent priorities $Q(s,i,\sigma)$.

The advantage of not enumerating $A(s)$ can be seen in the following example.

Example 7.1.2. We consider a system with a single resource type with $c_1 = 5$ resources and two project types consisting of a single activity type. Hence, each project type $p = 1, 2$ can only have one project state $\sigma = (p, \{1\}, \emptyset)$ for any system state. Hence, in total we have 2 project states σ_1 and σ_2. Next, we consider system state $s \in \mathcal{S}^{\mathrm{P}}$ with $n(\sigma_1, s) = n(\sigma_2, s) = 10$.

When fully enumerating all feasible decisions in $A(s)$, we have to consider six alternative decisions $(n(1, \sigma_1), n(1, \sigma_2))$ such that $\mathcal{A}(s)$ is given by

$$\mathcal{A}(s) = \{(0, 5), (1, 4), (2, 3), (3, 2), (4, 1), (5, 0)\} \tag{7.57}$$

For the evaluation of each decision, we would have to consider four transitions(except for the decisions $(0, 5)$ and $(5, 0)$). One transition for the completion of a project of type 1, one transition for completion of a project of type 2 and two transitions for the arrival processes (we have one arrival process for each project type). In total we have to consider $4 \cdot 4 + 3 \cdot 2 = 22$ transitions.

Using Algorithm 3, we only consider two activity groups $(1, \sigma_1)$ and $(1, \sigma_2)$ for which we have to evaluate one transition each. Thus, in total we have to evaluate only two transitions instead.

7.1.2.4 State Space Cardinality

As the computational effort and memory requirements to determine an optimal policy π^* depends largely on the number of states it is useful to know $|\mathcal{S}^{\mathrm{P}}|$. The following theorem delivers firstly a sufficient criterion for a state s be in $\mathcal{S}^{\mathrm{P}}$, and secondly a formula for $|\mathcal{S}^{\mathrm{P}}|$.

Theorem 7.1.4. *(a) Any state s with $\sigma \in \Sigma^P \; \forall n(\sigma, s) > 0$ and $K(s) \leq K^{\max}$ is in $\mathcal{S}^P$.*

(b) $|\mathcal{S}^P| = \begin{pmatrix} K^{\max} + |\Sigma^P| \\ |\Sigma^P| \end{pmatrix}$

Proof. To prove part (a), it is sufficient to show by induction over the number of projects in the system ($K(s)$) that any state s with $K(s) \leq K^{\max}$ is accessible from s^0 under some policy.

Induction start: For $K(s) = 0$, the system must be in s^0 such that the case is trivial.

Induction step: We show that each state s with $K(s) \leq K^{\max}$ is accessible from a finite sequence of states s'' with $K(s'') = K(s) \leq K^{\max}$ that are again accessible from a state s' with $K(s') = K(s) - 1$ under some policy. By induction assumption, s' is accessible from s^0 under some policy π such that $s' \in \mathcal{S}^{\mathrm{P}}$. Thus, all s'' are accessible from s^0 such that $s'' \in \mathcal{S}^{\mathrm{P}}$ and s must be accessible from s^0. As a consequence the assertion that $s \in \mathcal{S}^{\mathrm{P}}$ follows.

Now we have to consider two cases.

1. If $n(\sigma_p^{\mathrm{I}}, s) > 0$ holds for at least one project type $p \in \mathcal{P}$, then s is accessible by a single transition subsequent to an arrival event from a state s' with $K(s') - 1$.
2. Otherwise, we consider any project state σ with $n(\sigma, s) > 0$ and consider the following scenario. A project has just arrived in state $s''' = s - \boldsymbol{e}(\sigma) + \boldsymbol{e}(\sigma^{\mathrm{I}}_{p(\sigma)})$. We assume that afterwards activities $i \notin \mathcal{U}(\sigma)$ of the project are always scheduled and subsequent transitions are related only to completions of those activities. Hence, the subsequent system states s'' are given by $s'' = s''' - \boldsymbol{e}(\sigma^{\mathrm{I}}_{p(\sigma)}) + \boldsymbol{e}(\sigma')$ where σ' is an intermediate state with $p(\sigma') = p(\sigma)$ and we have for the sets of unfinished activities $\mathcal{U}(\sigma) \subset \mathcal{U}(\sigma') \subset \mathcal{U}(\sigma^{\mathrm{I}}_{p(\sigma)})$. As only activities $i \notin \mathcal{U}(\sigma)$ are scheduled system state s must be entered sometime such that it is accessible from s''' via the system states s''. Obviously, such a scenario is in line with the assumptions made so far for scheduling policies such that such a policy must exist. As s''' is covered by case 1 with $K(s''') = K(s'') = K(s) \leq K^{\max}$ s''' is accessible from s^0 such that $s''' \in \mathcal{S}^{\mathrm{P}}$. Thus states s'' are accessible from s^0 such that $s'' \in \mathcal{S}^{\mathrm{P}}$. Finally, we conclude s is accessible from s^0 via states s', s''' and s'' such that $s \in \mathcal{S}^{\mathrm{P}}$.

The proof of part (b) is based on the result from the first part. As a first step, transform the semi-open system into an equivalent closed system having fixed number of $K^{\max}$ projects in the system. We introduce a project state $\sigma^{\mathrm{NIS}} = (-1, \emptyset, \emptyset)$ for the projects that are not in the system. An interpretation is as follows: On completion, projects turns into generic projects without specific type that are queued in front of the system. Then, they wait for admission to the system at a rate $\lambda = \sum_{p \in \mathcal{P}} \lambda_p$ where the time between two admissions is exponentially distributed. On admission, a generic project turns into an project of type $p \in \mathcal{P}$ with probability $\frac{\lambda_p}{\lambda}$ which can be accepted or rejected.

Next, we observe that the state space must contain all possibilities of distributing $K^{\max}$ projects among the project states in $\Sigma^{\mathrm{P}} \cup \{\sigma^{\mathrm{NIS}}\}$. From combinatorics, this corresponds to the well know problem of distributing n items ($K^{\max}$ projects) among k buckets ($|\Sigma^{\mathrm{P}}| + 1$ project states) where the number of possibilities is given by $\binom{n+k-1}{k-1}$ such that

$$|\mathcal{S}^{\mathrm{P}}| = \binom{n+k-1}{k-1} = \binom{K^{\max} + |\Sigma^{\mathrm{P}}|}{|\Sigma^{\mathrm{P}}|}$$

□

The precise cardinality of the state space $|\mathcal{S}^{\mathrm{P}}|$ cannot always be computed using a closed expression as $|\Sigma^{\mathrm{P}}|$ depends on the structure of the networks $\mathcal{G}_p$. However, the following corollary delivers an interval for $|\mathcal{S}^{\mathrm{P}}|$.

Table 7.1 Parameters of Example 1

Parameter	Value
$\lvert\mathcal{R}\rvert$	3
c_r	1 $\forall r \in \mathcal{R}$
$K^{\max}$	20
$\lvert\mathcal{P}\rvert$	1

Corollary 7.1.1.

$$\begin{pmatrix} K^{\max} + \sum\limits_{p\in\mathcal{P}} |\mathcal{V}_p| \\ \sum\limits_{p\in\mathcal{P}} |\mathcal{V}_p| \end{pmatrix} \leq |\mathcal{S}^P| \leq \begin{pmatrix} K^{\max} + \sum\limits_{p\in\mathcal{P}} (2^{|\mathcal{V}_p|} - 1) \\ \sum\limits_{p\in\mathcal{P}} (2^{|\mathcal{V}_p|} - 1) \end{pmatrix} \tag{7.58}$$

Proof. The lower bound is obtained from the observation that for a project of type $p \in \mathcal{P}$ we have at least $|\mathcal{V}_p|$ possible project states where in each project state one of the activities is waiting. This is the case for projects where activities have to be executed in a strict linear order (when $OS = 1$) due to precedence relations. The upper bound is obtained as follows. For a project of type $p \in \mathcal{P}$, we have at most $2^{|\mathcal{V}_p|} - 1$ possible sets of waiting activities without counting the absorbing state σ_p^{F} of a project where all activities have been completed. This is the case for projects without any precedence relations between the activities (when $OS = 0$) such that for project state $\sigma_p^{\mathrm{I}} = (p, \mathcal{V}_p, \emptyset)$ all activities are waiting. Thus, activities of a project can be scheduled and completed in any order where any intermediate project σ' with $\mathcal{W}(\sigma') = \mathcal{U}(\sigma')$ may occur under some policy. □

Unfortunately, the upper bound for $|\mathcal{S}^{\mathrm{P}}|$ can be very large as the following example shows.

Example 7.1.3. For a problem instance with a single project type that consists of 5 activity types. Setting $K^{\max} = 10$ we obtain $3{,}003 \leq |\mathcal{S}^{\mathrm{P}}| \leq 1{,}121{,}099{,}408$.

In Sect. 7.3.1, we show how to remedy this problem using the class of *project state ordering policies*.

7.1.3 Numerical Example

In order to test the models and to gain insights into important features of optimal policies, we discuss a numerical example with a single project type. System parameters are given in Table 7.1. The parameters and the network of the project type are given in Fig. 7.1. To obtain a realistic case, we have set the payoff to a high value such that rejections should be avoided as much as possible. Otherwise, if rejections are less costly the optimal policy may exhibit an undesired behavior when the number of projects approaches the bound $K^{\max}$. The policy increases the

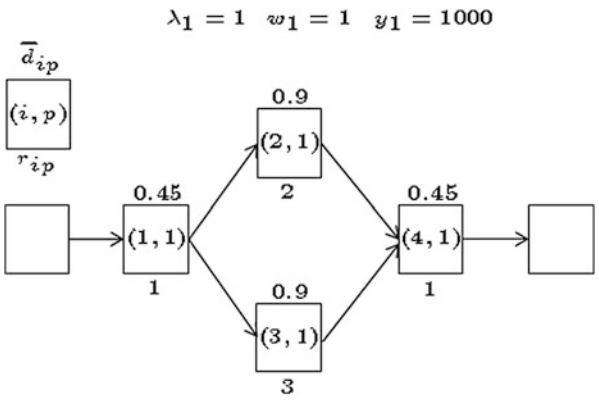

Fig. 7.1 Specifications of the example

Table 7.2 Objective function and state space cardinalities

	Non-preemptive	Preemptive
g^*	52.454	50.534
State space cardinality	683,209	53,130
Number of project states	12	5

rejection rate in order to reduce the flow times of the projects currently in the system. We will return to the effect of rejection costs in Sect. 7.1.3.4.

7.1.3.1 General Observations

Table 7.2 shows the value for the objective function as well as the state space cardinalities for the non-preemptive as well as the preemptive problem. At first, we observe that the optimal average cost is lower for the preemptive problem which can be easily explained by the fact that the preemptive problem is a relaxation of the non-preemptive problem. Thus, g^* of the preemptive problem is a lower bound for g^* of the non-preemptive problem.

Next, we observe that for the non-preemptive problem we obtain a state space that is more than 12 times larger than the state space for the preemptive problem. This can be explained by the fact for the non-preemptive problem activities in process that cannot be preempted need to be taken into account. Hence, we have 12 project states for the non-preemptive problem while for the preemptive problem there are only 4 project states that may occur in the pre-decision states. Table 7.6 shows the set Σ of all project states with the sets $\mathcal{W}(\sigma)$ and $\mathcal{E}(\sigma)$. In the rightmost column, we find project states marked using $\bullet$ to indicate that they belong to the set Σ^{P} of project states which may occur in the pre-decision states in case of the preemptive problem. For the non-preemptive problem, all project states $\sigma \in \Sigma$ may also occur in the pre-decision states. Recall that all project states (Table 7.3) not being in Σ^{P} may occur in the post-decision states.

Table 7.3 Project states

Σ	$\mathcal{W}(\sigma)$	$\mathcal{E}(\sigma)$	in Σ^P
σ_1	{1}	Ø	•
σ_2	{2}	Ø	•
σ_3	{3}	Ø	•
σ_4	{4}	Ø	•
σ_5	{2,3}	Ø	•
σ_6	Ø	{1}	
σ_7	Ø	{2}	
σ_8	Ø	{3}	
σ_9	Ø	{4}	
σ_{10}	{3}	{2}	
σ_{11}	{2}	{3}	
σ_{12}	Ø	{2, 3}	

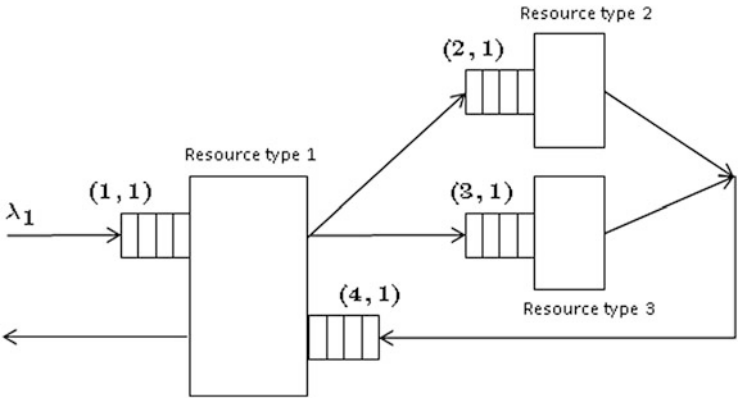

Fig. 7.2 Flow of the projects through the system

Before going into detail, we note that scheduling decisions depend on how resource types are involved in the flow of projects through the system. Figure 7.2 shows the system from the perspective of the resource types. Hence, it is useful to discuss the scheduling decisions for each resource type separately (although, in the CTMDP, a decision refers to activities for multiple resource types at a time). We observe that resource type 1 processes activity types $(1, 1)$ and $(4, 1)$. From Table 7.3, we know that activity 1 of a project (that is of activity type $(1, 1)$) can only be in $\mathcal{W}(\sigma_1)$ and activity 4 (of type $(4, 1)$) only in $\mathcal{W}(\sigma_4)$. Thus, decisions for resource type 0 refer either to scheduling of activity 1 of a project in state σ_1 or to activity 4 of a project in state σ_4.

Resource types 2 and 3 process only one activity type respectively. However, the activities of both activity types may occur in different project states. For the preemptive problem activities of activity type $(2, 1)$ may occur in σ_2 and σ_5 and for the non-preemptive problem additionally in σ_{11}. As a consequence, a decision refers to scheduling activity 2 either from a project in state σ_2 or state σ_5 (or state σ_{11}).

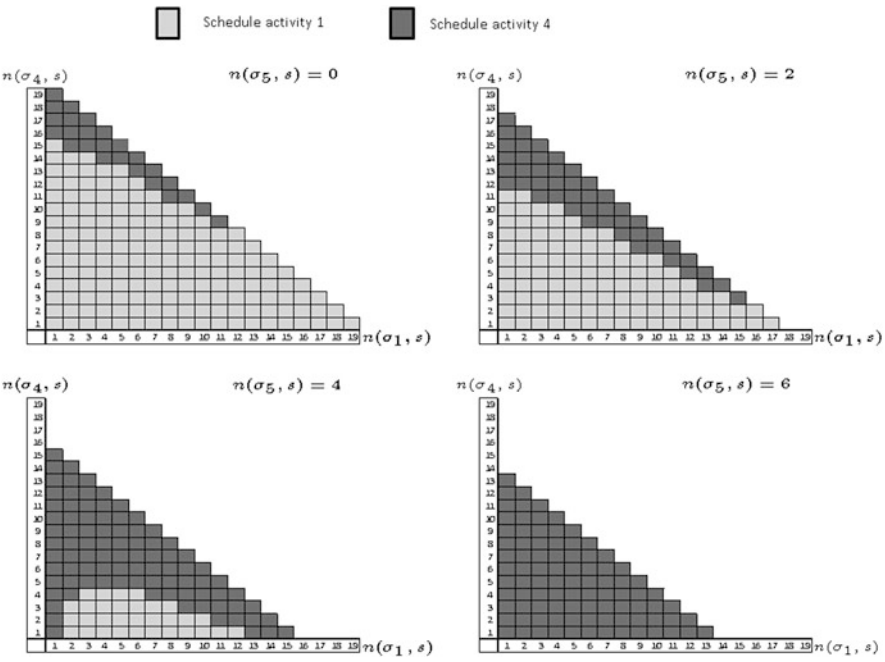

Fig. 7.3 Scheduling decisions for the preemptive problem at resource type 1

Due to its simpler structure, we first examine the optimal scheduling policy for the preemptive problem before considering the policy for the non-preemptive problem.

7.1.3.2 Optimal Policy for the Preemptive Scheduling Problem

To gain insights into the structure of the scheduling decisions for resource type 1, we consider the state space only at some specific regions of the state space. Figure 7.3 shows the optimal scheduling decisions where $n(\sigma_1, s) > 0$ and $n(\sigma_4, s) > 0$ for different levels of $n(\sigma_5, s)$. For all other $\sigma \in \Sigma^{\mathrm{P}}$, we have $n(\sigma, s) = 0$. We start the analysis by considering the case where $n(\sigma_5, s) = 0$. Then, as long as $n(\sigma_1, s)$ and $n(\sigma_4, s)$ are below a certain threshold within the light grey region only activity 1 of some project in σ_1 is scheduled. Beyond the threshold in the dark grey region, activity 4 of some project in σ_4 is scheduled. To understand the policy, we note that, if resource types 2 and 3 have no activities waiting ($n(\sigma_5, s) = 0$), we have idleness of both resource types after making a decision. Then, for small $n(\sigma_4, s)$ activity 1 from projects in state σ_1 is preferred as, on completion of activity 1, resource types 2 and 3 can be used for processing activities 2 and 3. Otherwise, unused idle times may be the consequence that lead to higher delays of projects in state σ_1 having

Table 7.4 Optimal scheduling decisions of resource type 1 and 2 for the preemptive problem

Case	Condition	Decision
Resource type 1		
1	$n(s,\sigma_2) > 0$	Schedule activity 2 from a project in state σ_2
2	$n(s,\sigma_2) = 0 \wedge n(s,\sigma_5) > 0$	Schedule activity 2 from a project in state σ_5
Resource type 2		
1	$n(s,\sigma_3) > 0$	Schedule activity 3 from a project in state σ_3
2	$n(s,\sigma_3) = 0 \wedge n(s,\sigma_5) > 0$	Schedule activity 3 from a project in state σ_5

activity 1 waiting (as there is less time available for completing activities of types 2 and 3).

If we increase $n(\sigma_5, s)$, such that there are more projects having activities 2 and 3 waiting, the risk that resource types 2 and 3 idle is reduced. Obviously, this may only happen if all activities waiting for resources 2 or 3 are completed before completion of activity 1 of a project in state σ_1. Thus, the dark grey region where activity 4 is preferred for scheduling becomes larger. This behavior can be interpreted as building up a *safety stock* (cf. Meyn [93]) for protecting resource types 2 and 3 from becoming idle. The preference for activity 4 for increasing $n(\sigma_4, s)$ can be explained by a decreasing benefit of scheduling activity 1. As for projects having activity 1 waiting activity 4 is still unfinished such that it is likely that they will be delayed later due to a large $n(\sigma_4, s)$.

Next, we consider the scheduling decisions related to resource types 2 and 3. Table 7.4 shows that for resource type 2 and 3 scheduling decisions are made according to simple rules irrespectively from the states of other projects. An explanation is given by the fact that activity 2 (3) from a project in state σ_2 (σ_3) is less critical in the sense that a delay of the activity is less likely to lead to a delay of the project than activity 2 (3) from a project in state σ_5. A project in state σ_5 may also be delayed by activity 3 (2).

7.1.3.3 Optimal Policy for the Non-preemptive Scheduling Problem

Figure 7.4 shows the optimal scheduling decisions for resource type 1. Note that for the non-preemptive problem, states where both resource types 2 and 3 idle, while there multiple projects are waiting in σ_5 are infeasible (except for states with $n(\sigma_5, s) = 1$ that are entered on completion of activity 1 of some project) as we require policies to be non-idling. Hence, except for the first case where both resource types idle, we consider states where one project always has both activities in process (σ_{12}) on resource types 2 and 3.

Generally, we observe that the structure of the optimal policy is similar to the structure of the optimal policy for the preemptive problem concerning the decisions for resource type 1. This observation suggests using the solution for the preemptive problem as an approximation which is easier to solve for the non-preemptive problem. We will return to this idea in Sect. 7.4.2.

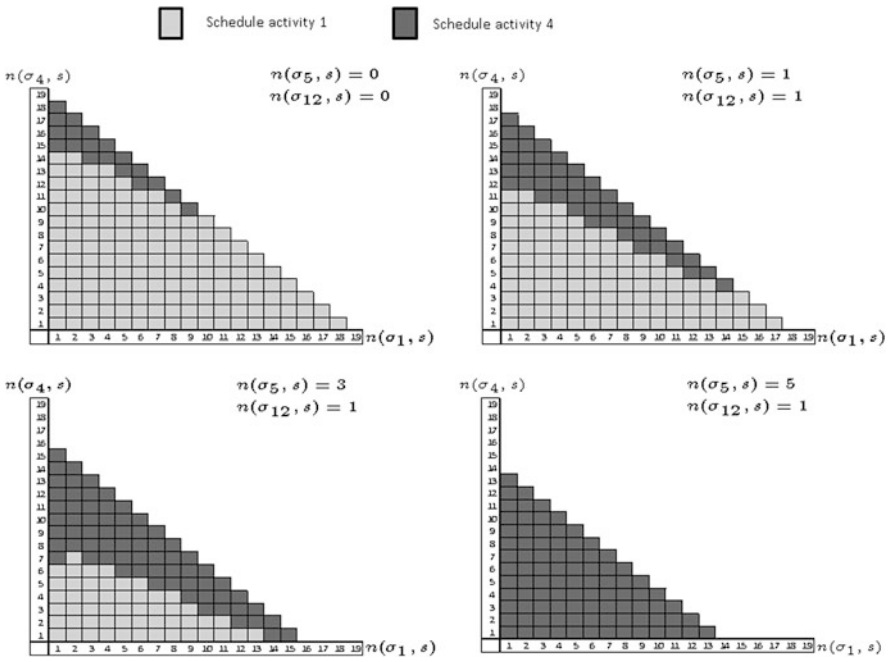

Fig. 7.4 Scheduling decisions for the non-preemptive problem at resource type 1

Table 7.5 Optimal scheduling decisions of resource type 1 and 2 for the non-preemptive problem

Case	Condition	Decision
Resource type 1		
1	$n(s, \sigma_2) > 0$	Schedule activity 2 from a project in state σ_2
2	$n(s, \sigma_2) = 0 \wedge n(s, \sigma_{11}) > 0$	Schedule activity 2 from a project in state σ_{11}
3	$n(s, \sigma_2) = 0 \wedge n(s, \sigma_{11}) = 0 \wedge n(s, \sigma_5) > 0$	Schedule activity 2 from a project in state σ_5
Resource type 2		
1	$n(s, \sigma_3) > 0$	Schedule activity 2 from a project in state σ_3
2	$n(s, \sigma_3) = 0 \wedge n(s, \sigma_{10}) > 0$	Schedule activity 2 from a project in state σ_{10}
3	$n(s, \sigma_3) = 0 \wedge n(s, \sigma_{10}) = 0 \wedge n(s, \sigma_5) > 0$	Schedule activity 2 from a project in state σ_5

Note that, for the case where resource types 2 and 3 idle without waiting activities, we have a maximum of 19 projects in the system. This can be explained by the fact that for the non-preemptive problem idleness of all three resource types can only occur on completion of activity 4 processed on resource type 1 such that the project leaves afterwards. If it were activity 1 of some project we would have one project being in state σ_5. The scheduling decisions related to resource types 1 and 2 become more involved than for the preemptive problem as projects having activity 2 (3) waiting for resource type 1 (2) may now be in state σ_2, σ_5 and σ_{11} (σ_3, σ_5 and σ_{10}). We will now present the optimal rules shown in Table 7.5 for this case.

Table 7.6 Performance figures for different policies

y_1	0	1,000	1,000	1,000	1,000	1,000
Policy	$\pi^{*,0}$	$\pi^{*,1,000}$	$\pi^{*,0}$	RAN	BD-GC-U	BD-GC-D
g	10.129	50.387	75.795	68.914	64.469	64.617
λ_1^{eff}	0.935	0.962	0.935	0.943	0.948	0.947
Avg. holding cost	10.129	11.816	10.129	11.856	12.109	12.103
Avg. rejection cost	0.000	38.571	65.667	57.0581	52.361	52.507
Avg. flowtime	10.837	12.286	10.837	12.569	12.773	12.776

Basically, the rule for the preemptive scheduling problem has been extended by the consideration of project state σ_{11} (σ_{10}). Additionally, we have the case that activity 2 (3) from a project in state σ_2 (σ_3) is preferred over activity 2 (3) from a project in state σ_{11} (σ_{10}) for a similar argument as the one used above for the preemptive problem. Furthermore, activity 2 (3) from a project in state σ_{11} (σ_{10}) is preferred of activity 2 (3) from a project in state σ_5 as it is more critical. The parallel activity 3 (2) being in process for a project in state σ_{11} is completed earlier than activity 3 (2) from a project in state σ_5 as it has not been scheduled yet.

Thus, we have a structural property of the optimal policies that activities from projects having more activities completed or more activities already scheduled should be preferred. This property can be observed for many instances having parallel networks. This idea has led us to the class of *project state ordering* policies which will be discussed in Sect. 7.3.

7.1.3.4 Effect of Rejection Cost

To conclude the investigation, we briefly address the effect of rejection costs. For simplicity, we only consider the preemptive case as similar observations could be obtained for the non-preemptive case.

In order to investigate the effect of rejection costs, we have computed the optimal policy $\pi^{*,0}$ for the case where $y_1 = 0$. In order to show the benefit of taking into account rejection costs, we have applied policy $\pi^{*,0}$ to the case where $y_1 = 1{,}000$. Table 7.6 shows the results for $\pi^{*,0}$ applied to the case with $y_1 = 0$ as well as $y_1 = 1{,}000$. As a benchmark, we show for the case where $y_1 = 1{,}000$ the performance of the optimal policy $pi^{*,1,000}$ as well as the performance of three priority policies, namely RAN, BD-GC-U and BD-GC-D. In the first row, it is indicated to which of the two cases ($y_1 = 0$ or $y_1 = 1{,}000$) the results in a column refer. Ties between activities with same priorities are broken randomly. BD-GC-U and BD-GC-D (with $\kappa = 1$) are two priority policies based on the *bottleneck dynamics* approach (cf. Lawrence and Morton [85] or Morton and Pentico [96]) that have performed well in the simulation study presented in Chap. 6. For details, we refer to Sect. 6.1.

To explain the performance, we show the parts of the average cost (average holding cost and average rejection cost) separately and the *effective arrival rate*

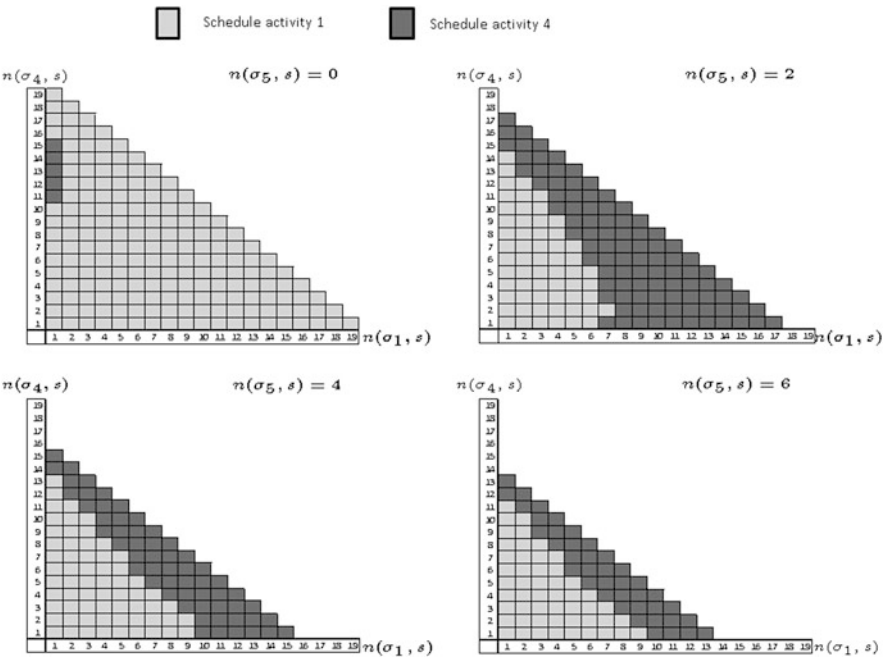

Fig. 7.5 Scheduling decisions for the preemptive problem at resource type 1

λ_1^{eff} which is the arrival rate obtained when rejections due to $K(s) = K^{\max}$ are taken into account. Since $w_1 = 1$ the average holding correspond to the average number of projects in the system. Thus, the average flow time can be easily obtained from *Little's law* (cf. Gross and Harris [54]) by dividing the average holding cost by the effective arrival rate λ_1^{eff}. We make a number of interesting observations. Firstly, the average holding cost as well as average flow times are lower when $y_1 = 0$. However, the average cost is higher if policy $\pi^{*,0}$ is applied to the case with $y_1 = 1{,}000$ than the average (overall) cost obtained from the optimal policy $\pi^{*,1{,}000}$. This can be explained by regarding λ_1^{eff}. As λ_1^{eff} is lower for $\pi^{*,0}$ than for $\pi^{*,1{,}000}$ we conclude that $\pi^{*,0}$ obviously exploits rejections if $K(s) = K^{\max}$ in order to reduce the average number and flow times of projects in the system. However, if $\pi^{*,0}$ is applied to the case with $y_1 = 1{,}000$ total cost become much higher.

For the priority policies, we observe that although RAN achieves a lower average holding cost and flow time than BD-GC-U as well as BD-GC-D the latter priority policies achieve a lower average (overall) cost g (Table 7.6).

Thus, we conclude that for semi-open system it is not sufficient to optimize average flow times or holding cost. Instead, total cost including rejection cost should be considered. It is even possible that for optimal policies larger average flow times are obtained while rejection cost may be lower.

Next, in order to understand how $\pi^{*,0}$ exploits the limit $K^{\max}$ to reduce λ_1^{eff}, we analyze important features of its structure. Figure 7.5 shows a part of the optimal

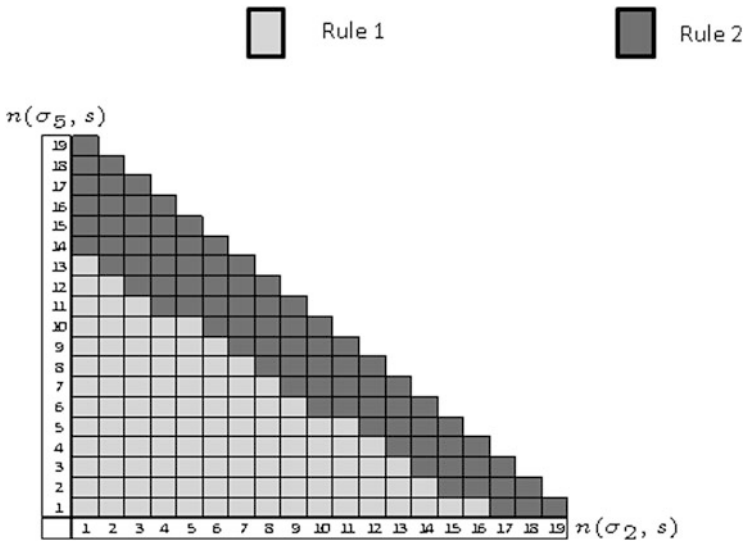

Fig. 7.6 Usage of rules for scheduling at resource type 2 if $y_1 = 0$

scheduling decisions for resource type 1 where $n(\sigma_1, s) > 0$ and $n(\sigma_4, s) > 0$ for different levels of $n(\sigma_5, s)$. For all other $\sigma \in \Sigma^{\mathrm{P}}$ we have $n(\sigma, s) = 0$. Comparing the structure of the decisions for $\pi^{*,0}$ with the structure of the decisions for $\pi^{*,1,000}$ as shown in Fig. 7.3 we observe that projects having activity 1 waiting are preferred more often as the light grey region becomes larger. This can be explained by the fact that preferring activities from less advanced projects in terms of less completed (or unfinished) activities leads, in short term, to more projects in the system as less projects are completed. Thus, a larger number of projects in the system make rejections more likely. However, in long term, when there are many advanced projects in the system (with only few unfinished activities), they do not have to compete with projects having arrived later such that their flow times become shorter.

Next, we consider the optimal decisions for resource type 2. While for the case with $y_1 = 1{,}000$ optimal decisions followed a very simple structure, the structure becomes more involved for the case with $y_1 = 0$. Instead, we now have two rules Rule 1 and Rule 2 for determining the optimal decision as shown in Table 7.7. Obviously, Rule 2 is the opposite of Rule 1 and prefers activity 2 from projects in state σ_5 instead of projects in σ_2. When to use one of both rules depends on the system state. Figure 7.6 indicates when to use one of both rules for system states where $n(\sigma_2, s) > 0, (\sigma_5, s) > 0$ and $n(\sigma_1, s) = 0, (\sigma_3, s) = 0, (\sigma_4, s) = 0$.

We observe that for larger numbers of projects in the system (dark grey region) Rule 2 is preferred and Rule 1 otherwise. This can be explained again by the fact that the policy tries to keep the number of projects in the system at a high level in order to make rejections more likely. Thus, less advanced projects in terms of

Table 7.7 Optimal scheduling decisions of resource type 2 for the case with $y_1 = 0$

Case	Condition	Decision
Rule 1		
1	$n(s, \sigma_2) > 0$	Schedule activity 2 from a project in state σ_2
2	$n(s, \sigma_2) = 0 \wedge n(s, \sigma_5) > 0$	Schedule activity 2 from a project in state σ_5
Rule 2		
1	$n(s, \sigma_5) > 0$	Schedule activity 2 from a project in state σ_5
2	$n(s, \sigma_5) = 0 \wedge n(s, \sigma_2) > 0$	Schedule activity 2 from a project in state σ_2

more unfinished activities are preferred. For other system states the structure of the optimal policy w.r.t. usage of the two scheduling rules is similar.

As activity type 3 has the same expected duration and successors as activity type 2 optimal decisions for resource type 3 have the same structure.

Note that the observed behavior of the optimal policy without consideration of rejection cost is typically undesired as considerable losses of revenues are the consequence. Furthermore, acceptance and rejection of new orders arriving at the system is subject to order acceptance on the tactical level (cf. Chap. 8) while scheduling on the operational level should not lead to any losses.

An interesting question remains – How do the structure of optimal policies looks like for open systems? We have found in a number of experiments where we have increased $K^{\max}$ that for a given set of states the policies become more similar to the policies with larger rejection costs. This can be explained by the fact that for large $K^{\max}$ the incentive becomes less to hold projects longer in the system for preventing future arrivals. For open systems with unbounded $K^{\max}$ project arrivals are completely independent from the scheduling policies.

7.2 Optimal Policy for the Single Resource Case Without Preemptions

In this section, we consider a special case with the following assumptions.

- System consists of a single resource ($|\mathcal{R}| = 1$ and $c_r = 1$).
- Activities of type (i, p) have generally distributed duration with mean $\overline{d}_{ip}$.
- Activities in process may not be preempted.
- The maximum number of projects in the system $K^{\max}$ is unbounded.

Then, the optimal scheduling policy is given by the following theorem.

Theorem 7.2.1. *The optimal scheduling policy minimizing the average weighted flow time is a priority index policy where an activity is selected according to*

$$\max_{(i,j)\in\mathcal{W}_r(t)} \frac{w_{\boldsymbol{p}_j}}{\sum_{m\in\mathcal{U}_j(t)} \overline{d}_{m\boldsymbol{p}_j}} \tag{7.59}$$

As a tie breaker among the activities of a project and between the projects $j \in \mathcal{J}(t)$ in case of equal priority any policy can be used.

Recall that a scheduling policy is a *priority index policy* (cf. Nino-Mora [100]) where projects competing for the resource are assigned an index which is used for prioritizing the projects. The index of a project depends only on its state at the time of the decision and is independent from the states of other projects.

Before giving the proof, we briefly introduce a problem known as *Klimov's problem* for which Klimov [72] proved that the optimal policy is a priority index policy and provided a recursion for obtaining the priorities.

The problem can be sketched as follows. We consider a set Ω of job classes which are processed by a single resource. Jobs of class $\omega \in \Omega$ arrive according to a Poisson process with rate λ_ω. For the cases where the resource is busy, the jobs of a class ω wait in their own queue with a length $n(\omega, t)$. The service time is assumed to be generally distributed with cumulative distribution function $B(\omega, t)$. On completion of a job of class ω, it turns into a job of class $\omega' \in \Omega$ with probability $p_{\omega\omega'}$ and leaves the system with probability $1 - \sum_{\omega' \in \Omega} p_{\omega\omega'}$.

As long as a job of class ω is in the system a holding cost w_ω is incurred per unit of time. As objective function Klimov considers the minimization of the average cost per unit of time.

Then, a sequence of sets (of job classes) $\Omega_1^*, \ldots, \Omega_o^*$ is defined by

$$\Omega_l^* = \left\{ \omega \in \Omega_l \,\middle|\, \frac{w_\omega}{\gamma(\omega, \Omega_l)} = \min_{\omega' \in \Omega_l} \frac{w_{\omega'}}{\gamma(\omega', \Omega_l)} \right\} \tag{7.60}$$

where

$$\Omega_1 = \Omega;\ \Omega_{l+1} = \Omega \setminus \bigcup_{m=1}^{l} \Omega_m^* \tag{7.61}$$

$\gamma(\omega, \Omega_i)$ denotes the expected remaining processing time (no waiting times) until a job of class ω leaves the set of job classes Ω_i. Based on the sequence of sets an optimal policy for scheduling, jobs of the different classes on the resource is given by the following theorem.

Theorem 7.2.2. *The optimal policy is to prefer job classes in the set Ω_{l+1}^* over jobs in the class Ω_l^*.*

The proof can be found in Klimov [72].

Now, we give proof of Theorem 7.2.1.

Proof. The proof is composed of two steps. In a first step we show that the problem can be interpreted a special case of Klimov's–problem and apply in a second step Theorem 7.2.2 to obtain the policy.

As at decision times no project is in process we do not have to account for the time any activity already has been in process. Thus, it is sufficient to consider

for project $j \in \mathcal{J}(t)$ the current state $\sigma_j(t) = \left(\boldsymbol{p}_j, \mathcal{W}_j(t), \mathcal{E}_j(t)\right)$ as defined in Sect. 7.1.1.1. After scheduling an activity (i, j) with $i \in \mathcal{W}(\sigma(j,t))$ at time t it is processed without preemption until completion as no preemptions are allowed. On completion of activity (i, j) the subsequent project state $\sigma^{\mathrm{CW}}(i, \sigma)$ given by (7.38). All successors of activity (i, j) for which all predecessors have been completed are added, if available, to the set of waiting activities. The states of the other projects do not change as only one activity can be executed at a time. We observe that it is not necessary to distinguish between two projects $j, j' \in \mathcal{J}(t)$ with $\sigma(j,t) = \sigma(j',t)$ such that we only need to account for $n(\sigma, t)$ being the number of projects in state σ at time t.

Next, we interpret each project state $\sigma \in \Sigma$ as a job type where for the choice of an activity $i \in \mathcal{W}(\sigma)$ a stationary (possibly random) policy $z : \Sigma \longrightarrow \mathbb{N}$ is given that delivers based on the current project state $\sigma \in \Sigma$ the activity from $\mathcal{W}(\sigma)$ to be scheduled. Thus, a project switches from state σ to $\sigma^{\mathrm{C}}(i, \sigma)$ with a probability $p\left(\sigma, \sigma^{\mathrm{C}}(i, \sigma)\right) = p\left(z(\sigma) = i \in \mathcal{W}(\sigma)\right)$. Obviously, as each project must be completed and no abortions are allowed we must have $1 - \sum_{\sigma' \in \Sigma} p(\sigma, \sigma') = 0 \ \forall \sigma \neq \sigma^{\mathrm{F}}_{p(\sigma)}$ and $1 - \sum_{\sigma' \in \Sigma} p(\sigma, \sigma') = 1 \ \forall \sigma = \sigma^{\mathrm{F}}_{p(\sigma)}$ as the project leaves the system on completion of its last activity. Thus, the set of project states Σ corresponds to the set Ω of job types and we can apply Theorem 7.2.2 to obtain the optimal policy.

At first Σ can be divided to subsets $\Sigma_1, \ldots, \Sigma_o$ of project types according to (7.61). Starting with Σ_i we obtain

$$\Sigma_{i+1} = \Sigma_i \backslash \Sigma_i^* \tag{7.62}$$

where

$$\Sigma_i^* = \left\{ \sigma \in \Sigma_i \,\middle|\, \frac{w_{p(\sigma)}}{\gamma(\sigma, \Sigma_i)} = \min_{\sigma' \in \Sigma_i} \frac{w_{p(\sigma')}}{\gamma(\sigma', \Sigma_i)} \right\} \tag{7.63}$$

where $\gamma(\sigma, \Sigma_i)$ is the expected remaining processing time of a project in state σ until leaves the set Σ_i of project states. To obtain the indices, we need $w_{p(\sigma)}$ and $\gamma(\sigma, \Sigma_i)$ for all $\sigma \in \Sigma_i$.

While $w_{p(\sigma)}$ is already given, $\gamma(\sigma, \Sigma_i)$ needs to be computed. Fortunately, it can be easily obtained exploiting a relationship between the sets $\Sigma_1, \ldots, \Sigma_i$. Without loss of generality we consider the project states $\sigma \in \Sigma_i \backslash \Sigma_i^*$. Then, concerning the project states $\sigma' \in \Sigma_i^*$ two cases can be distinguished.

1. $p(\sigma) = p(\sigma')$: By definition of Σ_i^*, we must have $\frac{w_{p(\sigma)}}{\gamma(\sigma, \Sigma_i)} > \frac{w_{p(\sigma')}}{\gamma(\sigma', \Sigma_i)}$. As $w_{p(\sigma)} = w_{p(\sigma')}$ we must have $\gamma(\sigma, \Sigma_i) < \gamma(\sigma', \Sigma_i)$. As each activity of a project can only processed once due to the acyclic project network we must have $\gamma(\sigma, \Sigma_i) > \gamma(\sigma^{\mathrm{C}}(z(\sigma), \sigma), \Sigma_i)$. Thus, we conclude that σ' cannot be accessed from σ by completions of unfinished activities in $\mathcal{U}(\sigma)$.
2. $p(\sigma) \neq p(\sigma')$: σ' cannot be accessed from σ as a project cannot change its type.

Thus, all project states that are accessible from σ must be in Σ_i such that $\gamma(\sigma, \Sigma_{i+1}) = \gamma(\sigma, \Sigma_i)$. By induction it follows that

$$\gamma(\sigma, \Sigma_{i+1}) = \gamma(\sigma, \Sigma) = \sum_{k \in \mathcal{U}(\sigma)} \overline{d}_{kp(\sigma)}$$

such that it is sufficient to compute the expected remaining processing times only once.

As obviously $\gamma(\sigma, \Sigma)$ only depend on the project state σ any policy z may be used for the selection of the activity to be scheduled from $\mathcal{W}(\sigma)$. Furthermore, ties may be broken between two project states $\sigma, \sigma' \in \Sigma$ with $\frac{w_{p(\sigma)}}{\gamma(\sigma,\Sigma)} = \frac{w_{p(\sigma')}}{\gamma(\sigma',\Sigma)}$ using any rule. This completes the proof. □

We observe that the optimal policy boils down to the $c\mu$–policy (cf. WSPT–policy in Sect. 6.1) for the special case where project types consist of a single activity type.

7.3 Project State Ordering Policies

We observe that especially for networks having only few precedence relations between the activity types the state space may become forbiddingly large even for small networks and a limited $K^{\max}$. This observation is in line with the results by other authors who have considered stochastic scheduling of projects using CTMDPs (cf. Sobel et al. [120] or Creemers et al. [31]). An option to reduce the state space is restricting the search for an optimal policy to a class of policies where in each state $s \in \mathcal{S}$ (or $s \in \mathcal{S}^{\mathrm{P}}$ for the preemptive problem) only a subset of the alternative decisions $a \in \mathcal{A}(s)$ is considered. Thus, parts of the state space $\mathcal{S}$ ($s \in \mathcal{S}^{\mathrm{P}}$) may no longer be accessible from s^0 for all policies of a given class.

In this section, we restrict our considerations to systems with $c_r = 1 \ \forall r \in \mathcal{R}$. For such systems, we propose the class of *project state ordering policies* (POPs) that may be preemptive or non-preemptive. Due to their simpler structure, we consider first preemptive POPs in Sect. 7.3.1 and extend the analysis to non-preemptive POPs in Sect. 7.3.2.

7.3.1 Preemptive Project State Ordering Policies

At first, we give in Sect. 7.3.1.1 the definitions and present some general properties of the state space. Afterwards, we establish in Sect. 7.3.1.2 an interesting relation to a queueing network representation of the state space where each activity type (i, p) has its own queue. The relation allows us to quantify more precisely the reduction of $|\mathcal{S}^{\mathrm{P}}|$ using POPs.

7.3.1.1 Definitions and General Structural Results

At first, we define a relation between project states σ that is fundamental for the definition of POPs

Definition 7.3.1. We have two project states σ_1, σ_2. Then, project state σ_1 is **more advanced** than σ_2 if $p(\sigma_1) = p(\sigma_2)$ and $\mathcal{U}(\sigma_1) \subset \mathcal{U}(\sigma_2)$.

The term $\mathcal{U}(\sigma_1) \subset \mathcal{U}(\sigma_2)$ states that in order to have a project j_1 that is in a more advanced state σ_1 than another project j_2 in state σ_2 there must be for each unfinished activity (i, j_1) with $i \in \mathcal{U}(\sigma_1)$ an unfinished activity (i, j_2) with $i \in \mathcal{U}(\sigma_2)$. Furthermore there must be some additional unfinished activities (i', j_2) with $i' \notin \mathcal{U}(\sigma_1)$ and $i \in \mathcal{U}(\sigma_2)$. We shortly write $\sigma_1 \succ_a \sigma_2$ in order to say that project state σ_1 is *more advanced* than project state σ_2. Now, we define the class of *project state ordering* policies (POPs).

Definition 7.3.2. Let $\sigma_1, \sigma_2 \in \Sigma(p, s)$ be two projects states with $\sigma_1 \succ_a \sigma_2$. Then, a preemptive *project state ordering* policy π^{PO} always prefers activity $i \in \mathcal{W}(\sigma_1)$ from a project in state σ_1 over activity $i \in \mathcal{W}(\sigma_2)$ from a project σ_2 $\forall i \in \mathcal{W}(\sigma_1) \cap \mathcal{W}(\sigma_2)$.

By $\Pi^{\text{PPO}} \subset \Pi$ we denote the class of preemptive *project state ordering* policies (POPs) where $\mathcal{S}^{\text{PPO}}$ is the set of states (state space) that communicate with s^0 under at least one preemptive POP plus the empty system state s^0.

The class of policies formalizes the intuition that waiting activities of a project being in a more advances state tend to be more critical than those of a project in a less advanced state. Criticality refers to the probability to lie on the critical path (cf. Elmaghraby [46]). To see this, let us consider the following example.

Example 7.3.1. We consider two projects j_1 and j_2 where j_1 is in state σ_1 and j_2 in state σ_2 with $\sigma_1 \succ_a \sigma_2$. Furthermore, we have activities (i, j_1) and (i, j_2) with $i \in \mathcal{W}(\sigma_1) \cap \mathcal{W}(\sigma_2)$ that are waiting. Then, activities (i', j_2) with $i' \in \mathcal{U}(\sigma_2) \backslash \mathcal{U}(\sigma_1)$ may not be successors of activity (i, j_2). Otherwise, they would be in $\mathcal{U}(\sigma_1)$ as well. Thus, activity (i, j_2) is less critical than activity (i, j_1) as activities (i', j_2) may be completed later than activity (i, j_2) such that scheduling (i, j_2) tends to have a smaller impact on project flow time that scheduling of project (i, j_1). Let us assume that we have for the two projects j_1 and j_2 $\mathcal{W}(\sigma_1) = \mathcal{U}(\sigma_1) = \{1\}$ and $\mathcal{W}(\sigma_2) = \mathcal{U}(\sigma_2) = \{1, 2\}$ of type $p \in \mathcal{P}$. Activity type $(1, p)$ is to be processed on a different resource type than activity type $(2, p)$. Then, Fig. 7.7 shows two scenarios. Scenario 1 refers to scheduling of activities $(1, j_1)$ and $(2, j_2)$ where Scenario 2 refers to scheduling of activities $(1, j_2)$ and $(2, j_2)$ where there are given realizations d_{1j_1},d_{1j_2} and d_{2j_2} for the activity durations. For simplicity, the presentation is only for the preemptive-resume case. Note that as only activity $(2, j_2)$ needs resource r_2 it is immediately rescheduled on completion of activity $(1, j_1)$ in Scenario 1 or $(1, j_2)$ in Scenario 2. We observe that the total holding cost for Scenario 1 is $w_p \cdot (d_{1j_1} + d_{2j_2})$ where for Scenario 2 we have $w_p \cdot (d_{1j_1} + d_{1j_2} + d_{2j_2})$. Obviously, the total holding cost increase for Scenario 2 as the flow time for project j_1 increases while it remains

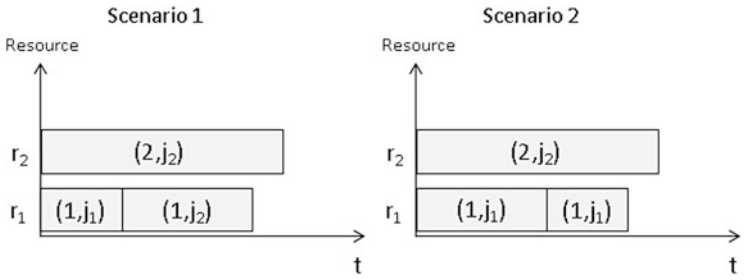

Fig. 7.7 Example illustrating the benefit of POPs

the same for project j_2 as activity $(2, j_2)$ is completed after completion of activity $(1, j_2)$.

If we have an order starting with the most advanced states defined on the set $\Sigma(p, s)$ according to Definition 7.3.2 such that $\sigma_1 \succ_a, \sigma_2 \succ_a \ldots \succ_a \sigma_{|\Sigma(p,s)|}$ we define $\sigma^{\mathrm{o}}(l, p, s) = \sigma_l \in \Sigma(p, s)$ to deliver the project state having rank l according to the order on $\Sigma(p, s)$.

The following theorem shows the project state ordering effect of a policy in Π^{PO} and implies that $\mathcal{S}^{\mathrm{PPO}} \subseteq \mathcal{S}^{\mathrm{P}}$. Furthermore, a sufficient condition for a state s be in $\mathcal{S}^{\mathrm{PPO}}$ is given.

Theorem 7.3.1. *The following properties hold for $\mathcal{S}^{PPO}$.*

(a) For any system state $s \in \mathcal{S}^{PPO}$ there is an order of the project states $\sigma \in \Sigma(p, s)\ \forall p \in \mathcal{P}$ such that $\sigma^o(1, p, s) \succ_{\mathrm{a}} \sigma^o(2, p, s) \succ_{\mathrm{a}} \ldots \succ_{\mathrm{a}} \sigma^o(|\Sigma(p, s)|, p, s)$.

(b) Any system state s with $K(s) \leq K^{\max}$ and an order of the project states $\sigma \in \Sigma(p, s)\ \forall p \in \mathcal{P}$ such that $\sigma^o(1, p, s) \succ_{\mathrm{a}} \sigma^o(2, p, s) \succ_{\mathrm{a}} \ldots \succ_{\mathrm{a}} \sigma^o(|\Sigma(p, s)|, p, s)$ must be in $\mathcal{S}^{PPO}$.

Proof. The proof of part (a) is by induction over the states visited on a sample path starting at the empty state $s_0 = s^0 = (0, \ldots, 0)$.

Induction start: For the case of an empty system ($s_0 = s^0$) and immediately after the arrival of the first project of type $p \in \mathcal{P}$, we have only one project in the system which is in state σ_p^I. Hence, $n(\sigma_p^I, s_1) = 1$ and $n(\sigma, s_1) = 0\ \forall \sigma \in \Sigma, \sigma \neq \sigma_p^I$ such that the induction assumption is met.

Induction step: We consider system state s_n accessed after n transitions. If $s_n = s^0$ the case is equivalent to the induction start. If $s_n \neq s^0$, we consider the transitions subsequent to the two types of events.

1. Arrival of a new project of type $p \in \mathcal{P}$: As all activities of the project are unfinished for the initial state σ_p^I we have $\sigma^{\mathrm{o}}(|\Sigma(p, s_{n+1})|, p, s_{n+1}) = \sigma_p^I \succeq_a \sigma^{\mathrm{o}}(|\Sigma(p, s_n)|, p, s_n)$.

2. Completion of an activity: We consider activity $i \in \mathcal{W}(\sigma^{\mathrm{o}}(l, p, s_n))$ of a project of type p in project state $\sigma^{\mathrm{o}}(l, p, s_n)$. Now, we assume that the activity is scheduled by decision $a_n = \pi^{\mathrm{PO}}(s_n)$. On completion of activity i the project state is $\sigma' = \sigma^{\mathrm{CPW}}(i, \sigma^{\mathrm{o}}(l, p, s_n))$ where $\mathcal{U}(\sigma') = \mathcal{U}(\sigma^{\mathrm{o}}(l, p, s_n))\backslash\{i\}$ such that $\sigma' \succ_a \sigma^{\mathrm{o}}(l, p, s_n)$.

   Furthermore, we know that $i \notin \mathcal{W}(\sigma^{\mathrm{o}}(l-1, p, s_{n+1}))$ as otherwise, by induction assumption, activity i from a project in state $\sigma^{\mathrm{o}}(l-1, p, s_n)$ would have been preferred by decision a_n. Thus, we have $\sigma^{\mathrm{o}}(l-1, p, s_{n+1}) = \sigma^{\mathrm{o}}(l-1, p, s_n) \succeq_a \sigma'$ such that we obtain for state s_{n+1} $\Sigma(p, s_{n+1}) = \Sigma(p, s_n) \cup \{\sigma'\}$ with

$$\sigma^{\mathrm{o}}(1, p, s_n) \succ_a \ldots \succ_a \sigma^{\mathrm{o}}(l-1, p, s_n) \succeq_a \sigma' \succ_a \sigma^{\mathrm{o}}(l, p, s_n) \succ_a \sigma^{\mathrm{o}}(l+1, p, s_n) \succ_a \ldots \succ_a \sigma^{\mathrm{o}}(|\Sigma(p, s_n)|, p, s_n)$$

   Note that if $n(\sigma^{\mathrm{o}}(l, p, s_n), s_n) = 1$ project state $\sigma^{\mathrm{o}}(l, p, s_n)$ must be eliminated from $\Sigma(p, s_{n+1})$ such that $\Sigma(p, s_{n+1}) = \Sigma(p, s_n) \cup \{\sigma'\}\backslash\{\sigma^{\mathrm{o}}(l, p, s_n)\}$. However, the order between the project states remains.

The proof of part (b) is by induction over the number of projects in the system $(K(s))$.

Induction start: For $K(s) = 0$ the system must be in s^0 such that the case is trivial.

Induction step: We consider any state s with $K(s) \leq K^{\max}$ and an order of the project states $\sigma \in \Sigma(p, s)\ \forall p \in \mathcal{P}$ such that

$$\sigma^{\mathrm{o}}(1, p, s) \succ_a \sigma^{\mathrm{o}}(2, p, s) \succ_a \ldots \succ_a \sigma^{\mathrm{o}}(|\Sigma(p, s)|, p, s)$$

Then, it is sufficient to show that s is accessible from a state s' with $K(s') = K(s)-1$ over a finite sequence of states s'' with $K(s'') = K(s) \leq K^{\max}$ that are again accessible from s' under some PO-policy. For state s' we have $\Sigma(p, s')$ with

$$\sigma^{\mathrm{o}}(1, p, s') \succ_a \sigma^{\mathrm{o}}(2, p, s') \succ_a \ldots \succ_a \sigma^{\mathrm{o}}(|\Sigma(p, s')|, p, s') \quad \forall p \in \mathcal{P}$$

and for the states s'' we have $\Sigma(p, s'')$ with

$$\sigma^{\mathrm{o}}(1, p, s'') \succ_a \sigma^{\mathrm{o}}(2, p, s'') \succ_a \ldots \succ_a \sigma^{\mathrm{o}}(|\Sigma(p, s'')|, p, s'') \quad \forall p \in \mathcal{P}$$

By induction assumption, s' is accessible from s^0 such that $s' \in \mathcal{S}^{\mathrm{PPO}}$. Thus, all states s'' are accessible from s' such that they are accessible from s^0 and we must have $s'' \in \mathcal{S}^{\mathrm{PPO}}$. Consequently, s must be accessible from s^0 over a sequence of states in $\mathcal{S}^{\mathrm{PPO}}$ such that $s \in \mathcal{S}^{\mathrm{PPO}}$ follows.

Now we have to consider two cases.

1. If $\sigma^{\mathrm{o}}(|\Sigma(p, s)|, p, s) = \sigma_p^{\mathrm{I}}$ for at least one project type $p \in \mathcal{P}$ then s is accessible by a single transition subsequent to an arrival from a state $s' \in \mathcal{S}^{\mathrm{PPO}}$ with $K(s')-1$ such that $s \in \mathcal{S}^{\mathrm{PPO}}$.

2. If $\sigma^{\mathrm{o}}(|\Sigma(p,s)|,p,s) \neq \sigma_p^{\mathrm{I}}\ \forall p \in \mathcal{P}$ we know that $\sigma^{\mathrm{o}}(|\Sigma(p,s)|,p,s) \succ_a \sigma_p^{\mathrm{I}}$ for any $p \in \mathcal{P}$ with $\Sigma(p,s) \neq \emptyset$. Now, we consider the following scenario. A project has just arrived in state $s''' = s - e(\sigma^{\mathrm{o}}(|\Sigma(p,s)|,p,s)) + e(\sigma_p^{\mathrm{I}})$ and for the project states in $\Sigma(p,s''')$ we must have

$$\sigma^{\mathrm{o}}(1,p,s''') = \sigma^{\mathrm{o}}(1,p,s) \succ_a \ldots \succ_a \sigma^{\mathrm{o}}(|\Sigma(p,s''')|-1,p,s''') = \sigma^{\mathrm{o}}(|\Sigma(p,s)|,p,s) \succ_a \sigma^{\mathrm{o}}(|\Sigma(p,s''')|,p,s''') = \sigma_p^{\mathrm{I}}$$

Next, we assume activities $i \notin \mathcal{U}(\sigma^{\mathrm{o}}(|\Sigma(p,s)|,p,s))$ of this project are always scheduled which is feasible under a POP, and that subsequent transitions are only related to completions of those activities. Hence, the subsequent system states are given by $s'' = s''' - e(\sigma_p^{\mathrm{I}}) + e(\sigma)$ where

$$\sigma^{\mathrm{o}}(|\Sigma(p,s)|,p,s) \succ_a \sigma^{\mathrm{o}}(|\Sigma(p,s'')|,p,s'') = \sigma \succ_a \sigma^{\mathrm{o}}(|\Sigma(p,s''')|,p,s''') = \sigma_p^{\mathrm{I}}$$

For each system state s'', we have for the project states in $\Sigma(p,s'')$

$$\sigma^{\mathrm{o}}(1,p,s) \succ_a \sigma^{\mathrm{o}}(2,p,s) \succ_a \ldots \succ_a \sigma^{\mathrm{o}}(|\Sigma(p,s)|,p,s) \succ_a \sigma$$

As only activities $i \notin \mathcal{U}(\sigma_{|\Sigma(p,s)|})$ are scheduled system state s must be entered such that it is accessible from s''' via states s'' under a POP. As s''' with $K^{\max}(s''') = K^{\max}(s)$ is entered just after an arrival the state is covered by case 1. Furthermore, the states s'' with $K^{\max}(s'') = K^{\max}(s)$ are accessible from s^0 via $s''' \in \mathcal{S}^{\mathrm{PPO}}$. Thus, we have $s'' \in \mathcal{S}^{\mathrm{PPO}}$. We conclude that s is accessible from s^0 via states s', s''' and s'' such that $s \in \mathcal{S}^{\mathrm{PPO}}$.

Finally, we note that if $n(\sigma_{|\Sigma(p,s)|},s) = 1$ $\sigma^{\mathrm{o}}(|\Sigma(p,s)|,p,s) \notin \Sigma(p,s'')$ and $\sigma^{\mathrm{o}}(|\Sigma(p,s)|,p,s) \notin \Sigma(p,s''')$ which does not have an effect on the order of the project states such that the induction step remains valid.

□

Due to the order between the project states in set $\Sigma(p,s)$, we exclude combinations of project states which may otherwise be possible such that $\mathcal{S}^{\mathrm{PPO}} \subseteq \mathcal{S}^{\mathrm{P}}$.

7.3.1.2 Project State Ordering Policies and Queueing Networks Representation of the System

In this section, we show, for unbounded $K^{\max}$, the equivalence of the state space $\mathcal{S}^{\mathrm{PPO}}$ and the state space $\mathcal{S}^{\mathrm{Q}}$ of a queueing network. In the queueing network activities of each type (i,p) wait in their own queue of length $n(i,p,s^{\mathrm{Q}})$ in system state $s^{\mathrm{Q}} \in \mathcal{S}^{\mathrm{Q}}$ which is given by

$$s^{\mathrm{Q}} = \big(n(i_1,p_1), n(i_2,p_2), \ldots, n(i_{|\mathcal{V}|},p_{|\mathcal{V}|})\big) \tag{7.64}$$

$\mathcal{V} = \{(i,p)|p \in \mathcal{P}, i \in \mathcal{V}_p\}$ is the set of all activity types. Thus, we write i_l and p_l for addressing the information of activity type $(i,p)_l$. Explicit representation of synchronization queues as considered for example by Cohen et al. [28] is not necessary since, as we will see, the information is implicitly contained in the state representation.

The equivalence between $\mathcal{S}^{\text{PPO}}$ and $\mathcal{S}^{\text{Q}}$ allows us to characterize and quantify more precisely the effect of POPs on the state space cardinality. Furthermore, it highlights the fact that, when using POPs, a queuing network representation is appropriate for the dynamic-stochastic multi-project scheduling problem with preemptions. For the investigation of the state space complexity, we use the following notation.

- $\sigma(\mathcal{W},p) = (\mathcal{W},\emptyset,p)$: Project state obtained from the combination of the set of waiting activities $\mathcal{W}$ and project type p.
- $\mathcal{U}(\mathcal{W},p)$: Set of all activities in the set $\mathcal{W}$ and their direct and indirect successors.
- $\mathcal{W}(\mathcal{U},p) = \{i \in \mathcal{U}|\forall i' \in \mathcal{U} : (i',i) \notin \mathcal{A}_p\}$: Set of activities in $\mathcal{U}$ of which no predecessors are in $\mathcal{U}$.
- $\mathcal{W}(p,s) = \bigcup_{\sigma \in \Sigma(p,s)} \mathcal{W}(\sigma)$: Index superset of all waiting activities related to projects of type p in system state s.
- $\mathcal{W}(p,s^{\text{Q}}) = \{i \in \mathcal{V}_p|n(i,p,s^{\text{Q}}) > 0\}$: Index superset of all waiting activities related to projects of type p in system state $s^{\text{Q}} \in \mathcal{S}^{\text{Q}}$.
- $\mathcal{U}(p,s) = \mathcal{U}(\mathcal{W}(p,s),p)$: Index superset of all unfinished activities related to projects of type p in system state $s \in \mathcal{S}^{\text{P}}$.
- $\mathcal{U}(p,s^{\text{Q}}) = \mathcal{U}(\mathcal{W}(p,s^{\text{Q}}),p)$: Index superset of all unfinished activities related to projects of type p in system state $s^{\text{Q}} \in \mathcal{S}^{\text{Q}}$.
- $n(i,p,s) = \sum_{\sigma \in \Sigma(p,s)} \mathbb{1}\{i \in \mathcal{W}(\sigma)\} n(\sigma,s)$: Number of activities of type (i,p) that are waiting in system state $s \in \mathcal{S}^{\text{P}}$.

Now, we state the main result delivering a bijection m which allows to conclude the equivalence of $\mathcal{S}^{\text{PPO}}$ and $\mathcal{S}^{\text{Q}}$ for unbounded $K^{\max}$.

Theorem 7.3.2. *For unbounded $K^{\max}$ there exists a bijection $m : \mathcal{S}^{PPO} \longleftrightarrow \mathcal{S}^{Q}$ mapping a state in $\mathcal{S}^{PPO}$ on a state in $\mathcal{S}^{Q}$ and vice versa. m is given by*

(i) $m : \mathcal{S}^{PPO} \longleftrightarrow \mathcal{S}^{Q}$*:*

$$n(i,p,s^{Q}) = n(i,p,s)\ \forall p \in \mathcal{P}, i \in \mathcal{V}_p \tag{7.65}$$

(ii) $m^{-1} : \mathcal{S}^{Q} \longleftrightarrow \mathcal{S}^{PPO}$*:*

Require: $s^{Q} \in \mathcal{S}^{Q}$

1: $s = s^0$

2: ***for*** $p \in \mathcal{P}$ ***do***

3: $\quad l \leftarrow 1$

4: $\quad s_1^{Q} = s^{Q}$

5: $\quad$ ***while*** *not* $n(i,p,s_l^{Q}) = 0\ \forall i \in \mathcal{V}_p$ ***do***

6: $\sigma_{lp}^{\min} = \sigma(\mathcal{W}(\mathcal{U}(s_l^Q, p), p), p)$

7: $n(\sigma_{lp}^{\min}, s) = \min\left\{n(i, p, s_l^Q) | i \in \mathcal{W}(\sigma_{lp}^{\min}, p)\right\}$

8: *Compute* s_{l+1}^Q *by letting for all* $p' \in \mathcal{P}, i \in \mathcal{V}_{p'}$

$$n(i, p', s_{l+1}^Q) = \begin{cases} n(i, p', s_l^Q) - n(\sigma_{lp}^{\min}, s) \; i \in \mathcal{W}(\sigma_{lp}^{\min}), p' = p \\ n(i, p', s_l^Q) \qquad\qquad\qquad\qquad otherwise \end{cases} \tag{7.66}$$

9: $l \leftarrow l + 1$

10: ***end while***

11: ***end for***

Note that a sequence of $\sigma_{lp}^{\min}$ for a project type $p \in \mathcal{P}$ constitutes a set $\Sigma(p, s)$ of state $s = m^{-1}(s^{\mathrm{Q}}) \in \mathcal{S}^{\mathrm{PPO}}$. Before we give the proof of Theorem 7.3.2, we state two lemmas needed to retrieve project state $\sigma^{\mathrm{o}}(|\Sigma(p, s)|, p, s)$ and $n(\sigma^{\mathrm{o}}(|\Sigma(p, s)|), p, s)$.

Lemma 7.3.1. *For any state* $s \in \mathcal{S}^{PPO}$*, we have*

$$\sigma^{o}(|\Sigma(p, s)|, p, s) = \sigma(\mathcal{W}(\mathcal{U}(p, s), p), p) \tag{7.67}$$

Proof. Be $L = |\Sigma(p, s)|$. From $\mathcal{U}(\sigma^{\mathrm{o}}(1, p, s)) \subset \mathcal{U}(\sigma^{\mathrm{o}}(2, p, s)) \subset \ldots \subset \mathcal{U}(\sigma^{\mathrm{o}}(L, p, s))$ it follows that

$$\mathcal{U}(p, s) = \mathcal{U}(\mathcal{W}(p, s), p) = \mathcal{U}(\sigma^{\mathrm{o}}(L, p, s)) \tag{7.68}$$

Then, $\mathcal{W}(\mathcal{U}(p, s), p)$ gives the set of activities without predecessors which is equal to $\mathcal{W}(\sigma^{\mathrm{o}}(L, p, s))$. Thus, we obtain the project state by

$$\sigma^{\mathrm{o}}(L, p, s) = \sigma(\mathcal{W}(\mathcal{U}(p, s), p), p).$$

□

The second lemma gives $n(\sigma^{\mathrm{o}}(|\Sigma(p, s)|, p, s), s)$.

Lemma 7.3.2. *For any state* $s \in \mathcal{S}^{PPO}$*, we have*

$$n(\sigma^{o}(|\Sigma(p, s)|, p, s), s) = \min\left(n(i, p, s) | i \in \mathcal{W}(\sigma^{o}(|\Sigma(p, s)|, p, s))\right) \tag{7.69}$$

Proof. Be $L = |\Sigma(p, s)|$. From

$$\mathcal{U}(p, s) = \mathcal{U}(\sigma^{\mathrm{o}}(L, p, s)) \supset \ldots \supset \mathcal{U}(\sigma^{\mathrm{o}}(1, p, s))$$

we have unfinished activities with $i \in \mathcal{U}(\sigma^{\mathrm{o}}(L, p, s)) \backslash \mathcal{U}(\sigma^{\mathrm{o}}(L-1, p, s))$ that are in no other set of unfinished activities such that $i \notin \mathcal{W}(\sigma^{\mathrm{o}}(l, p, s)) \subseteq \mathcal{U}(\sigma^{\mathrm{o}}(l, p, s)) \; \forall l = 1, \ldots, L-1$.

Furthermore, we must have some unfinished activities waiting such that $\mathcal{W}(\sigma^{\mathrm{o}}(L,p,s)) \cap (\mathcal{U}(\sigma^{\mathrm{o}}(L,p,s)) \backslash \mathcal{U}(\sigma^{\mathrm{o}}(L-1,p,s))) \neq \emptyset$. This follows from the fact that if we had $\mathcal{W}(\sigma^{\mathrm{o}}(L,p,s)) \subseteq \mathcal{W}(\sigma^{\mathrm{o}}(l,p,s))$ for some $l \in [1, L-1]$ we would have $\mathcal{U}(\sigma^{\mathrm{o}}(L,p,s)) \subseteq \mathcal{U}(\sigma^{\mathrm{o}}(l,p,s))$. Hence, for activities with $i \in \mathcal{W}(\sigma^{\mathrm{o}}(|L|,p,s)) \cap (\mathcal{U}(\sigma^{\mathrm{o}}(|L|,p,s)) \backslash \mathcal{U}(\sigma^{\mathrm{o}}(|L|-1,p,s)))$ we must have

$$n(i,p,s) = n(\sigma^{\mathrm{o}}(|L|,s),p,s).$$

By contrast for all activities with $i \in \mathcal{W}(\sigma^{\mathrm{o}}(|L|,p,s),s) \cap \mathcal{W}(\sigma^{\mathrm{o}}(l,p,s))$ (where $\mathcal{W}(\sigma^{\mathrm{o}}(l,p,s)) \subseteq \mathcal{U}(\sigma^{\mathrm{o}}(l,p,s))$) for at least other project state with $l = 1,\ldots,L-1$, we must have $n(i,p,s) > n(\sigma^{\mathrm{o}}(L,s),p,s)$. Thus, the assertion follows. □

Now, we give the proof of Theorem 7.3.2

Proof. In order to show that the mapping m is a bijection we show, firstly, that m is injective such for a state in $s \in \mathcal{S}^{\mathrm{PPO}}$ $m^{-1}(m(s)) = s$. Secondly, we show that m^{-1} is injective such that for a state in $s^{\mathrm{Q}} \in \mathcal{S}^{\mathrm{Q}}$ $m(m^{-1}(s^{\mathrm{Q}})) = s^{\mathrm{Q}}$.

The first part for showing that $m^{-1}(m(s)) = s$ is done by induction.

We assume that we have obtained a state $s^{Q} = m(s) \in \mathcal{S}^{\mathrm{Q}}$ from a state $s \in \mathcal{S}^{\mathrm{PPO}}$. From the definition of $m(s)$, we have $n(i,p,s^{\mathrm{Q}}) = n(i,p,s)\ \forall p \in \mathcal{P}, \forall i \in \mathcal{V}_p$. Furthermore, $\mathcal{W}(p,s^{\mathrm{Q}}) = \mathcal{W}(p,s)$ since, for all $i \in \mathcal{W}(p,s)$, there must exist at least one $\sigma \in \Sigma(p,s)$ with $i \in \mathcal{W}(\sigma)$. Thus, $n(i,p,s) > 0$ and $i \in \mathcal{W}(p,s^{\mathrm{Q}})$ which implies $\mathcal{U}(s^{\mathrm{Q}},p) = \mathcal{U}(s,p)$.

Now, in order to show that the recursion of m^{-1} retrieves s from s^{Q} it is sufficient to show that the intermediate states $s_l^{Q} \in \mathcal{S}^{\mathrm{Q}}$ can be obtained from states $s_l \in \mathcal{S}^{\mathrm{PPO}}$ by $s_l^{Q} = m(s_l)$. A state $s_l \in \mathcal{S}^{\mathrm{PPO}}$ characterized by $\Sigma(p,s_l) = \{\sigma^{\mathrm{o}}(1,p,s),\ldots,\sigma^{\mathrm{o}}(|\Sigma(p,s)|-(l-1),p,s)\} \subseteq \Sigma(p,s)$ where $n(\sigma,s_l) = n(\sigma,s)\ \forall \sigma \in \Sigma(p,s_l)$, $n(\sigma,s_l) = 0\ \forall \sigma \in \Sigma(p,s) \backslash \Sigma(p,s_l)$ and $n(\sigma,s_l) = n(\sigma,s)\ \forall \sigma \in \Sigma(p',s), p' \in \mathcal{P} \neq p$.

As $\mathcal{U}(s_l^{\mathrm{Q}},p) = \mathcal{U}(s_l,p)$ we obtain for state $s' = m^{-1}(m(s))$ from Lemma 7.3.1

$$\sigma_{lp}^{\min} = \sigma^{\mathrm{o}}(|\Sigma(p,s)|-l+1,p,s) \in \Sigma(p,s)$$

and from Lemma 7.3.2

$$n(\sigma_{lp}^{\min}) = n(\sigma(|\Sigma(p,s)|-l+1,p,s),s)$$

Thus, after termination of m^{-1}, we obtain for state s'

$$\begin{aligned}\Sigma(p,s') &= \{\sigma^{\mathrm{o}}(|\Sigma(p,s)|,p,s),\sigma^{\mathrm{o}}(|\Sigma(p,s)|-1,p,s),\ldots,\sigma^{\mathrm{o}}(1,p,s)\} \\ &= \Sigma(p,s)\ \forall p \in \mathcal{P}\end{aligned}$$

where, at position l, we have

$$n(\sigma^{\mathrm{o}}(|\Sigma(p,s)|-l+1,p,s),s') = n(\sigma^{\mathrm{o}}(|\Sigma(p,s)|-l+1,p,s),s)$$

Induction start: As $s_1^{\mathrm{Q}} = s^{\mathrm{Q}} = m(s) = m(s_1)$ the assertion is met for $l = 1$.

Induction step: Be given state s_l^{Q} which is assumed to be obtained from a state s_l.

From Lemmas 7.3.1 and 7.3.2, we obtain $\sigma_{lp}^{\min}$ and $n(\sigma_{lp}^{\min}, s_l)$. In order to obtain s_{l+1}^{Q} we set $n_{ip'}(s_{l+1}^{\mathrm{Q}}) = n_{ip'}(s_l^{\mathrm{Q}}) - n(\sigma_{lp}^{\min}, s_l)\ \forall i \in \mathcal{W}(\sigma_l^{\min})$ if $p' = p$ and $n_{ip'}(s_{l+1}^{\mathrm{Q}}) = n_{ip'}(s_l^{\mathrm{Q}})$ otherwise.

Now, we observe that s_{l+1}^{Q} is obtained from s_l^{Q} by not adding $n(\sigma_{lp}^{\min}, s_l)$ to the state variables $n(i, p, s_l^{\mathrm{Q}})\ \forall i \in \mathcal{W}(\sigma_{lp}^{\min})$.

Hence, s_{l+1}^{Q} can also be obtained from a state $s_{l+1} \in \mathcal{S}^{\mathrm{PPO}}$ where

$$\Sigma(p, s_{l+1}) = \Sigma(p, s_l) \backslash \{\sigma_{lp}^{\min}\} = \{\sigma^{\mathrm{o}}(1, p, s), \ldots, \sigma^{\mathrm{o}}(|\Sigma(p, s)| - l, p, s)\}$$

Thus, the assertion follows.

Before showing that $m(m^{-1}(s^{\mathrm{Q}})) = s^{\mathrm{Q}}$ we have to make sure in the second part that $s = m^{-1}(s^{\mathrm{Q}}) \in \mathcal{S}^{\mathrm{PPO}}\ \forall s^{\mathrm{Q}} \in \mathcal{S}^{\mathrm{Q}}$.

If $s \in \mathcal{S}^{\mathrm{PPO}}$ we must have, as stated by Theorem 7.3.1, an order for the sets $\Sigma(p, s)\ \forall p \in \mathcal{P}$ such that $\sigma^{\mathrm{o}}(1, p, s) \succ_a \sigma^{\mathrm{o}}(2, p, s) \succ_a \ldots \succ_a \sigma^{\mathrm{o}}(|\Sigma(p, s)|, p, s)$.

Thus, it is sufficient to show that we have $\mathcal{U}(s_1^{\mathrm{Q}}, p) \supset \mathcal{U}(s_2^{\mathrm{Q}}, p) \supset \ldots \supset \mathcal{U}(s_L^{\mathrm{Q}}, p)$ such that $\sigma_{1p}^{\min} \prec \sigma_{2p}^{\min} \prec \ldots \prec \sigma_{Lp}^{\min}$ where L is the last iteration after which $n(i, p, s_L^{\mathrm{Q}}) = 0\ \forall i \in \mathcal{V}_p$. Any state having this property must be in $\mathcal{S}^{\mathrm{PPO}}$ according to Theorem 7.3.1 part (b). The proof is by induction over the iterations $l = 1, \ldots, L$.

Induction start: The case is trivial for $l = 1$.

Induction step: By $\mathcal{U}(s_l^{\mathrm{Q}}, p) = \mathcal{U}(\mathcal{W}(p, s_l^{\mathrm{Q}}), p)$, we obtain the index superset of unfinished activities related to projects of type p in system state s_l^{Q}. By $\mathcal{W}(\mathcal{U}(s_l^{\mathrm{Q}}, p), p) = \mathcal{W}(\sigma_{lp}^{\min})$, the set of waiting activities (activities with $i \in \mathcal{U}(s_l^{\mathrm{Q}}, p)$ without predecessors in $\mathcal{U}(s_l^{\mathrm{Q}}, p)$), belonging to $\mathcal{U}(s_l^{\mathrm{Q}}, p)$, is obtained. Now, as activity $i \in \mathcal{W}(\sigma_{lp}^{\min})$ cannot be successor of another activity $i' \in \mathcal{W}(\sigma_{lp}^{\min})$ of a project in state $\sigma_{lp}^{\min}$, we have the property that if the activity index $i \in \mathcal{W}(\sigma_{lp}^{\min})$ is removed from $\mathcal{W}(p, s_l^{\mathrm{Q}})$ the superset will no longer be obtained such that $\mathcal{U}(\mathcal{W}(\sigma_{lp}^{\min}) \backslash \{i\}) \subset \mathcal{U}(\mathcal{W}(\sigma_{lp}^{\min}))$.

As, by definition of $n(\sigma_l^{\min}, s)$, we have $n(\sigma_l^{\min}, s) = n(i, p, s_l^{\mathrm{Q}})$ for some activity types (i, p) with $i \in \mathcal{W}(\sigma_l^{\min})$ we must have

$$n(i, p, s_{l+1}^{\mathrm{Q}}) = n(i, p, s_l^{\mathrm{Q}}) - n(\sigma_{lp}^{\min}, s) = 0$$

Thus, we have $\mathcal{W}(p, s_{l+1}^{\mathrm{Q}}) \subset \mathcal{W}(p, s_l^{\mathrm{Q}})$ and $\mathcal{U}(\mathcal{W}(p, s_{l+1}^{Q}), p) \subset \mathcal{U}(\mathcal{W}(p, s_l^{Q}), p)$ such that $\sigma_{l+1}^{\min} \succ_a \sigma_l^{\min}$. Hence, the assertion follows.

Finally, we show that $m(m^{-1}(s^{\mathrm{Q}})) = s^{\mathrm{Q}}$. From the fact that $\mathcal{W}(p, s_{l+1}^{\mathrm{Q}}) \subset \mathcal{W}(p, s_l^{\mathrm{Q}})$, we know for activity type (i, p) with $i \in \mathcal{W}(p, s^{\mathrm{Q}})$ that $n(i, p, s_{l'}^{\mathrm{Q}}) = 0$

for some $l' > 1$ while $n(i, p, s^{\mathrm{Q}}_{l'-1}) > 0$. Thus, $i \notin \mathcal{W}(\sigma^{\min}_{l'p})$ as we must have $\mathcal{W}(\sigma^{\min}_{l'p}) \subseteq \mathcal{W}(s^{\mathrm{Q}}_{l'}, p)$. Hence, for all following iterations $l'' = l' + 1, \ldots, L$ we have $i \notin \mathcal{W}(\sigma^{\min}_{l''p})$ and $n(i, p, s^{\mathrm{Q}}_{l''}) = 0$ as well.

This leads us to

$$\sum_{\substack{l=1 \\ i \in \mathcal{W}(\sigma^{\min}_{kp})}}^{L} n(\sigma^{\min}_{lp}, s) = n(i, p, s^{\mathrm{Q}})$$

Hence applying m recovers $n(i, p, s^{\mathrm{Q}}) = n(i, p, s)$. This completes the proof. □

From Theorem 7.3.2 we can conclude that using POPs a large part of the complexity resulting from networks with few precedence relations can be eliminated.

Corollary 7.3.1. *For unbounded $K^{\max}$ $|\mathcal{S}^{PPO}|$ is independent of the structure of the activity type networks $\mathcal{G}_p$.*

Proof. If no limit is imposed on the number of projects in the system any vector of queue lengths for the activity types (i, p) gives a feasible state $s^{\mathrm{Q}} \in \mathcal{S}^{\mathrm{Q}}$. The feasibility follows from the fact that according to Theorem 7.3.2 it can be mapped to a feasible state $s \in \mathcal{S}^{\mathrm{PPO}}$. As the number of feasible combinations of queue lengths (and thus the states s^{Q}) is independent from the network structure the assertion follows. □

The next theorem gives an interval for $|\mathcal{S}^{\mathrm{PPO}}|$ with limited $K^{\max}$.

Theorem 7.3.3. *If we allow a maximum of $K^{\max}$ projects in the system we have*

$$\binom{K^{\max} + \sum\limits_{p \in \mathcal{P}} |\mathcal{V}_p|}{\sum\limits_{p \in \mathcal{P}} |\mathcal{V}_p|} \leq |\mathcal{S}^{PPO}| \leq (K^{\max} + 1)^{\sum\limits_{p \in \mathcal{P}} |\mathcal{V}_p|} \tag{7.70}$$

Proof. The lower bound on the left is taken from Corollary 7.1.1 and is still valid as for networks with strict linear orders POPs do not effect state space cardinality.

For deriving the upper bound we use the equivalence $\mathcal{S}^{\mathrm{PPO}} \iff \mathcal{S}^{\mathrm{Q}}$ for unbounded $K^{\max}$. Then, if $K^{\max}$ is bounded the queue length for each activity type (i, p) is bounded by $K^{\max}$ such that $n(i, p, s^{\mathrm{Q}}) \leq K^{\max}$. Hence, a queue may have $K^{\max} + 1$ different lengths. However, as $K^{\max}$ restricts the number of projects in the system queue lengths may not have a value of $K^{\max}$ at the same time, for example when activities are to be processed in a strict linear order. Ignoring such interdependencies between queues due to $K^{\max}$ and precedence relations we have at most $(K^{\max} + 1)^{\sum\limits_{p \in \mathcal{P}} |\mathcal{V}_p|}$ combinations of the queue lengths. Note that, the upper bound is tight for the case of a single project type without precedence relations between activity types where queue lengths may have the maximum length $K^{\max}$ at the same time. This can be seen as follows. Starting at an empty system projects may

arrive before completion of any activity until $K(s) = K^{\max}$. As any state $s^{\mathrm{Q}} \in \mathcal{S}^{\mathrm{Q}}$ with queue length smaller or equal to $K^{\max}$ may be mapped to a state $s \in \mathcal{S}^{\mathrm{PPO}}$ any combination of queue lengths is feasible such that the upper bound is tight. □

While the lower bound remains equal to the lower bound when no POPs are considered, the new upper bound on the right hand side is much more tight such that for Example 7.1.3 we obtain $3{,}003 \leq |\mathcal{S}^{\mathrm{PPO}}| \leq 161{,}051$. Thus, the state space cardinality lies in a range that can easily be handled by the solution methodologies addressed in Chap. 4.

To summarize the findings, POPs help to reduce the state space cardinalities by a reduction of the dimensionality of the problem. While for general policies the state space cardinality is largely determined by the total number project states in Σ^{P}, the state space cardinality using POPs is largely determined by the total number of activity types in $\mathcal{V}$ where $|\Sigma^{\mathrm{P}}| \geq |\mathcal{V}|$.

7.3.2 Non-preemptive Project State Ordering Policies

At first, we define the set $\mathcal{C}(\sigma_1, \sigma_2) = (\mathcal{W}(\sigma_1) \cup \mathcal{E}(\sigma_1)) \cap (\mathcal{W}(\sigma_2) \cup \mathcal{E}(\sigma_2))$ being the set of common activities being ready for execution (in the sense that all predecessors have been completed) of two projects in project states σ_1, σ_2 with $p(\sigma_1) = p(\sigma_2)$. Then, we define the two sets $\mathcal{E}_1^{\mathrm{C}}(\sigma_1, \sigma_2) = \mathcal{E}(\sigma_1) \cap \mathcal{C}(\sigma_1, \sigma_2)$ and $\mathcal{E}_2^{\mathrm{C}}(\sigma_1, \sigma_2) = \mathcal{E}(\sigma_2) \cap \mathcal{C}(\sigma_1, \sigma_2)$ being the subsets of activities being in process that are also in the common set of ready activities.

Next, we extend the relation between project states already defined for the preemptive problem by Definition 7.3.1 to the non-preemptive problem.

Definition 7.3.3. We have two project states σ_1, σ_2. Then project state σ_1 is **more advanced** than σ_2 if $p(\sigma_1) = p(\sigma_2)$ and one of the two conditions holds:

1. $\mathcal{U}(\sigma_1) \subset \mathcal{U}(\sigma_2)$ and $\mathcal{E}_1^{\mathrm{C}}(\sigma_1, \sigma_2) \supseteq \mathcal{E}_2^{\mathrm{C}}(\sigma_1, \sigma_2)$
2. $\mathcal{U}(\sigma_1) = \mathcal{U}(\sigma_2)$ and $\mathcal{E}_1^{\mathrm{C}}(\sigma_1, \sigma_2) \supset \mathcal{E}_2^{\mathrm{C}}(\sigma_1, \sigma_2)$.

The term $\mathcal{E}_1^{\mathrm{C}}(\sigma_1, \sigma_2) \supset \mathcal{E}_2^{\mathrm{C}}(\sigma_1, \sigma_2)$ takes into account that activities of a project in a more advanced state may be in process but must be waiting in case of the project in the less advanced state. As $c_r = 1 \ \forall r \in \mathcal{R}$ $\mathcal{E}_1^{\mathrm{C}}(\sigma_1, \sigma_2) \supset \mathcal{E}_2^{\mathrm{C}}(\sigma_1, \sigma_2)$ implies $\mathcal{E}_2^{\mathrm{C}}(\sigma_1, \sigma_2) = \emptyset$.

Thus, for any $i \in \mathcal{C}(\sigma_1, \sigma_2)$ i may be in $\mathcal{E}(\sigma_1)$ or in $\mathcal{W}(\sigma_1)$ but must be in $\mathcal{W}(\sigma_2)$. Again, we shortly write $\sigma_1 \succ_a \sigma_2$ in order to say that project state σ_1 is *more advanced* than project state σ_2. Now, we repeat the definition of the class of *project state ordering* policies (POPs) which is taken from Sect. 7.3.1.1 without modifications.

Definition 7.3.4. Let $\sigma_1, \sigma_2 \in \Sigma(p, s)$ be two projects states respectively with $\sigma_1 \succ_a \sigma_2$. Then, a non-preemptive *project state ordering* policy π^{PO} always prefers

activity $i \in \mathcal{W}(\sigma_1)$ from a project in state σ_1 over activity $i \in \mathcal{W}(\sigma_2)$ of a project in state σ_2 $\forall i \in \mathcal{W}(\sigma_1) \cap \mathcal{W}(\sigma_2)$.

By $\Pi^{\text{PO}} \subset \Pi$ we denote the class of non-preemptive *project state ordering* policies (POPs) where $\mathcal{S}^{\text{PO}}$ is the set of states (state space) that communicate with s^0 under at least one non-preemptive POP plus the empty system state s^0.

The class of policies formalizes also for the non-preemptive case the intuition that activities of a project being in a more advanced state tend to be more critical than those of a project in a less advanced state. Recall that criticality refers the probability to lie on the critical path (cf. Elmaghraby [46]). To see this, let us consider the following example.

Example 7.3.2. Basically the idea of non-preemptive POPs corresponds to the idea of preemptive POPs (cf. Example 7.3.1). In addition, we may have the case where $\mathcal{U}(\sigma_2) = \mathcal{U}(\sigma_1)$ and $\mathcal{E}_1^{\text{C}}(\sigma_1, \sigma_2) \supset \mathcal{E}_2^{\text{C}}(\sigma_1, \sigma_2)$. Then, (i', j_1) with $i' \in \mathcal{U}(\sigma_1)$ tend to be more critical than activities (i', j_2) which can be seen as follows. For each activity (i, j_1) with $i \in \mathcal{E}_1^{\text{C}}(\sigma_1, \sigma_2)$, we must have $i \in \mathcal{W}(\sigma_2)$ such that (i, j_1) is completed before (i, j_2). Thus, (i, j_2) is more likely to delay completion of project j_2 than (i, j_1).

Finally, we prove a structural property for the state space of the non-preemptive problem that explains the name of the policy class.

Theorem 7.3.4. *Following an POP π^{PO} for any system state $s \in \mathcal{S}^{PO}$ we have for the projects of type $p \in \mathcal{P}$ an order of the project states $\sigma \in \Sigma(p, s)$ such that $\sigma^o(1, p, s) \succ_{\text{a}} \sigma^o(2, p, s) \succ_{\text{a}} \ldots \succ_{\text{a}} \sigma^o(|\Sigma(p, s)|, p, s)$.*

Proof. The proof is by induction over the states visited on a sample path starting at the empty state s^0 while a POP π^{PO} is followed.

Induction start: For the case of an empty system ($s_0 = s^0$) and immediately after the arrival of the first project of type $p \in \mathcal{P}$, we have only one project in the system which is in state σ_p^I. Thus, $n(\sigma_p^I, s_1) = 1$ and $n(\sigma, p) = 0$ $\forall \sigma \neq \sigma_p^I$ such that the induction assumption is met.

Induction step: We consider system state s_n entered after n transitions. If $s_n = s_0$ the case is equivalent to the induction start. If $s_n \neq s_0$ we first analyze the effect of a scheduling decision $a_n = \pi^{\text{PO}}(s_n)$ and show that also for the post-decision states the induction assumption is met.

Clearly, if only one project of type p is in the system the induction assumption is always met. For the effect of the scheduling decision in case of multiple projects, we consider two projects $k_1, k_2 \in [1, \ldots, K(s_n)]$ with $\sigma(k_1, s_n) \succeq_a \sigma(k_2, s_n)$. For $\sigma(k_1, s_n) \succ_a \sigma(k_2, s_n)$, π^{PO} must prefer activities (i, k_1) with $i \in \mathcal{W}(\sigma(k_1, s_n)) \cap \mathcal{W}(\sigma(k_2, s_n))$. If $\sigma(k_1, s_n) = \sigma(k_2, s_n)$ w.l.o.g. activities of project k_1 are preferred over activities of k_2. If no further activities of both projects are scheduled we must have $\sigma^S(\mathcal{B}(a_n, k_1, s_n), \sigma(k_1, s_n)) \succeq_a \sigma^S(\mathcal{B}(a_n, k_2, s_n), \sigma(k_2, s_n))$.

Next, we consider activities (i, k_1) with $i \in \mathcal{W}(\sigma(k_1, s_n))$ and $i \notin \mathcal{W}(\sigma(k_2, s_n))$. We know that activities (i, k_2) are not yet ready for execution as they neither

can be finished since $\mathcal{U}(\sigma(k_1, s_n)) \subseteq \mathcal{U}(\sigma(k_2, s_n))$ nor be in process since $\mathcal{E}_1^{\mathrm{C}}(\sigma(k_1), \sigma(k_2)) \supseteq \mathcal{E}_2^{\mathrm{C}}(\sigma(k_1), \sigma(k_2))$.

Next, we consider activities (i, k_2) with $i \notin \mathcal{W}(\sigma(k_1, s_n))$ and $i \in \mathcal{W}(\sigma(k_2, s_n))$. From $\mathcal{U}(\sigma(k_1, s_n)) \subseteq \mathcal{U}(\sigma(k_2, s_n))$ we know that if all predecessors have been completed for (i, k_2) they must have been completed for activity (i, k_1) as well. Thus, activities (i, k_1) must already have been completed or are in process ($i \in \mathcal{E}(\sigma(k_1, s_n))$).

Thus, no activities are scheduled of project k_1 that already have been finished for project k_2 as well as no activities of project k_2 are scheduled that are still unfinished for project k_1. As a consequence, we still have $\sigma^S(\mathcal{B}(a_n, k_1, s_n), \sigma(k_1, s_n)) \succeq_a \sigma^S(\mathcal{B}(a_n, k_2, s_n), \sigma(k_2, s_n))$ such that for the post-decision state $\hat{s}_n = \hat{s}(s_n, a_n)$ the induction assumption is met.

As a next step, starting with post-decision state $\hat{s}_n$, we consider the transitions subsequent to the two types of events.

1. Arrival of a new project of type $p \in \mathcal{P}$: As for the initial state σ_p^I all activities of the project are unfinished and not in process ($\mathcal{E}(\sigma_p^I) = \emptyset$) we have $\sigma_p^I \prec_a \sigma^{\mathrm{o}}(|\Sigma(p, \hat{s}_n)|, p, \hat{s})$. Thus, the induction assumption is met for $s_{n+1} = s^{\mathrm{A}}(s_n, a_n, p)$.
2. Completion of activity i of a project in state $\sigma^{\mathrm{o}}(l, p, \hat{s}_n)$ such that the project enters state $\sigma' = \sigma^{\mathrm{C}}(i, \sigma^{\mathrm{o}}(l, p, \hat{s}_n))$. Clearly, we have $\mathcal{U}(\sigma') \subset \mathcal{U}(\sigma^{\mathrm{o}}(l, p, \hat{s}_n))$. Furthermore, we know that $i \notin \mathcal{U}(\sigma^{\mathrm{o}}(l-1, p, \hat{s}_n))$ – We must have $i \notin \mathcal{W}(\sigma^{\mathrm{o}}(l-1, p, \hat{s}_n))$ as $\sigma^{\mathrm{o}}(l-1, p, \hat{s}_n) \succ_a \sigma^{\mathrm{o}}(l, p, \hat{s}_n)$ and $i \notin \mathcal{E}(\sigma^{\mathrm{o}}(l-1, p, \hat{s}_n))$ as no other activity can be scheduled on resource $r_{ip(\sigma^{\mathrm{o}}(l,p,\hat{s}_n))}$ due to $c_r = 1\ \forall r \in \mathcal{R}$. Hence, we have $\mathcal{U}(\sigma^{\mathrm{o}}(l-1, p, \hat{s}_n)) \subseteq \mathcal{U}(\sigma')$ and $\mathcal{E}_1^{\mathrm{C}}(\sigma^{\mathrm{o}}(l-1, p, \hat{s}_n), \sigma') \supseteq \mathcal{E}_2^{\mathrm{C}}(\sigma^{\mathrm{o}}(l-1, p, \hat{s}_n), \sigma')$ such that we have, for $\Sigma(p, s_{n+1})$ of system state $s_{n+1} = s^{\mathrm{C}}(s_n, a_n, i, \sigma^{\mathrm{o}}(l, p, s))$,

$$\sigma^{\mathrm{o}}(1, p, \hat{s}_n) \succ_a \ldots \succ_a \sigma^{\mathrm{o}}(l-1, p, \hat{s}_n) \succeq_a \sigma' \succ_a \sigma^{\mathrm{o}}(l+1, p, \hat{s}_n) \succ_a \ldots \succ_a \sigma^{\mathrm{o}}(|\Sigma(p, \hat{s}_n)|, p, \hat{s}_n)$$

   Note that $\sigma^{\mathrm{o}}(l, p, \hat{s}_n)$ is no longer in $\Sigma(p, s_{n+1})$ as each project state having activities in process can only be of one project at a time due to $c_r = 1$.

To conclude the proof, we briefly address removals of project states from $\Sigma(p, s_n)$. If $n(\sigma(k, s_n), s_n) = 1$ and $\mathcal{B}(a_n, k, s_n) \neq \emptyset$ we have $\Sigma(p, \hat{s}_n) = \Sigma(p, s_n) \backslash \{\sigma(k, s_n)\}$ as the project changes its state such that $n(\sigma(k, s), \hat{s}_n) = 0$. However the removal of $\sigma(k, s)$ does not affect the order between the states $\sigma \in \Sigma(p, \hat{s})$. The same is true for transitions from project state $\sigma^{\mathrm{o}}(l, p, \hat{s}_n)$ where $n(\sigma^{\mathrm{o}}(l, p, s), \hat{s}_n) = 1$. □

Obviously, the number of possible sets $\Sigma(p, s)$ of projects states that may occur is restricted such that $\mathcal{S}^{\mathrm{PO}} \subseteq \mathcal{S}$.

Table 7.8 State space cardinalities for general and for PO-policies

Policy class	Non preemptive	Preemptive
General	683,209	53,130
PO	102,838	19,481
% (PO/General)	15.05	36.67

7.3.3 Project State Ordering Priority Policies

If a RBP (cf. Definition 6.1.1) only considers the activities that are in line with the definition of a POP (cf. Definition 7.3.2) a resource based project state ordering priority policy (RBPOP) is obtained. For simplicity, we also refer to *PO-priority policies*.

If the restriction to POPs implies a reduction of alternative decisions without much loss of performance, elimination of some suboptimal decisions can be expected. Thus, PO-priority policies may show an improved performance over general priority policies.

7.3.4 Numerical Example

Considering the numerical example from Sect. 7.1.3 we observe that for a high rejection cost $y_1 = 1{,}000$ the optimal policies are POPs. This can be seen by the fact that the rules for the decisions of resources 2 and 3 prefer activities from projects in more advances states where $\sigma_5 \succ_a \sigma_2$ and $\sigma_5 \succ_a \sigma_3$ for the preemptive problem. For the non-preemptive problem, we additionally have $\sigma_{10} \succ_a \sigma_5$ and $\sigma_{11} \succ_a \sigma_5$.

Table 7.8 shows the potential to reduce the state space cardinality by using POPs. By $\%(PO/General)$ we refer to the fraction of the state space cardinality for POPs from the state space cardinality for general policies without any restrictions. Obviously, even for such small examples considerable reductions (especially for the non-preemptive problem) are possible.

Next, we consider the performance of POPs for the case with rejection cost $y_1 = 0$ and $y_1 = 1{,}000$. Table 7.9 shows the performance of preemptive and Table 7.10 of non-preemptive policies. As a benchmark we have added the results for some priority policies, namely BD-GC-U, BD-GC-D (with RAN as tie breaker) and RAN (for details cf. Chap. 6). The results have been obtained from 10 simulation runs with a length of 500,000 project arrivals and a warm-up of 10,000 project arrivals. Variance has been reduced using *common random numbers* (CRN) such that the paired t-confidence intervals for any two policies do not contain 0 at a confidence level of 95 % (cf. Law [84]).

We observe that restricting the consideration to the class of POPs does not lead to a loss of optimality for $y_1 = 1{,}000$. This is in line with the observation made before that the optimal general policy is a POP. However, even though for $y_1 = 0$

Table 7.9 Performance of preemptive general and PO-policies

Policy class	Policy	y_1 0	1,000
General	Optimal	10.129	50.387
	BD-GC-U	12.109	64.469
	BD-GC-D	12.110	64.617
	RAN	11.856	68.914
PO	Optimal	10.302	50.387
	BD-GC-U	12.111	64.629
	BD-GC-D	12.118	64.594
	RAN	11.471	60.478

Table 7.10 Performance of non-preemptive general and PO-policies

Policy class	Policy	y_1 0	1,000
General	Optimal	10.399	52.382
	BD-GC-U	12.039	64.413
	BD-GC-D	12.034	64.465
	RAN	11.714	66.822
PO	Optimal	10.541	52.382
	BD-GC-U	12.008	63.973
	BD-GC-D	12.009	64.115
	RAN	11.462	60.456

the optimal general policy is no longer a POP, the optimal POP does not perform much worse (optimality gap is only 1.4 %) and still clearly outperforms the RBPs. Furthermore, we observe that RBPOPs have a slight advantage over general RBPs.

We will investigate the potential of POPs to reduce state space cardinalities and their performance for more general settings in Sect. 7.5.

7.4 Scheduling Using Approximate Dynamic Programming

7.4.1 Basic Idea

Despite the usage of policy classes, the state space of the scheduling problem may become very large. Even for the preemptive problem using POPs, the size of the state space is exponential in the number of activity types. Thus, for higher numbers of activity types or projects in the system (e.g. for open systems with unbounded $K^{\max}$) the computation of an optimal policy using value iteration or policy iteration becomes intractable. This phenomenon is known as *curse of dimensionality* (cf. Powell [105]).

A number of approaches, summarized under the notion of *approximate dynamic programming* (ADP) (cf. Bertsekas and Tsitsiklis [17] and Powell [105]), aim at remedying the curse of dimensionality by approximating the value function $h(s)$ by a function $\tilde{h}(s)$. While for an exact representation of $h(s)$ normally all states are required $\tilde{h}(s)$ typically has a much more compact representation.

The policy obtained from ADP may no longer be optimal but still perform very well.

At first, we present in Sect. 7.4.2, an approximation of the value function for the non-preemptive problem based on the value function of the preemptive problem. In Sect. 7.4.3, we present for open systems approximations of the value function based on linear function approximation. Finally, in Sect. 7.4.4 it is shown how an approximation for the non-preemptive problem can be obtained based on linear function approximation for the preemptive problem.

7.4.2 Approximation Based on the Preemptive Problem

Comparing the policies for the non-preemptive and the preemptive problem which have been obtained for the example in Sect. 7.1.3 we make two observations. Firstly, the state space for the non-preemptive problem is much larger than for the preemptive problem. Secondly, the structure of the policies as well as the objective function values are very similar.

The observations can be explained as follows. If there are many projects in the system the number of waiting activities is much higher than the number of activities in process. Thus, the impact of the fact whether a single activity is in process or waiting becomes smaller while the impact of waiting times due to other activities that are still waiting becomes more prominent.

This suggests using the value function of the preemptive problem without consideration of activities in process as an approximation for the value function of the non-preemptive problem.

After determining an optimal policy for the preemptive problem we have the value function $h^{\mathrm{P}*}(s)\ \forall s \in \mathcal{S}^{\mathrm{P}}$ as well as the average cost $g^{\mathrm{P}*}$ under the optimal policy $\pi^{\mathrm{P}*}(s)\ \forall s \in \mathcal{S}^{\mathrm{P}}$.

However, $h^{\mathrm{P}*}(s)$ cannot be applied directly as an approximation since it is not defined for states $s' \in \mathcal{S}$ where for some $\sigma \in \Sigma(p, s')$ we have $\mathcal{E}(\sigma(s')) \neq \emptyset$.

Hence, it is necessary to map the states s' onto states in $s \in \mathcal{S}^{\mathrm{P}}$ by $s^{\mathrm{P}}(s')$ as defined in (7.32). Then, the approximation $\tilde{h}^{\mathrm{P}}(s)\ \forall s \in \mathcal{S}$ is given by

$$\tilde{h}^{\mathrm{P}}(s) = h^{\mathrm{P}*}(s^{\mathrm{P}}(s))\ \forall s \in \mathcal{S} \tag{7.71}$$

The following lemma gives two results. Firstly, it states that $\tilde{h}^{\mathrm{P}}(s)$ is defined for all $s \in \mathcal{S}$. Secondly, it states that we may also use the value function obtained from the preemptive problem if the class POPs is considered, with $\mathcal{S}^{\mathrm{PPO}} \subseteq \mathcal{S}^{\mathrm{P}}$. However,

we have to follow a POP for the non-preemptive problem as well, such that for all $s \in \mathcal{S}^{\mathrm{PO}}$ we have all subsequent states $s' \in \mathcal{S}^{\mathrm{PO}}$.

Lemma 7.4.1. *(a) For all $s \in \mathcal{S}$ we have $s^P(s) \in \mathcal{S}^P$.*
(b) For all $s \in \mathcal{S}^{PO}$ we have $s^P(s) \in \mathcal{S}^{PPO}$

Proof. Part (a) follows directly from Theorem 7.1.4 by noting that $K(s^{\mathrm{P}}) = K(s)$.

For the proof of part (b) we use from Theorem 7.3.4 that for a state $s \in \mathcal{S}^{\mathrm{PO}}$ and project states $\sigma_1, \sigma_2 \in \Sigma(p, s)$ we must have $\sigma_1 \succ \sigma_2$ or $\sigma_2 \succ \sigma_1$. W.l.o.g. we assume $\sigma_1 \succ \sigma_2$. From Definition 7.3.3, we know that $\mathcal{U}(\sigma_1) \subseteq \mathcal{U}(\sigma_2)$. As σ^{P} does not have any effect on $\mathcal{U}$ we know that if $\mathcal{U}(\sigma_1) = \mathcal{U}(\sigma_2)$ we have that $\sigma^{\mathrm{P}}(\sigma_1) = \sigma^{\mathrm{P}}(\sigma_2)$. Hence, both project states are mapped onto the same project state by $s^{\mathrm{P}}(s)$. If $\mathcal{U}(\sigma_1) \subset \mathcal{U}(\sigma_2)$ we have $\sigma^{\mathrm{P}}(\sigma_1) \succ \sigma^{\mathrm{P}}(\sigma_1)$. From Theorem 7.3.1 part (b), we know that, by the existence of the order on the project states in $\Sigma(p, s^{\mathrm{P}}(s))\ \forall p \in \mathcal{P}$, we must have $s^{\mathrm{P}}(s) \in \mathcal{S}^{\mathrm{PPO}}$. □

From Lemma 7.4.1, we directly obtain that the state space cardinality for the preemptive problem is a lower bound of the state space cardinality for the non-preemptive problem such that $|\mathcal{S}^{\mathrm{P}}| \leq |\mathcal{S}|$ and $|\mathcal{S}^{\mathrm{PPO}}| \leq |\mathcal{S}^{\mathrm{PO}}|$.

Then, we use the approximation to obtain an approximate policy π_{ADPP} for the non-preemptive problem in

$$\pi_{\mathrm{ADPP}}(s) = \mathrm{argmin}_{a \in \mathcal{A}(s)} \left\{ \frac{c(s) - g^{\mathrm{P}*}}{\beta(s,a)} + \sum_{s' \in \mathcal{S}} q(s'|s,a)(k(s,a,s') + \tilde{h}^{\mathrm{P}}(s)) \right\} \quad \forall s \in \mathcal{S} \quad (7.72)$$

where the determination of the optimal decision can be done in an efficient way.

We observe that mapping of successor states onto states in $\mathcal{S}^{\mathrm{P}}$ is done by shifting activities in process back to the sets of waiting activities. This allows the interpretation that for the approximation a decision $a \in \mathcal{A}(s)$ is selected that is optimal if, just after the decision, preemptions were allowed for the rest of the infinite planning horizon. Figure 7.8 illustrates the idea. The only difference between the policy based on the approximation and the optimal policy for the preemptive problem is the fact that in state $s \in \mathcal{S}$ we may not preempt activities already in process. Thus, the expression

$$\left\{ \frac{c(s) - g^{\mathrm{P}*}}{\beta(s,a)} + \sum_{s' \in \mathcal{S}} q(s'|s,a)(k(s,a,s') + \tilde{h}^{\mathrm{P}}(s)) \right\}$$

gives the long term relative cost for the preemptive problem after making decision a in state $s \in \mathcal{S}$.

For determining the optimal decision a^*, we may use the simplified representation of a decision as used in Sect. 7.1.2.2 for the preemptive problem. Thus, we

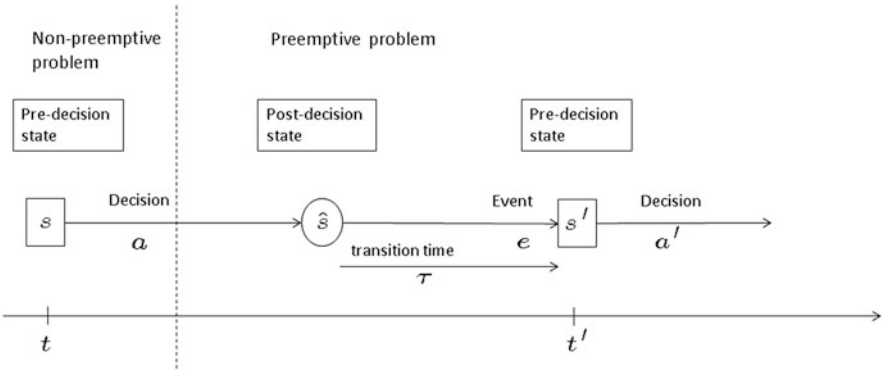

Fig. 7.8 Interpretation of a decision when using the value function of the preemptive problem as an approximation

have the opportunity of applying essentially the same procedure as discussed in Sect. 7.1.2.3. However, in addition, it must be taken into account that activities in process may not be preempted in state s such that we have to modify slightly the formulation of the LP and the solution procedure.

$$\min \left(\sum_{\sigma \in \Sigma^{\mathrm{S}}} \sum_{i \in \mathcal{W}(\sigma)} n(i,\sigma) Q(s,i,\sigma) \right) \tag{7.73}$$

where

$$Q(s,i,\sigma) = \mu_{ip(\sigma)} \left(h^{\mathrm{P}*}(s^{\mathrm{CP}}(s^{\mathrm{P}}(s),i,\sigma)) - h^{\mathrm{P}*}(s^{\mathrm{P}}(s)) \right) \tag{7.74}$$

The minimization is due to scarce resource types subject to the following constraints:

$$\sum_{\sigma \in \Sigma^{\mathrm{S}}} \sum_{i \in \mathcal{W}(\sigma)} I\left(r(i,p(\sigma)) = r\right) n(i,\sigma) \leq f(r,s) \;\; \forall r \in \mathcal{R} \tag{7.75}$$

$$n(i,\sigma) \leq n(\sigma,s) \quad \forall \sigma \in \Sigma, i \in \mathcal{W}(\sigma) \tag{7.76}$$

$$n(i,\sigma) \in \mathbb{N}_0 \quad \forall \sigma \in \Sigma, i \in \mathcal{W}(\sigma) \tag{7.77}$$

Here, we have replaced c_r by $f(r,s)$ in order to take into account activities in process in state s. As the mappings s^{CP} require a state in $\mathcal{S}^{\mathrm{P}}$ we have to use $s^{\mathrm{P}}(s)$.

We break ties between activity groups (i,σ) as follows.

1. Project type $p(\sigma)$.
2. Set of waiting activities $\mathcal{W}(\sigma)$.
3. Set of activities in process $\mathcal{E}(\sigma)$.
4. Index i.

Table 7.11 Performance of policies using the approximation from the preemptive problem

Policy class	Policy	Rejection cost 0	1,000
General	Optimal	10.399	52.382
	ADP-P	10.497	52.660
	RAN	11.856	68.914
PO	Optimal	10.541	52.382
	ADP-P-PO	10.587	52.464
	RAN	11.462	60.456

Table 7.12 Reduction of the state space cardinality when using the preemptive problem for an approximation

Policy class	Non-preemptive	Preemptive	%
General	683,209	53,130	7.78
PO	102,838	19,481	18.94

In order to be able to compare policies based on the approximation with optimal policies for the non-preemptive problem we order project states lexicographically in the same way as we order project states in Sect. 7.1.1.1 for the generation of decision alternatives. Then, in order to a derive a decision for non-preemptive problem ties between activities from group (i, σ) are broken using the project numbers $k = 1, \ldots, K(s)$. Projects having smaller numbers k with $\sigma(k, s)$ are preferred.

To show that the idea works we have applied the approach to the example from Sect. 7.1.3. Table 7.11 shows the results for the general version (ADP-P) of the policy obtained from the preemptive problem as well as its PO-version (ADP-P-PO). The results have been obtained from simulation with the same settings as in Sect. 7.3.4. As a benchmark, we have added the results for RAN which has performed best among the priority policies tested. We observe that ADP-P as well ADP-P-PO are near optimal for $y_0 = 0$ as well as $y_1 = 1{,}000$ and clearly outperform RAN. Furthermore, ADP-P-PO slightly outperforms ADP-P for $y_1 = 1{,}000$ which can be explained by the fact that non-preemptive POPs take into account by definition activities in process. Thus, a part of the suboptimal decisions are excluded before decisions are made using the approximative value function $\tilde{h}^{\mathrm{P}}(s)$.

Next, we briefly consider the reduction of the state space cardinality if the preemptive problem is used to obtain an approximation. Table 7.12 shows the cardinalities for general policies and POPs in the non-preemptive and the preemptive case. The column % shows the fraction of the state space cardinality for the preemptive problem from the state space cardinality for the non-preemptive problem. We conclude that considerable reductions of the state space cardinality at only little loss of performance are possible. Combining the idea of POPs with the approximation based on the preemptive problem we can reduce a state space of 683,209 states to a state space of only 19,481 states which corresponds to a fraction of only 2.85 %.

The performance of policies based on the approximation from the preemptive problem will be investigated more systematically in Sect. 7.5.

7.4.3 Approximation Using Linear Function Approximation

The approximation based on the solution of the preemptive problem has three drawbacks. Firstly, we still need to store a large number of states. Secondly, it does not alleviate the curse of dimensionality for the preemptive problem. Finally, it does not allow to determine policies for open systems having infinite state spaces.

Hence, we consider the option to approximate the value function using an *approximation architecture* (cf. Bertsekas and Tsitsiklis [17]) being a function that has a compact parameterized representation. In addition to *non-linear architectures* such as neural networks, *linear architectures* are common. For our purposes, we use a linear architecture $\tilde{h}^{\text{Lin}}(s, \mathbf{v})$ of the form

$$\tilde{h}^{\text{Lin}}(s, \mathbf{v}) = v_0 + \sum_{m=1}^{M} v_m \cdot \phi_m(s) \tag{7.78}$$

where $\mathbf{v}$ is a vector of parameters. Note that the linear approximation architecture corresponds essentially to a linear regression model. $\phi_m(s)$ are functions depending on state variables which are denoted as *basis functions* or *features*. The basis functions are weighted using the parameters v_m such that the value function $h(s)$ is approximated. M is the number of basis functions used for the approximation.

In order to obtain a value function approximation leading to a good scheduling policy, we have to adress two issues.

1. Selection of basis functions.
2. Determination of weights.

The selection of appropriate basis functions is problem specific and requires knowledge of the problem structure. We address this issue for our problem in Sect. 7.4.3.1.

For the determination of the weights there are different kinds of approaches available. *Simulation-based approaches* (cf. Bertsekas [16], Bertsekas and Tsitsiklis [17] or Powell [105]) essentially collect observations from simulation runs in order to determine approximations of the value function. However, we have found in preliminary experiments that due to the high variation of the system variables determining estimates of the weights may be very time consuming while it is not clear whether the approximations obtained lead to good policies. This is in line with the observations made by Meyn [93] for small queueing networks.

As we intend to consider more complex examples we have concentrated on a number of ADP-approaches do not need simulation.

A straight forward approach presented in Sect. 7.4.3.2 that is also useful to test approximation architectures is to approximate an open system via a semi-open system with finite $K^{\max}$. As a first step, we determine for the semi-open system an optimal policy. Afterwards, we apply linear regression on parts of the state space. As a result, hopefully a good policy is obtained which generalizes to the case of unlimited numbers of projects.

As for larger numbers of activity types and project types the approach is no longer tractable we have also tested approaches that do not require the value function of an optimal policy. A method based on the minimization of the Bellman error (cf. Bertsekas and Tsitsiklis [17]) turned out to be a good choice. A detailed description of the method will be given in Sect. 7.4.3.3.

Another class of approaches that are not based on simulation is *approximate linear programming* (ALP) (cf. De Farias and Van Roy [34, 35] and Veach [130]). However, as first tests delivered less promising results, in terms of performance of the policies obtained, we have dropped ALP from further consideration.

7.4.3.1 Selection of Basis Functions

For open queueing networks, approximation architectures based on polynomials defined on the queue lengths and number of projects in process have successfully been applied in many cases (cf. Koole and Pot [77], Roubos and Bhulai [114] or Vyzas [133]). From fluid approximations of queueing networks it is known that the value function may have a piece-wise quadratic structure (cf. Meyn [93]). This suggests two kinds of architectures for our problem.

Architecture based on project states To obtain an architecture for our problem where parallel project networks are possible we observe that the representation based on project states can be seen as a generalization of a queueing network. For the case that the networks have no parallel paths ($OS = 1$) we have for any project state σ at most one activity waiting or in process (such that $\mathcal{W}(\sigma) \cup \mathcal{E}(\sigma) = \{i\}$ for some $i \in \mathcal{V}_p$). Hence, $n(\sigma, s)$ corresponds to the number of activities of type $(i, p(\sigma))$ which are waiting in a queue or which are in process.

This suggests using a polynomial approximation architecture based on the numbers $n(\sigma, s)$ of the projects in project states $\sigma \in \Sigma$ for a system state s. For our investigations, we consider architectures based on first order, second order and third order polynomials which are given by

$$\tilde{h}^{\mathrm{PSLin1}}(s, \mathbf{v}) = v_0 + \sum_{k=1}^{|\Sigma^S|} v_k \cdot n(\sigma_k, s) \tag{7.79}$$

$$\tilde{h}^{\mathrm{PSLin2}}(s, \mathbf{v}) = v_0 + \sum_{k=1}^{|\Sigma^S|} v_k \cdot n(\sigma_k, s) + \sum_{k=1}^{|\Sigma^S|} \sum_{l=k}^{|\Sigma^S|} v_{kl} \cdot n(\sigma_k, s) \cdot n(\sigma_l, s) \tag{7.80}$$

$$\begin{aligned} \tilde{h}^{\mathrm{PSLin3}}(s, \mathbf{v}) = {} & v_0 + \sum_{k=1}^{|\Sigma^S|} v_k \cdot n(\sigma_k, s) + \sum_{k=1}^{|\Sigma^S|} \sum_{l=k}^{|\Sigma^S|} v_{kl} \cdot n(\sigma_k, s) \cdot n(\sigma_l, s) + \\ & \sum_{k=1}^{|\Sigma^S|} \sum_{l=k}^{|\Sigma^S|} \sum_{m=l}^{|\Sigma^S|} v_{klm} \cdot n(\sigma_k, s) \cdot n(\sigma_l, s) \cdot n(\sigma_m, s) \end{aligned} \tag{7.81}$$

Architecture based on queue lengths for the activity types The equivalence between $\mathcal{S}^{\text{PPO}}$ to $\mathcal{S}^{\text{Q}}$ suggests architecture for the preemptive problem that are based on the queue lengths given by $n(i, p, s)$. For the representation, we order the activity types such that they are assigned an index k. Again, we consider architectures based on first, second and third order polynomials that, however, use $n(i, p, s)$ instead of $n(\sigma, s)$. The architectures are as follows.

$$\tilde{h}^{\text{QLin1}}(s, \mathbf{v}) = v_0 + \sum_{k=1}^{|\mathcal{V}|} v_k \cdot n(i_k, p_k, s) \tag{7.82}$$

$$\tilde{h}^{\text{QLin2}}(s, \mathbf{v}) = v_0 + \sum_{k=1}^{|\mathcal{V}|} v_k \cdot n(i_k, p_k, s) + \sum_{k=1}^{|\mathcal{V}|} \sum_{l=k}^{|\mathcal{V}|} v_{kl} \cdot n(i_k, p_k, s) \cdot n(i_l, p_l, s) \tag{7.83}$$

$$\begin{aligned}\tilde{h}^{\text{QLin3}}(s, \mathbf{v}) = v_0 &+ \sum_{k=1}^{|\mathcal{V}|} v_k \cdot n(i_k, p_k, s) + \sum_{k=1}^{|\mathcal{V}|} \sum_{l=k}^{|\mathcal{V}|} v_{kl} \cdot n(i_k, p_k, s) \cdot n(i_l, p_l, s) + \\ &\sum_{k=1}^{|\mathcal{V}|} \sum_{l=k}^{|\mathcal{V}|} \sum_{m=l}^{|\mathcal{V}|} v_{klm} \cdot n(i_k, p_k, s) \cdot n(i_l, p_l, s) \cdot n(i_m, p_m, s)\end{aligned} \tag{7.84}$$

The advantage of the architectures based on queue lengths is the fact that the number of basis functions may be much smaller. The number of project states may, depending on the network structure, grow exponentially in the number of activity types (cf. proof of Corollary 7.1.1) while the number of activity types remains constant.

Note that we will use the architectures primarily for deriving policies of open systems without limits on the number of projects. For semi-open systems with $K^{\max} < \infty$ the architectures might need additional basis functions in order to capture the effect of rejection cost at the boundary of the state space (cf. Sect. 7.1.3). Indications on the kind of additional basis functions can be found in Bhulai and Koole [18] who have derived closed expressions of the relative value function for an M/M/1-system with rejections of projects.

7.4.3.2 Semi-open System as an Approximation for the Open System

The idea is to approximate an open system using an semi-open system with $K^{\max} < \infty$ that exhibits a similar behavior as the open system (cf. Pot [104] or Vyzas [133]). Figure 7.9 outlines the general idea. For constructing a semi-open system we have two options. As a first option, we may set $K^{\max} < \infty$ to a high value where states with $K(s) > K^{\max}$ are very unlikely to be entered for the open system. Thus, the error due to truncation is kept small (cf. Pot [104]). However, a rather high value $K^{\max}$ might be needed such that the state space may grow very large.

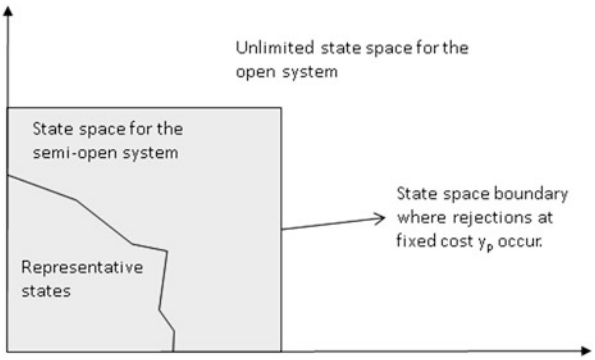

Fig. 7.9 General idea of using a semi-open system as an approximation for an open system

Thus, as a second option, we set $K^{\max}$ to a lower value and determine the values for the rejection cost y_p such that the relative value function $h(s)$ for the truncated problem is similar to the value function of the original problem (open system).

This can be done as follows. We first simulate the open system with unbounded $K^{\max}$ under some policy. Here, we use RAN which selects activities randomly. As a result, we obtain $g^{\text{o}}(\pi_{\text{RAN}})$ being the average cost under RAN for an open system.

In a second step, we set $y_p = y \ \forall p \in \mathcal{P}$ where y is determined such that for the semi-open system we obtain $g(\pi_{\text{RAN}}) \approx g^{\text{o}}(\pi_{\text{RAN}})$.

Next, we determine, for the semi-open system, an optimal policy π^* with average cost g^* using the standard methodologies discussed in Chap. 4. Finally, in order to obtain the weights for a linear approximation architecture $\tilde{h}^{\text{Lin}}(s, \mathbf{v})$, we apply linear regression (cf. Bertsekas and Tsitsiklis [17]) on a set of *representative states* $\tilde{\mathcal{S}} \subseteq \mathcal{S}$ where the following problem has to be solved.

$$\frac{1}{2} \min_{\mathbf{v}} \sum_{s \in \tilde{\mathcal{S}}} \gamma(s) \left(\tilde{h}^{\text{Lin}}(s, \mathbf{v}) - h(s) \right)^2 \tag{7.85}$$

The solution is given by

$$\mathbf{v}^* = \left(\mathbf{A}^T(\tilde{\mathcal{S}}) \text{diag}(\boldsymbol{\gamma}) \mathbf{A}(\tilde{\mathcal{S}}) \right)^{-1} \mathbf{A}^T(\tilde{\mathcal{S}}) \text{diag}(\boldsymbol{\gamma}) \mathbf{h} \tag{7.86}$$

where

$$\mathbf{A}(\tilde{\mathcal{S}}) = \begin{pmatrix} \phi_1(s_1) & \phi_2(s_1) & \dots & \phi_L(s_1) \\ \phi_1(s_2) & \phi_2(s_2) & \dots & \phi_L(s_2) \\ \vdots & \vdots & \vdots & \vdots \\ \phi_1(s_{|\tilde{\mathcal{S}}|}) & \phi_2(s_{|\tilde{\mathcal{S}}|}) & \dots & \phi_L(s_{|\tilde{\mathcal{S}}|}) \end{pmatrix} \tag{7.87}$$

Then, the approximate policy is obtained from

$$\pi_{\text{ADPLS}}(s) = \arg\min_{a \in \mathcal{A}(s)} \left\{ \frac{c(s) - g^*}{\beta(s,a)} + \sum_{s' \in \mathcal{S}} q(s'|s,a)(k(s,a,s') + \tilde{h}^{\text{Lin}}(s, \mathbf{v}^*)) \right\} \quad \forall s \in \mathcal{S} \quad (7.88)$$

If, in case of many basis functions, the matrix $\mathbf{A}(\tilde{\mathcal{S}})$ becomes very large a gradient descent approach similar to the approach proposed for the Bellman error method in Sect. 7.4.3.3 may be used (cf. Bertsekas and Tsitsiklis [17]). As it is often not clear which set of *representative states* results in an approximation leading to a good policy we test alternative sets of representative states $\tilde{\mathcal{S}}_1, \ldots, \tilde{\mathcal{S}}_{N^{\tilde{\mathcal{S}}}}$. As an additional option for tuning the linear regression, we take into account state relevance weights $\boldsymbol{\gamma}(s)$ that are given for the sets of representative states by the vectors $\boldsymbol{\gamma}_1, \ldots, \boldsymbol{\gamma}_{|N^{\tilde{\mathcal{S}}}|}$ (cf. De Farias and Van Roy [35]). The purpose of state relevance weights $\boldsymbol{\gamma}(s)$ is to control the precision of the approximation at the different states. If a state s a high state relevance weight, the weights for the basis functions of an architecture are determined such that the squared error $\left(\tilde{h}^{\text{Lin}}(s, \mathbf{v}) - h(s)\right)^2$ for the respective state becomes smaller.

Note that although an approximation might appear to be a good fit the resulting policy might not perform well. Unfortunately, from a theoretical point of view, the relationship between the choice of representative states, state relevance weights and performance of the policy based on the architecture is often not clear (cf. Roubos and Bhulai [113] or Pot [104]). Thus, we test alternative sets of representative states that may be randomly generated and select the set (with the corresponding state relevance weights) that delivers the best performing policy.

Basically, we can apply linear regression on the entire state space of the truncated problem such that $\tilde{\mathcal{S}} = \mathcal{S}$. However, approximation errors which are likely to occur near the boundary of the state space may impair the quality of the value function approximation. Hence, according to our experience, $\tilde{\mathcal{S}}$ should only contain states that are reasonably distant from the boundary of the state space. We discuss the determination of $\tilde{\mathcal{S}}$ in Sect. 7.4.3.4 in more detail.

The entire procedure is given by Algorithm 4 (denoted as ADP-LS). We perform a search over the sets $\tilde{\mathcal{S}}_1, \ldots, \tilde{\mathcal{S}}_{N^{\tilde{\mathcal{S}}}}$ and store in $\mathbf{v}^{\min}$ the weights of the approximation that results in the best policy in terms of the minimum average cost. The evaluation in Step 8 can be done in two ways.

1. Any methodology for the solution of the evaluation equations such as VI may be used (cf. Chap. 4).
2. Simulation run of sufficient length. This method has the advantage that the evaluation may be based on the performance for the open system. In order to reduce the needed length of the simulation run *common random numbers* (CRN) as a variance reduction technique (cf. Law [84]) has been found to be effective. The quality of the estimate for the average cost g is less important as only the best set of representative states is to be identified.

Algorithm 4 ADP-LS

Require: $\mathcal{S}$ of a truncated problem with given $K^{\max}$, $\tilde{\mathcal{S}}_1, \ldots, \tilde{\mathcal{S}}_{N^{\tilde{\mathcal{S}}}}, \boldsymbol{\gamma}_1, \ldots, \boldsymbol{\gamma}_{N^{\tilde{\mathcal{S}}}}$.

1: Determine optimal policy π^* and optimal $h(s)$ $\forall s \in \mathcal{S}$.
2: $g_{\text{ADPLS}} \leftarrow \infty$
3: $z \leftarrow 0$
4: **Do**
5: $z \leftarrow z + 1$
6: $\tilde{\mathcal{S}} \leftarrow \tilde{\mathcal{S}}_z$
7: Determine $\mathbf{v}$ by solving (7.85).
8: Determine g for the policy obtained from (7.88).
9: **if** $g < g^{\min}$ **then**
10: $\mathbf{v}(\pi^*) \leftarrow \mathbf{v}$
11: $g^{\min} \leftarrow g$
12: **end if**
13: **Until** $z = N^{\tilde{\mathcal{S}}}$

In order to test whether the approximation architectures capture sufficient structure of the value function such that the principles of dynamic programming can be applied we have also implemented an approximate version of policy iteration (cf. Sect. 4.6.2). The main modification is that the policy improvement in Step 12 is replaced by the computation of the approximation $\tilde{h}^{\text{Lin}}(s, \mathbf{v}(\pi_{n-1}))$ obtained for the policy π_{n-1} from the last iteration such that the new policy is given by

$$\pi_n(s) = \underset{a \in \mathcal{A}(s)}{\operatorname{argmin}} \left\{ \frac{c(s) - g(\pi_{n-1})}{\beta(s,a)} + \sum_{s' \in \mathcal{S}} q(s'|s,a)(k(s,a,s') + \tilde{h}^{\text{Lin}}(s, \mathbf{v}(\pi_{n-1}))) \right\} \quad \forall s \in \mathcal{S} \tag{7.89}$$

where $\mathbf{v}(\pi_{n-1})$ is the vector of weights obtained from the value function $h(s, \pi_{n-1})$ of policy π_{n-1}. As we use an approximation, convergence towards an optimal policy is no longer guaranteed (cf. Bertsekas and Tstsiklis [17]). Hence, we stop after a fixed number of iterations and store the approximation that has yielded the best policy so far.

To find a good approximation $\tilde{h}^{\text{Lin}}(s, \mathbf{v}(\pi_{n-1}))$ we search again over a sets $\tilde{\mathcal{S}}_1, \ldots, \tilde{\mathcal{S}}_{N^{\tilde{\mathcal{S}}}}$ of representative states having vectors of state relevance weights $\boldsymbol{\gamma}_1, \ldots, \boldsymbol{\gamma}_{N^{\tilde{\mathcal{S}}}}$. Note that if simulation is used for finding the best set of representative states CRN may be used to keep simulation runs short. However, for the average cost g to be used in (7.89) an additional longer simulation run should be used to obtain a better estimate. The procedure is given by Algorithm 5 (denoted as ADP-PI-LS). Finally, we briefly address a common issue of linear regression, namely lack of *multicollinearity*. One possibility where this issue occurs is when some basis functions may not yield different values on a set of representative states. Such cases happen for example if a combination of project states never occurs in the

Algorithm 5 ADP-PI-LS

Require: $\mathcal{S}$ of a truncated problem with given $K^{\max}$, $\tilde{\mathcal{S}}_1, \ldots, \tilde{\mathcal{S}}_{N^{\tilde{\mathcal{S}}}}$, $\boldsymbol{\gamma}_1, \ldots, \boldsymbol{\gamma}_{N^{\tilde{\mathcal{S}}}}$, π_0.

$n \leftarrow 0$.

Policy evaluation: Determine $g(\pi_0)$ $h(\pi_0, s)$ $\forall s \in \mathcal{S}$ by solving the equations:

$$h(\pi_0, s) = \frac{c(s) - g(\pi_0)}{\beta(s, \pi_0(s))} + \sum_{s' \in \mathcal{S}} q(s'|s, \pi_0(s))(k(s, \pi_0(s), s') + h(\pi_0, s')) \quad \forall s \in \mathcal{S}$$

$g(\pi^{\min}) \leftarrow \infty$

Do

  $n \leftarrow n + 1$.

  $g(\pi_n) \leftarrow \infty$

  $z \leftarrow 0$

  Do

    $z \leftarrow z + 1$

    $\tilde{\mathcal{S}} \leftarrow \tilde{\mathcal{S}}_z$

    Determine $\mathbf{v}(\pi_{n-1})$ by solving (7.85).

    Determine g for the policy obtained from

$$\pi_n(s) = \underset{a \in \mathcal{A}(s)}{\operatorname{argmin}} \left\{ \frac{c(s) - g(\pi_{n-1})}{\beta(s, a)} + \sum_{s' \in \mathcal{S}} q(s'|s, a)(k(s, a, s') + \tilde{h}^{\text{Lin}}(s, \mathbf{v}(\pi_{n-1}))) \right\} \quad \forall s \in \mathcal{S}$$

    if $g < g(\pi_n)$ **then**

      $\mathbf{v}(\pi_{n-1}) \leftarrow \mathbf{v}$

      $g(\pi_n) \leftarrow g$

    end if

  Until $z = N^{\tilde{\mathcal{S}}}$

  Policy evaluation: Obtain $h(\pi_n, s)$ $\forall s \in \mathcal{S}$ by solving the equations

$$h(\pi_n, s) = \frac{c(s) - g(\pi_n)}{\beta(s, \pi_n(s))} + \sum_{s' \in \mathcal{S}} q(s'|s, \pi_n(s))(k(s, \pi_n(s), s') + h(\pi_n, s')) \quad \forall s \in \mathcal{S}$$

  if $g(\pi_n) < g(\pi^{\min})$ **then**

    $g(\pi^{\min}) \leftarrow g(\pi_n)$.

    $\mathbf{v}(\pi^{\min}) \leftarrow \mathbf{v}(\pi_{n-1})$.

  end if

Until $n = N^{\text{ADPPILS}}$

set of representative states such that the basis function delivers always 0. Then, it is not possible to determine a weight. To resolve this issue, we eliminate such basis functions from the architecture. All other cases where lack of multicollinearity occurs are resolved by sampling a new set of representative states.

7.4.3.3 Bellman Error Minimization

A drawback of the methods discussed in Sect. 7.4.3.2 is the fact that the state space, even for a semi-open system, might become too large for applying the methodologies discussed so far. Hence, we consider another methodology (cf. Bertsekas and Tsitsiklis [17]) for determining $\mathbf{v}$ given a policy π.

We rearrange the evaluation equation (4.12) by bringing all terms to the left hand side in order to obtain the following condition to be met by the value function $h(s)\ \forall s \in \mathcal{S}$ for any policy π

$$\frac{c(s) - g(\pi)}{\beta(s, \pi(s))} + \sum_{s' \in \mathcal{S}} q(s'|s, \pi(s))(k(s, \pi_n(s), s') + h(s')) - h(s) = 0 \quad \forall s \in \mathcal{S} \tag{7.90}$$

The condition needs no longer be met if an approximation $\tilde{h}^{\text{Lin}}(s, \mathbf{v})$ is used instead of $h(s)$.

Thus, we define the Bellman error $D(s, \mathbf{v}, \pi)$ as the violation of condition (7.90).

$$D(s, \mathbf{v}, \pi) = \frac{c(s) - g(\pi)}{\beta(s, \pi(s))} + \sum_{s' \in \mathcal{S}} q(s'|s, \pi(s))(k(s, \pi_n(s), s') + \tilde{h}^{\text{Lin}}(s', \mathbf{v})) - \tilde{h}^{\text{Lin}}(s, \mathbf{v}) \tag{7.91}$$

Now, we see that, as an alternative to linear regression, we can determine $\mathbf{v}$ such that the Bellman errors for a set of representative states $\tilde{\mathcal{S}}$ are minimized. This can done by solving the following problem.

$$\min_{\mathbf{v}} \sum_{s \in \tilde{\mathcal{S}}} \gamma(s) D(s, \mathbf{v}, \pi)^2 \tag{7.92}$$

Analogously to linear regression, where the weighted squared errors between the true value function and the approximation are minimized, we use squared Bellman errors which again are weighted using state relevance weights $\gamma(s)$. For the solution we have considered two options (cf. Bertsekas and Tsitsiklis [17]).

1. Single step of *Newton's-method* as given by

$$\mathbf{v} = \left(\nabla^2 f(\mathbf{v}_0)\right)^{-1} \nabla f(\mathbf{v}_0) \tag{7.93}$$

where $f(\mathbf{v}) = \sum_{s \in \tilde{\mathcal{S}}} \gamma(s) D(s, \mathbf{v}, \pi)^2$.

Generally, Newton's method applied for optimization iteratively approaches the optimum of a function $f(\mathbf{v})$ starting from some given point $\mathbf{v}_0$ by minimizing in each iteration a quadratic approximation. As the function is already quadratic a single step is sufficient. We have initialized the algorithm by $\mathbf{v}_0 = \mathbf{0}$. Note that

the step of Newton's method is equivalent to a step of the Gauss-Newton method because $D(s, \mathbf{v}, \pi)$ is a linear function of $\mathbf{v}$.

2. Gradient descent approach. Here, we apply a *steepest-descent approach* with *diagonal scaling* for accelerating convergence. At first, we initialize the algorithm with $\mathbf{v}_0(s) = \mathbf{0}$. Then, we compute a diagonal matrix $\mathbf{D}$ having $\left(\frac{\partial^2 f(v_{0i})}{\partial^2 v_{0i}}\right)^{-1}$ at the diagonal. Note that for computing the matrix the choice of $\mathbf{v}_0$ does not have any impact as the second derivatives of the quadratic function $f(\mathbf{v})$ are constant. Now, we use the following recursion until convergence

$$\mathbf{v}_{n+1} = \mathbf{v}_n - \xi \cdot \mathbf{D} \nabla f(\mathbf{v}_n) \tag{7.94}$$

ξ denotes a step size for which a value of 1 has worked well.

The gradient descent approach has the advantage that the Hessian matrix $\nabla^2 f(\mathbf{v}_0)$ and its inverse being a $M \times M$-matrix needs not be computed. Thus the gradient descent approach is a good choice for a large number M of basis functions. For the diagonal matrix, we only have to store the values at the diagonal.

The minimization of the Bellman error can be embedded into approximate policy iteration that is similar to Algorithm 5. Algorithm 6 gives the entire procedure.

The major difference to Algorithm 5 is the computation of the vector $\mathbf{v}(\pi_{n-1})$ related to policy π_{n-1} from the previous iteration. We observe that for the improvement of the policy, the average cost $g(\pi_n)$ is not determined by Bellman error minimization. Instead, it may be obtained from the solution of evaluation equations or in a simulation run (cf. De Farias and Van Roy [34]). Thus, the approach can also be applied to state spaces being too large (even for semi-open systems) for the approaches considered in Sect. 7.4.3.2. For the search over the sets of representative states, we may apply CRN for short simulation runs. However, we recommend again to use an additional longer run to obtain a better estimate for $g(\pi_n)$ after having determined the best set of representative states.

Finally, we remark that the Bellman error minimization does not allow to use the constant v_0 in the approximation architectures for the following reason. When we replace in (7.95) $\tilde{h}^{\text{Lin}}(s, \mathbf{v})$ by linear function as given by (7.78) we obtain

$$D(s, \mathbf{v}, \pi) = \frac{c(s) - g(\pi)}{\beta(s, \pi(s))}$$
$$+ \sum_{s' \in \mathcal{S}} q(s'|s, \pi(s))(k(s, \pi_n(s), s') + v_0 + \sum_{m=1}^{N} v_m \cdot \phi_m(s')) - v_0 - \sum_{m=1}^{N} v_m \cdot \phi_m(s) \tag{7.95}$$

From $\sum_{s' \in \mathcal{S}} q(s'|s, \pi(s)) v_0 - v_0 = v_0 - v_0 = 0$ as $\sum_{s' \in \mathcal{S}} q(s'|s, \pi(s)) = 1$ v_0 is eliminated. Thus, v_0 is not determined such that we simply set $v_0 = 0$.

Algorithm 6 ADP-PI-BE

Require: $\tilde{\mathcal{S}}_1, \ldots, \tilde{\mathcal{S}}_{N^{\tilde{\mathcal{S}}}}, \boldsymbol{\gamma}_1, \ldots, \boldsymbol{\gamma}_{N^{\tilde{\mathcal{S}}}}, \pi_0$.

$n \leftarrow 0$.
Policy evaluation: Determine the average cost $g(\pi_0)$.
$\pi^{\min} = \infty$
Do
  $n \leftarrow n + 1$.
  $z \leftarrow 0$
  $g(\pi_n) \leftarrow \infty$
  Do
    $z \leftarrow z + 1$
    $\tilde{\mathcal{S}} \leftarrow \tilde{\mathcal{S}}_z$
    Determine $\mathbf{v}(\pi_{n-1})$ by solving the problem stated in (7.92) for $\tilde{\mathcal{S}}_z$.
    Determine g for the policy obtained from

$$\pi_n(s) = \underset{a \in \mathcal{A}(s)}{\operatorname{argmin}} \left\{ \frac{c(s) - g(\pi_{n-1})}{\beta(s,a)} + \sum_{s' \in \mathcal{S}} q(s'|s,a)(k(s,a,s') + \tilde{h}^{\mathrm{Lin}}(s, \mathbf{v}(\pi_{n-1}))) \right\} \quad \forall s \in \mathcal{S}$$

    if $g < g(\pi_n)$ **then**
      $\mathbf{v}(\pi_{n-1}) \leftarrow \mathbf{v}$
      $g(\pi_n) \leftarrow g$
    end if
  Until $z = N^{\tilde{\mathcal{S}}}$
  if $g(\pi_n) < g(\pi^{\min})$ **then**
    $g(\pi^{\min}) \leftarrow g(\pi_n)$
    $\mathbf{v}(\pi^{\min}) \leftarrow \mathbf{v}(\pi_{n-1})$
  end if
Until $n = N^{\mathrm{ADPPIBE}}$

The lack of multicollinearity may also occur for the Bellman error minimization which is resolved in the same way as for linear regression.

7.4.3.4 Determining Sets of Representative States

To control the composition of a set of representative states $\tilde{\mathcal{S}}$ we specify an interval $[K^{\mathrm{L}}, K^{\mathrm{U}}]$. Thus, for any state $s \in \mathcal{S}$ to be considered we must have $K(s) \in \left[K^{\mathrm{L}}, K^{\mathrm{U}}\right]$. We have determined sets of representative states in two ways.

1. Select all states $s \in \mathcal{S}$ with $K(s) \in \left[K^{\mathrm{L}}, K^{\mathrm{U}}\right]$. Then, we have

$$\tilde{\mathcal{S}} = \left\{ s \in \mathcal{S} | K(s) \in [K^{\mathrm{L}}, K^{\mathrm{U}}] \right\}$$

2. Sampling of states on a simulation run where RAN is used for scheduling and a maximum cardinality $|\tilde{\mathcal{S}}|^{\max}$ is specified such that $|\tilde{\mathcal{S}}| \leq |\tilde{\mathcal{S}}|^{\max}$. Then, we have $\tilde{\mathcal{S}} \subseteq \{s \in \mathcal{S} | K(s) \in [K^{\mathrm{L}}, K^{\mathrm{U}}]\}$. During the simulation run, states are collected until $|\tilde{\mathcal{S}}|^{\max}$ is reached. RAN is used in order access a large range of possible states as every possible decision may be selected in a state with non-zero probability.

7.4.4 *Approximation for the Non-preemptive Problem Based on Linear Function Approximation for the Preemptive Problem*

Instead of applying linear function approximation to the non-preemptive problem, we can use linear function approximation for the preemptive problem. This has the advantage that it is easier to determine an approximation for the following reasons. Firstly, the number of basis functions is lower for the PSLin-architectures as they depend on the number of project states. Secondly, we can use the QLin-architectures based on the queues for the activity types. Thirdly, it tends to be easier to find good sets of representative states due to a smaller state space.

The procedure consists of two steps.

1. Determine a linear function approximation $\tilde{h}^{\mathrm{LinP}}(s)$ for the preemptive problem.
2. Derive an approximation for the non-preemptive problem based on the approximation for the preemptive problem by using $\tilde{h}^{\mathrm{LinP}}(s)$ instead of $h^{\mathrm{P}*}(s)$ in the approximation $\tilde{h}^{\mathrm{P}}(s)$ given by (7.71). Note that for determining an optimal decision efficiently we may use $\tilde{h}^{\mathrm{LinP}}(s)$ instead of $h^{\mathrm{P}*}(s)$ in (7.74).

7.5 Computational Study

The computational study has the following main objectives. Firstly, the benefit of project state ordering policies (POPs) should be assessed where attention is paid to their performance and their potential to reduce state space cardinality. We refer to the class of stationary policies without restrictions (except the restrictions made in Chap. 2) to a certain class as *general*. Secondly, the focus is on the benefit of optimal policies relative to simple resource based priority policies (RBPs). As all priority policies considered are RBPs we simply refer to them as priority policies. When considering RBPs, we also consider the benefit of RBPOPs. As we only consider RBPs we refer to RBPOPs as PO-priority policies. Finally, we investigate the potential of using linear function approximation (with special focus to open systems) to obtain good policies.

At first, we outline in Sect. 7.5.1 the experimental design before we present the results for the preemptive problem in Sect. 7.5.4. In Sect. 7.5.5 we discuss the results

Table 7.13 System related parameters for the problem instances

Parameter	Value
$\|\mathcal{R}\|$	2,3
c_r	1
$(CV^{\overline{d},\min}, CV^{\overline{d},\max})$	[0; 0.2], [0.4; 0.6], [0.8; 1.0]
$\lambda^{\max}$	0.133
u_r	0.7, 0.9
$K^{\max}$	5,10

for the non-preemptive problem where also the performance of the approximation based on the preemptive problem is investigated. Section 7.5.6 is dedicated to an investigation of linear function approximation.

7.5.1 Experimental Design

For the investigation, we have used one set of problem instances with a single project type and one set with two project types. The problem instances are kept moderate in size as the focus is on the performance of POPs, state space cardinalities and on the benefit of optimal policies relative to priority policies for different ranges of problem parameters.

In the following, we describe, in detail, the experimental design. For a detailed description of the parameters and the generation procedure for the problem instances, we refer to Chap. 5. We consider a system with two resource types ($|\mathcal{R}| = 2$) and three resource types ($|\mathcal{R}| = 3$) with $c_r = 1 \ \forall r \in \mathcal{R}$. Thus, in the following, we refer to resources instead of resource types.

Furthermore, we control the coefficient of variation $CV_{\overline{d}_r}$ related to the expected durations of the activity types to be processed on a resource $r \in \mathcal{R}$ by the requirement that it lies in a specified interval $(CV^{\overline{d},\min}, CV^{\overline{d},\max})$ for each resource $r \in \mathcal{R}$. In our experimental design we have specified three intervals $[0; 0.2]$, $[0.4; 0.6]$ and $[0.8; 1.0]$ covering ranges of low, medium and high degrees of variation of the expected activity durations at each resource.

The maximum arrival rate is set to $\lambda^{\max} = 0.1333$ which is reduced accordingly to attain a specified level of utilization u which is set to 0.7 and 0.9 for both resources, respectively. The maximum number of projects in the system $K^{\max}$ is set to 5 and 10 projects. Table 7.13 summarizes the values for the system parameters.

Instances with a single project type We have set the holding cost to $w_1 = 1$ per time unit and the rejection cost to $y_1 = 1{,}000$. If rejection cost would be set to a lower value scheduling policies may exploit the bound $K^{\max}$ on the number of projects in order to reduce average holding cost by increasing the rejection rate (cf. Sect. 7.1.3). This is typically not a desired behavior of scheduling policies as acceptance/rejection of new projects is typically subject to order acceptance decisions on the tactical level (cf. Chap. 8).

Table 7.14 Project type related parameters for the problem instances with one project type

Parameter	Value
$(\lvert\mathcal{V}_{11}\rvert, \lvert\mathcal{V}_{12}\rvert)$ if $\lvert\mathcal{R}\rvert = 2$	(3, 2)
$(\lvert\mathcal{V}_{11}\rvert, \lvert\mathcal{V}_{12}\rvert, \lvert\mathcal{V}_{13}\rvert)$ if $\lvert\mathcal{R}\rvert = 3$	(2, 2, 1)
OS_1	0, 0.2, 0.4, 0.6, 0.8, 1.0
w_1	1
y_1	1,000
N^{PI}	5

Table 7.15 Project type related parameters for the problem instances with two project types

Parameter	Value
$(\lvert\mathcal{V}_{p1}\rvert, \lvert\mathcal{V}_{p2}\rvert)$,	(2, 1)
OS_p	0, 0.3, 0.6, 1.0
(w_1, w_2)	(1, 2)
y_p	1,000
N^{PI}	5
a_p	0.5

Furthermore, $N^{\mathrm{PI}} = 5$ samples are drawn for r_{i1} and $\overline{d}_{i1}$ for given problem parameters. The project type is composed of five activity types ($|\mathcal{V}_1| = 5$).

For the sets $\mathcal{V}_{1r}\ \forall r \in \mathcal{R}$ of activity types to be processed on the resource types, we specify the cardinalities to be $|\mathcal{V}_{11}| = 3$ and $|\mathcal{V}_{12}| = 2$ for the case of $|\mathcal{R} = 2|$. For the case of $|\mathcal{R}| = 3$ we have set $|\mathcal{V}_{11}| = 2$, $|\mathcal{V}_{12}| = 2$ and $|\mathcal{V}_{13}| = 1$.

The network of the project type is controlled via the order strength (OS) where OS_1 is set to 0, 0.2 0.4, 0.6, 0.8 and 1. For each value, we have generated one network sample. Table 7.14 summarizes the values for the set of instances.

Thus, the set comprises 720 instances resulting from the product of the levels w.r.t. $|\mathcal{R}|$ (2), $(CV^{\overline{d},\min}, CV^{\overline{d},\max})$ (3), u_r (2), $K^{\max}$ (2), OS (6) and N^{PI} (5).

Instances with two project types For this set of instances, we assume two project types where each project type $p \in \mathcal{P}$ is composed of three activity types ($|\mathcal{V}_p| = 3$). Both project types have a network, that has the same value OS_p which is set to 0, 0.3, 0.6 and 1. For each value of OS, we, again, generate one network sample. As there are two possible networks only for $OS_p = 0.6$, for all other values both project types have the same network.

The holding costs per time unit are set to $w_1 = 1$ and $w_2 = 2$ whereas the rejection costs are set to $y_1 = y_2 = 1{,}000$. Thus, the two project types differ for a problem instance w.r.t. holding costs per time unit and the samples for r_{ip} and $\overline{d}_{ip}$.

Finally, the total arrival rate is shared equally among the two project types such that we set the fractions of the total arrival rate to $a_1 = a_2 = 0.5$.

As both project types consist only of 3 activity types we consider only instances with $|\mathcal{R}| = 2$. Thus, for the sets $\mathcal{V}_{1r}\ \forall r \in \mathcal{R}$ of activity types to be processed on the resource types, we specify the cardinalities for both project types $p \in \mathcal{P}$ to be $|\mathcal{V}_{p1}| = 2$ and $|\mathcal{V}_{p2}| = 1$.

Table 7.15 summarizes the values for the two sets of instances where we have used p as index if both project types have the same parameter value in a problem instance.

Thus, the set comprises 240 instances resulting from the product of the levels w.r.t. $(CV^{\overline{d},\min}, CV^{\overline{d},\max})$ (3), u_r (2), $K^{\max}$ (2), OS (4) and N^{PI} (5).

7.5.2 Priority Policies

As a benchmark for optimal policies, we have considered priority policies. As simple priority policies, we consider the well known WSPT-policy (or $c\mu$-policy) and the RAN-policy where activities are randomly selected from the sets $\mathcal{W}(r,t)$ of activities waiting for a resource $r \in \mathcal{R}$. Furthermore, we have added two priority policies based on the *bottleneck dynamics* approach (cf. Lawrence and Morton [85] or Morton and Pentico [96]) that have performed well in the simulation study presented in Chap. 6. The policies are namely BD-GC-U and BD-GC-D (for details see Sect. 6.1) with a lookahead parameter $\kappa = 1$. Ties between activities with same priorities are broken randomly.

7.5.3 Simulation Setup

For the evaluation of optimal scheduling policies and priority policies, we have used simulation. For each policy and each problem instance, we have carried out 10 simulation runs with 30,000 project arrivals (where some may be rejected due to $K^{\max}$). The warm-up period has been set to 10,000 projects. In order to reduce variance between the results for different policies, we have used *common random numbers* (cf. Law [84]). We have verified that, for most cases, the resulting paired t-confidence intervals (cf. Law [84]) for any two policies do not contain 0 at a confidence level of 95 % and the confidence intervals for g are not larger than 10 % of the mean.

When applying scheduling policies we break ties between projects in the same state according to their arrival times.

7.5.4 Results for the Preemptive Problem

7.5.4.1 State Space Cardinalities

In order to bound computation times, we have defined a limit of 1,000,000 states for $|\mathcal{S}^{\mathrm{P}}|$ and $|\mathcal{S}^{\mathrm{PPO}}|$. If the limit is exceeded for a given problem instance the computation of an optimal policy (general or PO) is aborted.

Instances with a single project type Table 7.16 shows the cardinalities for the set of problem instances. As $|\mathcal{S}^{\mathrm{P}}|$ (and $|\mathcal{S}^{\mathrm{PPO}}|$ to a lesser extent) depend on $|\Sigma|$ the

Table 7.16 State space cardinalities for a single project type at different levels of *OS*

$K^{\max}$		5			10		
OS_1	$\lvert\Sigma^{\mathrm{P}}\rvert$	$\lvert\mathcal{S}^{\mathrm{P}}\rvert$	$\lvert\mathcal{S}^{\mathrm{PPO}}\rvert$	(%)	$\lvert\mathcal{S}^{\mathrm{P}}\rvert$	$\lvert\mathcal{S}^{\mathrm{PPO}}\rvert$	(%)
0	31	376,992	7,776	2.06	–	161,050	–
0.2	19	42,504	3,276	7.71	–	61,220	–
0.4	11	4,368	1,176	26.92	352,716	18,876	5.35
0.6	9	2,002	924	46.15	92,378	14,872	16.10
0.8	7	792	560	70.71	19,448	8,294	42.65
1	5	252	252	100.00	3,003	3,003	100.00

Table 7.17 State space cardinalities for two project types at different levels of *OS*

$K^{\max}$		5			10		
OS	$\lvert\Sigma^{\mathrm{P}}\rvert$	$\lvert\mathcal{S}^{\mathrm{P}}\rvert$	$\lvert\mathcal{S}^{\mathrm{PPO}}\rvert$	(%)	$\lvert\mathcal{S}^{\mathrm{P}}\rvert$	$\lvert\mathcal{S}^{\mathrm{PPO}}\rvert$	(%)
0	14	11,628	3,885	33.41	–	116,754	–
0.3	10	3,003	1,638	54.55	184,756	40,040	21.67
0.6	8	1,287	966	75.06	43,758	21,021	48.04
1	6	462	462	100.00	8,008	8,008	100.00

state space cardinalities implicitly depend on the network structure given by *OS* and the number of activity types $|\mathcal{V}_p|$. Furthermore, they depend on $K^{\max}$. Thus, the following tables show the cardinalities $|\mathcal{S}^{\mathrm{P}}|$ and $|\mathcal{S}^{\mathrm{PPO}}|$ for different values of *OS* and $K^{\max}$ when general policies and POPs are considered. The percentages indicate the fraction of the cardinality of $\mathcal{S}^{\mathrm{PPO}}$ from the cardinality of $\mathcal{S}^{\mathrm{P}}$.

The results confirm the theoretical finding that for small values of *OS* where networks have less precedence relations state spaces may become very large if general policies are considered. Furthermore, the correctness of Theorem 7.1.4 could be verified. For example, when $OS = 0$ and $K^{\max} = 5$ we obtain $376{,}992 = \binom{5+31}{31}$, and for $OS = 0.4$ and $K^{\max} = 10$ we obtain $352{,}716 = \binom{10+11}{11}$. We observe that the growth of the state space can be considerably tempered using POPs such that $|\mathcal{S}^{\mathrm{PPO}}|$ may only be a small fraction of $|\mathcal{S}^{\mathrm{P}}|$. We verify that the upper bound from Theorem 7.3.3 is tight as $7{,}776 = (5+1)^5$ for $OS = 0$ and $K^{\max} = 5$ and $161{,}050 = (10+1)^5$ for $OS = 0$ and $K^{\max} = 10$. Thus, we could solve all problem instances to optimality using POPs where using general policies the state was larger than our limit of 1,000,000 states for instances with $K^{\max} = 10$ and $OS \leq 0.2$ (Table 7.16).

Instances with two project types Table 7.17 confirms the observations made in Sect. 7.5.5.1 for the case with two project types. Again, we could determine an optimal POP for all instances.

Table 7.18 Relative performance of priority policies for a single project type at different levels of *OS*

OS_1	BD-GC-U(%)	BD-GC-D(%)	WSPT(%)	RAN(%)	Best(%)	Max best(%)
0	0.1	0.1	51.0	41.6	−0.1	1.2
0.2	1.2	1.2	44.9	40.1	1.1	19.7
0.4	4.5	4.4	33.8	32.0	3.8	24.8
0.6	4.4	4.4	28.2	26.9	3.1	25.9
0.8	4.7	4.6	22.6	26.1	3.3	20.3
1	11.5	11.3	17.4	21.2	5.0	25.9

Table 7.19 Relative performance of priority policies for a single project type at different levels of $CV_{\overline{d}_r}$

$CV_{\overline{d}_r}$	BD-GC-U(%)	BD-GC-D(%)	WSPT(%)	RAN(%)	Best(%)	Max best(%)
[0; 0.2]	3.5	3.4	56.3	39.8	2.9	18.2
[0.4; 0.6]	5.2	5.2	29.0	30.9	3.7	25.9
[0.8; 1.0]	4.5	4.4	13.7	22.0	1.6	19.7

7.5.4.2 Performance of Optimal Policies

At first, we could verify that the optimality gap between an optimal POP and an optimal general policy is zero for the instances where we could determine an optimal general policy. This indicates that the restriction to POPs does not lead to a loss of optimality for the problem instances considered.

Next, we report on the performance of optimal policies relative to priority policies.

Instances with a single project type Table 7.18 shows the mean optimality gap (given by $\frac{g(\pi^{\text{prio}})}{g(\pi^{*PO})} - 1$) of the priority policies to the optimal POP at different levels of the order strength (*OS*). The mean gap in the column *Best* is based on the best solution gap obtained for each problem instance. The column *Max Best* shows the maximum value for the best gap obtained within a set of problem instances with the same OS in order to demonstrate the range of possible best gaps (Table 7.18).

First of all, we observe that it is beneficial to search for optimal policies especially for *OS*-values larger than 0.2. For small values of *OS* the performance of the BD-policies is near optimal on average while for higher *OS*-values the best gaps are 3–5 % on average. However, much are higher gaps up to 25.9 % are possible. Table 7.19 shows the mean as well as the maximum gaps for different ranges of $CV_{\overline{d}_r}$. We observe that, for higher ranges, the benefit of optimal policies becomes smaller on average as the smaller mean best solution gap of 1.7 % indicates. This can be explained by the fact that WSPT becomes more beneficial for higher $CV_{\overline{d}_r}$ (Table 7.19).

Table 7.20 Relative performance of priority policies for two project types at different levels of *OS*

OS_p	BD-GC-U (%)	BD-GC-D (%)	WSPT (%)	RAN (%)	Best (%)	Max best (%)
0.0	2.0	2.0	23.9	28.5	1.7	8.1
0.3	12.4	2.5	18.6	27.9	2.3	12.7
0.6	12.8	3.3	17.0	25.0	3.0	11.1
1.0	27.0	3.8	12.2	26.4	3.7	17.1

Table 7.21 Relative performance of priority policies for two project types at different levels of CV_d

$CV_{\overline{d}_r}$	BD-GC-U (%)	BD-GC-D (%)	WSPT (%)	RAN (%)	Best (%)	Max best (%)
[0; 0.2]	5.4	4.9	17.7	25.6	4.8	17.1
[0.4; 0.6]	10.6	3.0	23.7	29.2	3.0	11.5
[0.8; 1.0]	24.4	0.7	12.3	26.1	0.5	2.5

Instances with two project types Tables 7.20 and 7.21 confirm that the observations made for a single project type are also valid for multiple project types (Tables 7.20 and 7.21).

7.5.4.3 Performance of Project State Ordering Priority Policies

In the following, we discuss the results of PO-priority policies for the case with a single project type as well as for the case with two project types.

Instances with a single project type Table 7.22 shows the mean optimality gap of general priority policies and that of PO-priority policies for the instances with a single project type.

We observe that BD-GC-U based on the bottleneck dynamics approach shows an inferior performance on average. By contrast, BD-GC-D, WSPT and the RAN exhibit, on average, an improved performance.

An explanation is the fact that, as the general priority policies BD-GC-U and BD-GC-D prefer projects with a smaller expected remaining workload they already behave in many cases like POPs. Recall that for any two project states $\sigma_1, \sigma_2 \in \Sigma$ with $\sigma_1 \succ_a \sigma_2$ we always have $\mathcal{U}(\sigma_1) \subset \mathcal{U}(\sigma_2)$ such that $\sum_{i \in \mathcal{U}(\sigma_1)} \overline{d}_{ip(\sigma)} < \sum_{i \in \mathcal{U}(\sigma_2)} \overline{d}_{ip(\sigma)}$. Thus, activities of a given type are preferred from projects having less expected remaining workload. As a consequence the benefit of PO-priority policies based on the bottleneck dynamic approach is reduced as their general counterparts already mimic in part the behavior of POPs. Then, reducing the decisions sets by considering PO-priority policies may also increase the probability of suboptimal decisions to be selected. This explains the inferior performance of the PO-version of BD-GC-U.

Table 7.22 Performance of PO-priority policies for a single project type

	General (%)	PO (%)
BD-GC-U	5.4	6.2
BD-GC-D	4.3	4.3
WSPT	33.0	25.1
RAN	31.3	22.7
Best	2.9	3.6

Table 7.23 Performance of PO-priority policies for two project types

	General (%)	PO (%)
BD-GC-U	13.5	13.4
BD-GC-D	2.9	3.0
WSPT	17.9	16.6
RAN	27.0	22.9
Best	2.7	2.8

Instances with two project types Table 7.23 shows the mean solution gap for the general priority policies and that of PO-priority policies for the instances with two project types (Table 7.23).

Again the PO-priority policies BD-GC-U- and the BD-GC-U-policies do not outperform the general counterparts on average while the PO-priority policies WSPT and the RAN do.

We conclude that PO-priority policies slightly outperform general priority policies. But essentially the benefit of optimal policies as shown in Sect. 7.5.4.2 remains.

7.5.5 Results for the Non-preemptive Problem

7.5.5.1 State Space Cardinalities

In order to bound computation times we, again, have set a limit of 1,000,000 for the state space cardinality.

Instances with a single project type Table 7.24 show the cardinalities for the set of problem instances. As, again, $\mathcal{S}$ and $\mathcal{S}^{\mathrm{PO}}$ to a lesser extend depend on $|\Sigma|$ the cardinalities implicitly depend on the network structure given by OS, the number of activity types $|\mathcal{V}_p|$ and $K^{\max}$. Thus, the following tables show $|\mathcal{S}|$ and $|\mathcal{S}^{\mathrm{PO}}|$ for different values of OS and $K^{\max}$ when general policies and POPs are considered. As the state space cardinalities depend in part on $|\mathcal{R}|$ we have non-integer means. The percentages indicates the fraction of the cardinality of $\mathcal{S}^{\mathrm{PO}}$ from $\mathcal{S}^{\mathrm{P}}$.

The results confirm the observation already made for the preemptive problem that for small values of OS where networks have less precedence relations state spaces may become very large when general policies are considered. Again, the growth can be considerably tempered again using POPs such that $|\mathcal{S}^{\mathrm{PO}}|$ may only a

Table 7.24 State space cardinalities for a single project type at different levels of *OS*

$K^{\max}$		5			10		
OS_1	$\lvert\Sigma\rvert$	$\lvert\mathcal{S}\rvert$	$\lvert\mathcal{S}^{\mathrm{PO}}\rvert$	(%)	$\lvert\mathcal{S}\rvert$	$\lvert\mathcal{S}^{\mathrm{PO}}\rvert$	(%)
0	175	–	57,511	–	–	–	–
0.2	82.1	–	20,355.8	–	–	478,418.3	–
0.4	32.4	53,963.7	5,692.8	10.55	–	127,784.3	–
0.6	24.9	18,906.5	4,469.2	23.64	–	101,045.3	–
0.8	17.7	5,275.7	2,371.8	44.96	241,337.8	52,618.6	21.80
1	10	845.2	805.8	95.34	15,379.2	15,379.2	100.00

Table 7.25 State space cardinalities for two project types at different levels of *OS*

$K^{\max}$		5			10		
OS_p	$\lvert\Sigma\rvert$	$\lvert\mathcal{S}\rvert$	$\lvert\mathcal{S}^{\mathrm{PO}}\rvert$	(%)	$\lvert\mathcal{S}\rvert$	$\lvert\mathcal{S}^{\mathrm{PO}}\rvert$	(%)
0	46	167,434	27,445	16.39	–	–	–
0.3	27	26,525.6	9,264.4	34.93	–	333,969.8	–
0.6	19.4	8,217	4,740.8	57.70	511,985.4	159,948.8	31.24
1	12	1,755.8	1,755.8	100.00	50,243.8	50,243.8	100.00

small percentage of $|\mathcal{S}|$. Thus, using POPs, we could solve all problem instances to optimality where using general policies for instances with $K^{\max} = 10$ or $OS \leq 0.2$ the state space was larger than our limit of 1,000,000 states.

Instances with two project types Table 7.25 confirms the observations made in Sect. 7.5.5.1 for the case with two project types. Again we could determine the optimal POP for all instances except for $K^{\max} = 10$ and $OS_1 = 0$.

7.5.5.2 Performance of Optimal Policies

At first, we note that, for the instances tested, we could always verify where an optimal general policy could be determined that POPs are globally optimal as well.

Next, we report on the performance of optimal POPs relative to priority policies. In order to have the same numbers of instances at each combination of problem parameters we only consider ranges of parameters where for all instances an optimal POP could be determined such that instances with $OS_1 = 0$ are excluded from consideration.

Instances with a single project type Table 7.26 shows the mean optimality gap (given by $\frac{g(\pi^{\mathrm{prio}})}{g(\pi^{*PO})} - 1$) of the different priority policies to the optimal policy at different levels of the order strength (*OS*). The mean gap in the column *Best* is based on the best solution gap obtained for each problem instance. The column *Max Best* shows the maximum value for the best gap obtained in order to demonstrate the range of possible gaps.

Table 7.26 Performance of RBPs for a single project type at different levels of *OS*

OS_p	BD-GC-U (%)	BD-GC-D (%)	WSPT (%)	RAN (%)	Best (%)	Max best (%)
0.2	2.8	2.8	31.0	41.4	2.5	18.2
0.4	4.8	4.5	22.3	31.6	3.9	19.8
0.6	4.3	4.4	18.9	27.2	3.1	21.9
0.8	4.1	4.1	15.0	26.1	2.8	16.1
1	7.3	7.2	11.7	18.7	2.8	19.2

Table 7.27 Performance of priority policies for a single project type at different levels of $CV_{\bar{d}_r}$

$CV_{\bar{d}_r}$	BD-GC-U (%)	BD-GC-D (%)	WSPT (%)	RAN (%)	Best (%)	Max best (%)
[0; 0.2]	3.8	3.8	37.6	26.6	3.3	10.8
[0.4; 0.6]	5.2	5.1	15.8	26.0	3.6	21.9
[0.8; 1.0]	5.0	4.9	5.9	34.4	2.2	19.0

Table 7.28 Performance of priority policies for two project types at different levels of *OS*

OS_p	BD-GC-U (%)	BD-GC-D (%)	WSPT (%)	RAN (%)	Best (%)	Max best (%)
0.3	8.9	3.1	11.8	26.6	2.9	12.1
0.6	8.0	2.8	9.7	23.0	2.7	13.6
1.0	14.8	3.2	6.5	22.5	2.6	13.6

Table 7.29 Performance of priority policies for two project types at different levels of CV_d

$CV_{\bar{d}_r}$	BD-GC-U (%)	BD-GC-D (%)	WSPT (%)	RAN (%)	Best (%)	Max best (%)
[0; 0.2]	4.0	3.4	10.0	16.1	3.3	11.3
[0.4; 0.6]	7.2	3.8	12.7	22.0	3.4	13.6
[0.8; 1.0]	20.5	1.9	5.3	34.0	1.5	7.7

First of all, we observe that it is beneficial to search for optimal policies especially for an *OS* larger than 0.2. For a small *OS* the performance of the BD-policies is near optimal on average while for higher *OS*-values the best gaps are about 3 % on average. However, much are higher gaps up to 21.9 % are possible. Table 7.27 shows the mean gaps for different ranges of $CV_{\bar{d}_r}$. We observe that for higher ranges optimal policies become less beneficial as an mean of 2.1 % of the best gap indicates. This can be explained by the fact that WSPT becomes more beneficial.

Instances with two project types Tables 7.28 and 7.29 confirm that the observations made for a single project type are also valid for multiple project types.

7.5.5.3 Performance of Project State Ordering Priority Policies

In the following, we discuss the results for PO-priority policies in case of a single project type as well as in case of two project types.

Table 7.30 Performance of PO-priority policies for a single project type

	General (%)	PO (%)
BD-GC-U	4.7	4.3
BD-GC-D	4.6	4.2
WSPT	19.8	17.2
RAN	29.0	23.9
Best	3.1	2.4

Table 7.31 Performance of PO-priority policies for two project types

	General (%)	PO (%)
BD-GC-U	10.6	10.4
BD-GC-D	3.0	3.0
WSPT	9.3	8.7
RAN	24.0	22.8
MinGap	2.7	2.6

Instances with a single project type Table 7.30 shows the mean solution gap of general priority policies and of PO-priority policies for the instances with a single project type.

We observe that the PO-versions of all priority policies show an improved performance on average.

Instances with two project types Table 7.30 shows the mean solution gap of the general priority policies and of PO-priority policies for the instances with two project types (Table 7.31).

As for the case with a single project type, we observe that the PO-versions of all priority policies show an improved performance on average. Interestingly both PO-priority policies based on the bottleneck dynamics approach (BD-GC-U, BD-GC-D) outperform their general counterparts. This can be explained by the fact that POPs according to their definition for the non-preemptive case take into account activities in process and waiting activities which none of the general priority policies do.

We conclude that also for the non-preemptive case PO-priority policies slightly outperform general priority policies. But, essentially, the benefit of optimal policies as shown in Sect. 7.5.4.2 remains.

7.5.5.4 Performance of the Value Function Approximation from the Preemptive Problem

In this section, we investigate the performance of the policy based on the value function from the preemptive problem. We denote the policy in the following as ADP-P-PO as we consider only the PO-version of the policy. At first, we discuss the performance for the instances with a single project type before we consider the performance for the instances with two project types. As a benchmark for ADP-P-PO, we use the solution gap obtained for the best general priority policy as well as for the best PO-priority policy.

Table 7.32 Performance of ADP-P-PO for a single project type at different levels of *OS*

		Best	
OS_1	ADP-P-PO (%)	General (%)	PO (%)
0.2	0.4	2.5	1.3
0.4	0.5	3.9	2.9
0.6	0.6	3.1	2.5
0.8	0.7	2.8	2.5
1	0.4	2.8	2.8

Table 7.33 Performance of ADP-P-PO for a single project type at different levels of CV_d

		Best	
$CV_{\overline{d}_r}$	ADP-P-PO (%)	General (%)	PO (%)
[0; 0.2]	0.2	3.3	2.8
[0.4; 0.6]	0.3	3.6	3.1
[0.8; 1.0]	1.1	2.2	1.2

Table 7.34 Performance of ADP-P-PO for two project types at different levels of *OS*

		Best	
OS_p	ADP-P-PO (%)	General (%)	PO (%)
0.3	2.3	2.9	2.7
0.6	2.5	2.7	2.6
1	2.8	2.6	2.6

Instances with a single project type Table 7.32 shows the mean solution gaps for different levels of OS.

We observe that for each level ADP-P-PO outperforms on average the best priority policies (general RBPs and RBPOPs).

Table 7.33 shows the average solution gaps for different levels of $CV_{\overline{d}_r}$.

We observe that for low ranges of $CV_{\overline{d}_r}$ ADP-P-PO clearly outperforms the priority policies on average. However, for high ranges of $CV_{\overline{d}_r}$, the performance of the preemptive approximation deteriorates but remains better on average than the performance of the priority policies. This can be explained by the fact that for high $CV_{\overline{d}_r}$ activities with extremely long expected durations may occur. For such activities the effect of not allowing preemptions has a higher impact on optimal scheduling decisions.

Instances with two project types Table 7.34 shows the average solution gaps for different levels of order strength.

Here, we observe that on average ADP-P-PO performs at most slightly better. The observation can be explained by considering the average performance for different levels of $CV_{\overline{d}_r}$ as shown in Table 7.35.

We observe that, for low ranges of $CV_{\overline{d}_r}$, ADP-P-PO clearly outperforms on average the priority policies. However, for high ranges of $CV_{\overline{d}_r}$, its performance is worse than that of the priority policies. Thus, the mean gaps at different levels of *OS* are adversely affected by the bad performance for high ranges of $CV_{\overline{d}_r}$.

Table 7.35 Performance of ADP-P-PO for two project types at different levels of CV_d

$CV_{\overline{d}_r}$	ADP-P-PO (%)	Best	
		General (%)	PO (%)
[0; 0.2]	0.4	3.3	3.1
[0.4; 0.6]	1.2	3.4	3.4
[0.8; 1.0]	6.0	1.5	1.5

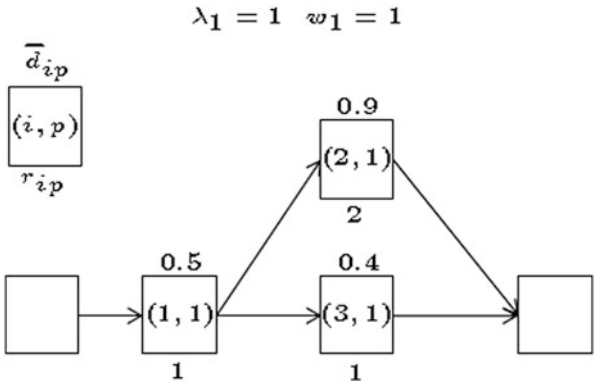

Fig. 7.10 Example used to test approximation architectures

7.5.6 *Performance of Linear Function Approximation*

In this section, we investigate the performance of using linear function approximation for the case of open systems (unbounded $K^{\max}$).

In Sect. 7.5.6.1, we test firstly, a number of methodological options from Sect. 7.4.3 such as the approximation architectures to be used or the determination of representative states and derive recommendations. In Sect. 7.5.6.2, we present a number of case studies to illustrate the benefit of linear function approximation.

7.5.6.1 Test of the Approximation Architectures and First Computational Insights

For our investigation, we consider the following example with two resource types where $c_1 = c_2 = 1$ and unbounded $K^{\max}$ (Fig. 7.10).

In order to test the approximation architectures discussed in Sect. 7.4.3.1 we approximate the open system using a semi-open system with $K^{\max} = 50$. Using simulation, we obtain for $g^{\circ}(\text{RAN})$ an estimate of 14.934. Then, we set $y_1 = 485$ for which we obtain for $g(\text{RAN})$ an estimate of 14.985 such that $g(\text{RAN}) \approx g^{\circ}(\text{RAN})$.

Next, an optimal preemptive POP yielding $g^* = 13.568$ has been computed for the semi-open system which serves as a benchmark for assessing the quality of the policies obtained from value function approximation.

Table 7.36 Performance of the approximation architectures for different sets of representative states

	[0;5]			[0;10]		
Architecture	MSQE $\tilde{\mathcal{S}}$	MSQE $\mathcal{S}^{\mathrm{P}}$	g	MSQE $\tilde{\mathcal{S}}$	MSQE $\mathcal{S}^{\mathrm{P}}$	g
PSLin1	100.085	1.434e + 10	16.690	1,039.888	1.848e + 09	16.690
PSLin2	0.620	4.738e + 08	13.935	9.810	3.722e + 07	14.397
PSLin3	0.013	2.324e + 09	13.930	0.325	5.383e + 07	14.760
QLin1	113.221	1.447e + 10	16.690	1,318.612	1.929e + 09	16.690
QLin2	0.997	5.344e + 08	14.257	19.313	4.669e + 07	14.385
QLin3	0.049	3.552e + 09	13.993	1.252	1.501e + 08	14.166
	[0;15]			[0;20]		
Architecture	MSQE $\tilde{\mathcal{S}}$	MSQE $\mathcal{S}^{\mathrm{P}}$	g	MSQE $\tilde{\mathcal{S}}$	MSQE $\mathcal{S}^{\mathrm{P}}$	g
PSLin1	4,482.944	4.373e + 08	14.581	13,161.322	1.332e + 08	14.581
PSLin2	51.163	6.547e + 06	14.434	172.975	1.679e + 06	14.538
PSLin3	1.524	3.668e + 06	14.587	4.786	3.882e + 05	14.328
QLin1	6,133.586	4.718e + 08	16.690	18,965.209	1.493e + 08	16.690
QLin2	114.682	9.084e + 06	14.713	420.704	2.580e + 06	15.0872
QLin3	8.079	1.773 + 07	14.617	31.612	3.095e + 06	14.731

Performance of approximation architectures and impact of representative states For the investigation, we use as set of representative states $\tilde{\mathcal{S}} = \{s \in \mathcal{S}^{\mathrm{P}} | K(s) \in [K^{\mathrm{L}}, K^{\mathrm{U}}]\}$. The intervals $[K^{\mathrm{L}}, K^{\mathrm{U}}]$ considered were $[0, 5]$, $[0, 10]$, $[0, 15]$ and $[0, 20]$. For each interval, we fitted the approximation architectures to the value function of the states in $\tilde{\mathcal{S}}$. The state relevance weights were set to 1 for all our experiments in this section. The results are shown in Table 7.36. The column *Architecture* shows the name of the architecture. The columns *MSQE* $\tilde{\mathcal{S}}$ and *MSQE* $\mathcal{S}^P$ show the *mean squared error* (MSQE) of the approximate value functions to the value function for the semi-open system computed for the states in $\tilde{\mathcal{S}}$ and $\mathcal{S}^{\mathrm{P}}$ respectively. The columns g show the average cost obtained when the policy based on the approximate value function is applied on the semi-open system. Obviously, the best performance is obtained when architectures with a degree of the polynomial larger than one are fitted in representative states near the origin of the state space (state s^0). A possible explanation is the fact that near the origin the value function has smaller values (value function is nearly monotone in the number of projects in the system). Thus, the value function tends to have less extreme variations between the states such that an architecture may capture more easily the structure of a value function that is relevant for a good policy. Furthermore, states near the origin seem to have higher probabilities such that they are more relevant for determining optimal policies.

Next, we observe that, in terms of MSQE and average cost, a degree of 1 is not sufficient to capture the structure of the value function. Furthermore, architectures

Table 7.37 Number of basis functions for the preemptive problem

	Preemptive	
Architecture	General	PO
PSLin1	5	5
PSLin2	15	14
PSLin3	35	30
QLin1	4	4
QLin2	10	10
QLin3	20	20

based on a cubic polynomial (PSLin3 and QLin3) lead to smaller values of MSQE in $\tilde{\mathcal{S}}^{\mathrm{P}}$ than architectures based on a quadratic polynomial (PSLin2 and QLin2) but not to much better policies. In some cases, policies resulting from the architectures based on a cubic polynomial perform even worse. An explanation is that a higher degree of the polynomial allows better fits for the representative states in $\tilde{\mathcal{S}}^{\mathrm{P}}$ but makes the architecture more susceptible to *overfitting* (cf. Roubos [112]). Overfitting refers to the phenomenon that an architecture that has been perfectly fitted to a subset of the state space does not generalize well to the entire state space as the architecture has too many parameters. This is indicated MSQE $\mathcal{S}^{\mathrm{P}}$ values that are higher for architectures based on cubic polynomials than for architectures based on quadratic polynomials. The observation is in line with the findings of Roubos and Bhulai [114] who have approximated the value functions for queueing systems. We conclude that the architectures based on a quadratic polynomial are sufficient to capture the structure of the value function such that a good policy can be obtained. In the following we restrict our considerations to approximation architectures based on a quadratic polynomial (PSLin2 and QLin2).

Finally, we observe that the architectures based on the queueing network perspective (QLin1, QLin2, QLin3) deliver slightly worse approximations that the architectures based on the project states (PSLin1, PSLin1, PSLin3). Despite the equivalence of both perspectives (cf. Theorem 7.3.2) in case of POPs basis functions based on simple queue lengths neglect important features. An example is the case where two activities waiting for a resource are from the same project. If those activities have the same set of successors it may be less important which activity to be scheduled first as the successors need the completion of both. Such features are better captured by an architecture based on project states. However, architectures based on the queueing network perspective have less basis functions (except for project networks with $OS = 1$). Table 7.37 shows for the example the number of basis functions for each architecture when general scheduling policies and when POPs are considered.

We observe that POPs help to reduce the number of basis functions for the architectures based on project states. This is due to the fact that when following a POP states of projects of a given type $p \in \mathcal{P}$ are always ordered according to the sets of unfinished activities (cf. Theorem 7.3.1). Thus, combinations of non-ordered project states may no longer occur such that the corresponding product terms can be

Table 7.38 Improvement by sampling representative states

Architecture	Fixed	Sampled
PSLin2	13.935	13.628
QLin2	14.257	13.618

Table 7.39 Performance of ADP-PI-LS and ADP-PI-BE

	ADP-PI-LS		ADP-PI-BE	
Iteration	PSLin2	QLin2	PSLin2	QLin2
0	14.558	14.558	14.558	14.558
1	14.365	14.090	13.854	16.400
2	13.663	13.617	14.467	13.962
3	13.647	13.633	14.419	14.872
4	13.648	13.626	13.740	13.785
5	13.645	13.627	13.700	14.087
Best	13.645	13.617	13.700	13.785

eliminated from the architecture. Finally, we verify that the architectures based on the queueing network perspective have much less basis functions such that they are an interesting option if there are many project states.

Sampling of representative states Next, we have tested the effect of sampling representative states. We sampled 20 sets $\tilde{\mathcal{S}}_1^{\mathrm{P}}, \ldots, \tilde{\mathcal{S}}_{20}^{\mathrm{P}} \subset \tilde{\mathcal{S}}^{\mathrm{P}}$ of 50 states from the set $\tilde{\mathcal{S}}^{\mathrm{P}} = \left\{s \in \mathcal{S}^{\mathrm{P}} | K(s) \in [0,5]\right\}$. Table 7.38 contrasts for the PSLin2- and the QLin2-architecture the best policy obtained over all sampled sets with the policy obtained for the set $\tilde{\mathcal{S}}^{\mathrm{P}}$.

We observe that, by searching over multiple sampled sets of representative states, the performance of the approximation w.r.t. average cost obtained for the resulting policy can be considerably improved such that near optimal policies can be obtained with an optimality gap of only 0.44 % for the PSLin2-architecture and 0.36 % for the QLin2-architecture.

Test of approximate policy iteration As a next step, we have tested the idea of policy improvement using the approximate policy iteration algorithm based on least square fit to the value function for the semi-open system with $K^{\max} = 50$ (Algorithm 5) and based on Bellman error minimization (Algorithm 6). In the following, we refer to the resulting policies as ADP-LS-PI and ADP-BE-PI. Thus, we can verify that the architectures are able to capture the value function of different intermediate policies. Furthermore, we verify that the architectures, indeed, capture sufficient structure of the value function such that they can be used in ADP-approaches that exploit the principles of dynamic programming. Approximate policy iteration should exhibit a similar convergence behavior (at least partially) as policy iteration applied on the true value function. As initial policy, we have used BD-GC-U which has performed best.

Table 7.39 shows the results of the architectures PSLin2 and QLin2 for five iterations. Obviously, the idea of policy improvement works as, for many iterations,

Table 7.40 Performance of priority policies

	Average cost	
Priority policy	Preemptive	Non-preemptive
WSPT	14.540	14.510
BD-GC-U	14.524	14.523
BD-GC-D	14.524	14.511
RAN	14.527	14.616

a better policy is derived based on the policy from a previous iteration. As, for each iteration, the same sets of representative states are used, the change of a policy can be attributed to the policy improvement step. As the value function is approximated, the improvement step may lead to an inferior policy (cf. Bertsekas and Tsitsiklis [17]). However, it is worthwhile to perform multiple improvement steps in order to obtain a near optimal policy as for both algorithms and architectures the average cost tend to decrease over multiple iterations. Furthermore, we observe that the performance of ADP-PI-BE is slightly inferior to ADP-PI-LS. This can be explained by the inferior quality of the approximations obtained from the Bellman error method as it uses less information than the linear regression in ADP-LS-PI. However, it can be applied in cases where even for the semi-open system the state space becomes too large. Finally, for both algorithms and architectures a near optimal policy is obtained (solution gap is at most 1.5 %).

Performance for open systems Next, we examine how well the policies obtained for the semi-open system generalize to an open system for the preemptive as well as the non-preemptive problem. As we no longer have an optimal policy, we present as a benchmark the results obtained for PO-priority policies (with a RAN used as tie breaker) for the preemptive and non-preemptive problem in Table 7.40. Here, we only consider PO-priority policies such that the POPs from the ADP-approaches do not have an advantage from the elimination of decisions by the restriction to POPs. At first, we examine the effect of $K^{\max}$ of the semi-open systems for approximating an open system where we apply Algorithm 4 with 20 sets of representative states (50 states each) with an interval $\left[K^{\mathrm{L}}, K^{\mathrm{U}}\right] = [0; 5]$. The resulting policy is denoted in the following as ADP-LS. Tables 7.41 and 7.42 show the results for the architectures PSLin2 and QLin2 separately. The column *Optimal* shows the average cost for the optimal policy for the semi-open system while the columns *Preemptive (semi-open)* and *Preemptive (open)* show the average cost for ADP-LS when applied to the semi-open system as well as an open system with preemptions allowed. For open systems we have evaluated the policies using simulation with 10 replications of 500,000 project arrivals each. The warm up period had a length of 10,000 arrivals. Finally, the column *Non-preemptive (semi-open)* show the results when the value function approximation of ADP-LS is used as outlined in Sect. 7.4.3.2 in order to obtain a policy for the non-preemptive problem.

For both architectures, the policies generalize well to open systems where the performance deteriorates when the truncation level is reduced. This can be explained by the higher distortion of the value function due to boundary of the state space.

Table 7.41 Performance of the PSLin2-architecture at different levels of $K^{\max}$ and generalization to open systems

Kmax	Optimal	Preemptive (semi-open)	Preemptive (open)	Non-preemptive (open)
50	13.568	13.628	13.887	14.023
20	13.731	13.830	13.830	14.104
10	13.835	13.885	14.469	14.533

Table 7.42 Performance of the QLin2-architecture at different levels of $K^{\max}$ and generalization to open systems

Kmax	Optimal	Preemptive (semi-open)	Preemptive (open)	Non-preemptive (open)
50	13.568	13.619	13.873	14.008
20	13.731	13.831	13.900	14.052
10	13.835	13.875	14.188	14.311

Table 7.43 Performance of the policies obtained from ADP-LS-PI,ADP-BE-PI,ADP-VI

	Open-PSLin2		Open-QLin2	
Policy	Preemptive	Non-preemptive	Preemptive	Non-preemptive
ADP-LS-T	13.887	14.023	13.873	14.008
ADP-LS-PI	13.900	14.054	13.876	14.016
ADP-BE-PI	13.976	14.124	14.053	14.218

However, even for low $K^{\max}$ policies that are superior to the priority policies may be obtained. Thus, we conclude that generally semi-open systems even with lower $K^{\max}$ may be used for approximating open systems.

Table 7.43 shows the performance of the best policy obtained from ADP-LS, ADP-PI-LS and ADP-PI-BE for both architectures. Again, the approximations have been applied to the semi-open system (preemptive) with $K^{\max} = 50$ and to an open system with and without preemptions.

We observe that while the performance of the policies is near optimal for the semi-open system the policies outperform the PO-priority policies (cf. Table 7.40).

Application of linear function approximation to non-preemptive problem In principle, linear function approximation can also be directly applied to the non-preemptive problem. However this has two drawbacks. Firstly, larger state spaces need to be considered when using linear regression for obtaining an approximation. Secondly, the PSLin-architectures become larger as the Table 7.44 shows.

Thus, as for the non-preemptive problem, the approximation based on the preemptive problem has worked sufficiently well, we have restricted our considerations to the preemptive problem and use the approximation for the non-preemptive problem.

Finally, we remark that, although the investigation has been carried out on a single problem instance, we could verify the observations made so far also for other problem instances which we have tested.

Table 7.44 Number of basis functions for the non-preemptive problem

	Preemptive	
Architecture	General	PO
PSLin1	11	11
PSLin2	57	48
PSLin3	203	144
QLin1	3	3
QLin2	9	9
QLin3	19	19

7.5.6.2 Application of Linear Function Approximation to Selected Problem Instances

According to our observations application of linear function approximation needs tuning w.r.t. number and size for the sets of representative states, range for the sets of representative states and the architecture to be used (PSLin2 or QLin2). Furthermore, linear function approximation does not work well for every problem instance for two reasons.

1. Approximation architectures may not capture the structure of the value function well for each problem instance. An option to remedy this problem may be to use more elaborate architectures such as piece-wise quadratic approximation architectures (cf. Veach [130] for an example). However, the computational burden for finding good sets of representative states may increase. Furthermore, techniques providing approximations of better quality such as those based on simulation (cf. Bertsekas and Tsitsiklis [17] or Powell [105]) may be considered.
2. As already mentioned, linear function approximation works best with sets of representative state from the region near the origin of the state space (given by state s^0). Thus, there may be the problem that the architecture may work for semi-open systems but does not generalize well for open systems. For open systems the system may be in regions of the state space with larger numbers of projects in the system $K(s)$ where the value function may differ in its structure from the value function near the origin of the state space. We have observed that this is the case, in particular, for instances where project types have different holding cost w_p per time unit. However, if project types have equal holding costs w_p per time unit the architectures, discussed so far, delivered in many cases promising results. Again, to remedy this limitation more elaborate architectures such as piece-wise quadratic approximation architectures might improve the quality of the approximations.

In the following, we demonstrate for a number of problem instances that linear function approximation can successfully be applied in order to obtain policies for open systems that outperform priority policies. The first two examples are of medium size (5 and 6 activity types) while the third one is a larger one with 10 activity types.

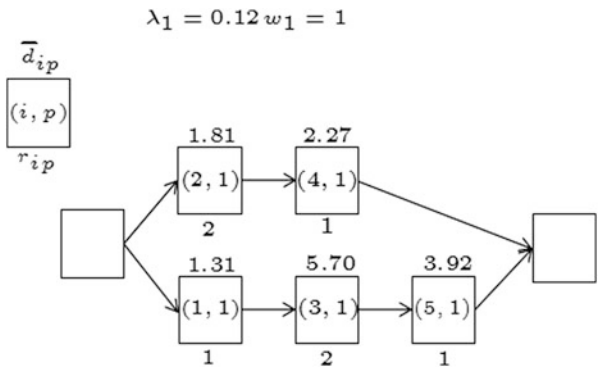

Fig. 7.11 Data of the problem instance with a single project type composed of five activity types

The experimental setup is as follows. For the problem instance, we have derived preemptive POPs using a semi-open system with $K^{\max} = 10$ (ADP-LS) and approximate policy iteration based on the Bellman error minimization (ADP-PI-BE). The evaluation of the sets of representative states and the intermediate policies as a part of the Bellman error method was based on simulation. State relevance weights have been set to 1 without further tuning as sufficiently good results have been obtained.

Furthermore, we have combined the two methods with the PSLin2 and the QLin2 architecture such that we have obtained four policies from ADP that are namely ADP-LS-PSLin2, ADP-LS-QLin2, ADP-PI-BE-PSLin2 and ADP-PI-BE-QLin2. As a benchmark, we consider different PO-priority policies that are namely BD-GC-U, BD-GC-D, WSPT and RAN. We also have tested the general versions of the priority policies and have found that they do not have significant advantages or perform even worse than the PO-priority policies.

In order to evaluate the policies, we have used 10 simulation runs of 500,000 projects arriving at the system. For the evaluation of representative states as a part of the Bellman error method we have used shorter simulation runs of only 100,000 project arrivals. The warm up period was set to 10,000 project arrivals for all simulation runs. Variance has been reduced using *common random numbers* (cf. Law [84]) such that the paired t-confidence intervals (cf. Law [84]) for any two policies do not contain 0 at a confidence level of 95 %.

Single project type with five activity types We consider a problem instance with a single project type composed of five activity types. They are to be processed by two resource types where we assume $c_r = 1\ \forall r \in \mathcal{R}$. Figure 7.11 gives the information of the project type.

The network has an OS of 0.6. We have set the arrival rate and the activity durations such that a utilization per resource of 0.9 is attained.

For the semi-open system, we have obtained a CTMDP with $|\mathcal{S}^{\mathrm{P}}| = 18{,}876$ states and $|\Sigma^{\mathrm{P}}| = 11$ project states while the PSLin2 architecture has 60 basis func-

Table 7.45 Performance of the preemptive policies obtained for the instance with a single project type consisting of five activity types

Policy	Average cost
ADP-LS-PSLin2	13.280
ADP-LS-QLin2	12.931
ADP-PI-BE-PSLin2	13.256
ADP-PI-BE-QLin2	13.514
BD-GC-U	16.412
BD-GC-D	16.437
WSPT	17.771
RAN	15.133

Table 7.46 Performance of the non-preemptive policies obtained for the instance with a single project type consisting of five activity types

Policy	Average cost
ADP-LS-PSLin2-P	13.702
ADP-LS-QLin2-P	13.301
ADP-PI-BE-PSLin2-P	13.726
ADP-PI-BE-QLin2-P	14.011
BD-GC-U	16.454
BD-GC-D	16.422
WSPT	17.548
RAN	15.778

tions and the QLin2 architecture 21 basis functions. For both architectures (PSLin2 and QLin2) 50 sets of 500 representative states from the interval $[K^L; K^U] = [0; 5]$ have been used. Table 7.45 shows the average cost for the four policies and different PO-priority policies as a benchmark (Table 7.45).

Firstly, we observe that all policies obtained from ADP clearly outperform the PO-priority policies. Secondly, policies based on the QLin2 architecture are superior to policies based on the PSLin2 architecture if ADP-LS is used and inferior if ADP-PI-BE is used. Obviously, although the QLin2 architecture has much less basis functions it captures enough structure in order obtain policies performing reasonably well.

Next, we have used the approximations for the preemptive problem as outlined in Sect. 7.4.4 in order obtain policies for the non-preemptive problem. The policies are namely ADP-LS-PSLin2-P, ADP-LS-QLin2-P, ADP-PI-BE-PSLin2-P and ADP-PI-BE-QLin2-P. Table 7.46 shows the results for the policies obtained from ADP as well as the PO-priority policies (Table 7.46). The results are similar to the preemptive problem. Again the policy from ADP outperform the PO-priority policies.

Two small project types We consider a problem instance with two small project types. Each has three activity types which are to be processed by two resource types with $c_r = 1 \ \forall r \in \mathcal{R}$. Figure 7.12 gives the information of the two project types.

The networks have an OS of 0.6. We have set the arrival rate and the activity durations such that a utilization per resource of 0.9 is attained.

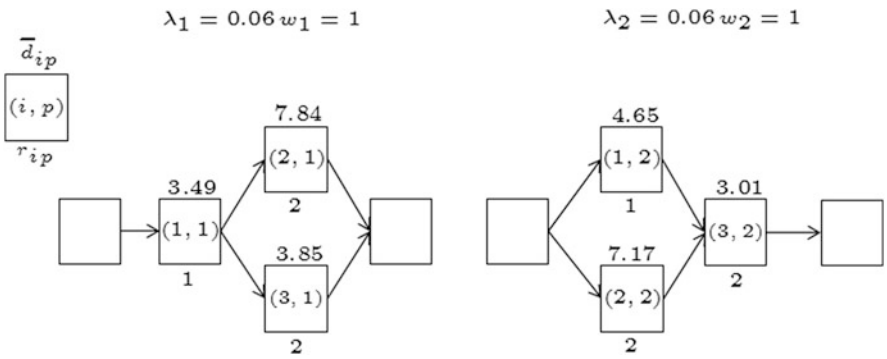

Fig. 7.12 Data of the problem instance with two project types composed of three activity types each

Table 7.47 Performance of the preemptive policies obtained for the instance with two project types consisting of three activity types each

Policy	Average cost
ADP-LS-PSLin2	11.840
ADP-LS-QLin2	12.718
ADP-PI-BE-PSLin2	12.177
ADP-PI-BE-QLin2	12.406
BD-GC-U	16.608
BD-GC-D	13.388
WSPT	14.881
RAN	14.398

For the semi-open system we have obtained a CTMDP with $|\mathcal{S}^{\mathrm{P}}| = 21,021$ states and $|\Sigma^{\mathrm{P}}| = 8$ project states while the PSLin2-architecture has 43 basis and the QLin2-architecture 28 basis functions. Again we have used 50 sets of representative states from an interval $[K^L; K^U] = [0; 5]$ where for the PSLin2-architecture the number of representative states has been set to 500 and for the QLin2-architecture to 200.

Table 7.47 shows the average cost for the four policies and different PO-priority policies as a benchmark. Firstly, we observe that the policies obtained from ADP using the PSLin2 architecture clearly outperform the PO-priority policies. Secondly, policies using the QLin2-architecture are inferior to the policies using the PSLin2-architecture but also outperform the PO-priority policies. Next, we derived policies for the non-preemptive problem. The results are shown in Table 7.48. Again, the policies from ADP outperform the PO-priority policies.

Single project type with ten activity types We consider a problem instance with a single project type composed of 10 activity types. They are to be processed by two resource types where we assume $c_r = 1\ \forall r \in \mathcal{R}$. Figure 7.13 gives the information of the project type. The network has an OS of 0.8. We have set the arrival rate and the activity durations such that a utilization per resource of 0.9 is attained.

Table 7.48 Performance of the non-preemptive policies obtained for the instance with two project types consisting of three activity types each

Policy	Average cost
ADP-LS-PSLin2-P	12.273
ADP-LS-QLin2	12.975
ADP-PI-BE-PSLin2-P	12.618
ADP-PI-BE-QLin2	12.730
BD-GC-U	16.560
BD-GC-D	13.377
WSPT	14.700
RAN	14.589

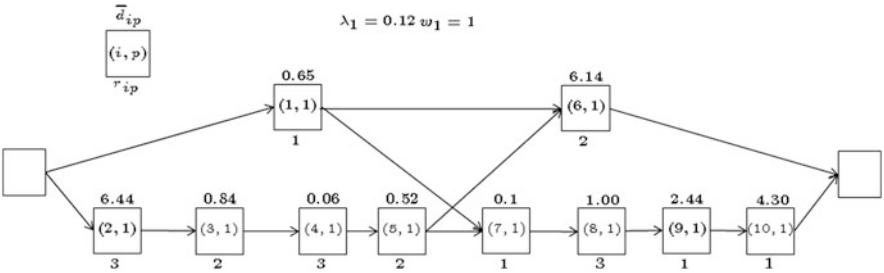

Fig. 7.13 Data of the problem instance with a single project type composed of 10 activity types

Table 7.49 Performance of the preemptive policies obtained for the instance with a single project type consisting of 10 activity types

Policy	Average cost
ADP-PI-BE-PSLin2	18.266
ADP-PI-BE-QLin2	18.454
BD-GC-U	24.708
BD-GC-D	24.674
WSPT	20.015
RAN	24.496

For the semi-open system we have obtained a CTMDP with $|\Sigma^{\mathrm{P}}| = 18$ project states while the PSLin2 architecture has 170 basis functions and the QLin2 architecture 66 basis functions. Due to the larger number of basis functions, we have used 200 sets of representative states from an interval $[K^L; K^U] = [0; 5]$ where for the PSLin2 architecture the number of representative states has been set to 2,000 and for the QLin2 architecture to 500. As $|\mathcal{S}^{\mathrm{PPO}}| > 1{,}000{,}000$ we only have used ADP-PI-BE for determining the weights of the approximation architectures.

Table 7.49 shows the average cost for the four policies and different PO-priority policies as a benchmark. Firstly, we observe that all policies obtained from ADP clearly outperform the PO-priority policies. Secondly, policies using the QLin2 architecture are, as expected, inferior to the policies using the PSLin2 architecture. However, although the QLin2 architecture has much less basis functions, it captures enough structure in order obtain policies outperforming the PO-priority policies.

Table 7.50 Performance of the non-preemptive policies obtained for the instance with a single project type consisting of 10 activity types

Policy	Average cost
ADP-PI-BE-PSLin2-P	21.399
ADP-PI-BE-QLin2-P	22.428
BD-GC-U	30.341
BD-GC-D	30.341
WSPT	22.143
RAN	40.380

Table 7.50 shows the results for the policies obtained from ADP as well as the PO-priority policies for the non-preemptive case. The results are similar to the preemptive problem. Again the policy with the PSLin2 architecture outperform the PO-priority policies while the policy with the QLin2 architecture performs slightly less well than WSPT but outperforms all other PO-priority policies.

Chapter 8
Integrated Dynamic Order Acceptance and Capacity Planning

This chapter is dedicated to the joint optimization of order acceptance (OA) decisions and capacity planning decisions. In Sect. 8.1, we model the problem as outlined in Sect. 2.3 as a CTMDP for which in Sect. 8.2 methodological issues are addressed for the efficient computation of optimal policies. In Sect. 8.3, we investigate the structure of optimal policies and the benefits of optimizing OA and capacity planning decisions.

8.1 Stochastic Dynamic Programming

In this section, we model the problem as a continuous-time Markov decision process (CTMDP) (cf. Puterman [108]) and give a detailed descriptions of its components. The framework has been taken from Powell [106].

8.1.1 State Variables

A state $s \in \mathcal{S}$ of the system, at any decision time, is given by

$$s = \left(\boldsymbol{n}^{\mathrm{W}}, \boldsymbol{n}^{\mathrm{E}}\right) \tag{8.1}$$

Let $\boldsymbol{n}^{\mathrm{W}} = \left(n_1^{\mathrm{W}}, n_2^{\mathrm{W}}, \ldots, n_{|\mathcal{P}|}^{\mathrm{W}}\right)$ and $\boldsymbol{n}^{\mathrm{E}} = \left(n_1^{\mathrm{E}}, n_2^{\mathrm{E}}, \ldots, n_{|\mathcal{P}|}^{\mathrm{E}}\right)$ denote vectors where n_p^{W} is the number of projects of type $p \in \mathcal{P}$ that are waiting to be processed and n_p^{E} is the number of projects of type $p \in \mathcal{P}$ that are in process. As we consider only a single resource $\boldsymbol{n}^{\mathrm{E}}$ is a unit vector. $s^0 = ((0, \ldots, 0), (0, \ldots, 0))$ is the system state where the system is empty.

P. Melchiors, *Dynamic and Stochastic Multi-Project Planning*, Lecture Notes in Economics and Mathematical Systems 673, DOI 10.1007/978-3-319-04540-5_8,

8.1.2 Decision Variables

We define the set of alternative decisions $\mathcal{A}(s)$ of state $s \in \mathcal{S}$. A decision $a \in \mathcal{A}(s)$ refers to the types of orders that are to be accepted or rejected, the projects to be scheduled next to the resource and the usage of non-regular capacity. Decisions on scheduling and usage of non-regular capacity are also referred to as capacity planning decisions. A decision a is given by

$$a = \left(\boldsymbol{\delta}^{\mathrm{M}}, \boldsymbol{\delta}^{\mathrm{E}}, \delta^{\mathrm{C}}\right) \tag{8.2}$$

Next, we explain the variables of a decision a in detail where we firstly address the variables $\boldsymbol{\delta}^{\mathrm{M}}$ related to order acceptance decisions and secondly address the variables $\boldsymbol{\delta}^{\mathrm{E}}$ and δ^{C} related to capacity planning decisions.

8.1.2.1 Order Acceptance Decisions

In order to keep the state space small we take the idea of Ross and Tsang [111] by deciding in state $s \in \mathcal{S}$ whether we accept an order if the next event is related to an arrival of an order of type $p \in \mathcal{P}$ or general type $\phi \in \Phi$. Hence, we do not have to store in an additional state variable whether an order has arrived or not. Then, vector $\boldsymbol{\delta}^{\mathrm{M}} = \left(\delta_1^{\mathrm{M}}, \delta_2^{\mathrm{M}}, \ldots, \delta_{|\mathcal{P}|}^{\mathrm{M}}\right)$ is a vector of binary variables δ_p^{M} that refer to the types $p \in \mathcal{P}$ of orders to be accepted as follows

$$\delta_p^{\mathrm{M}} = \begin{cases} 1 \text{ Accept the next order if it turns out to be of type } p. \\ 0 \text{ Reject the next order if it turns out to be of project type } p. \end{cases} \tag{8.3}$$

In order to determine the fixed cost associated with OA we have to distinguish three cases for each general project type $\phi \in \Phi$.

1. If $\delta_p^{\mathrm{M}} = 0 \ \forall p \in \mathcal{P}(\phi)$ – All projects of any type $p \in \mathcal{P}(\phi)$ are rejected such that no MPP is performed at all. Thus, no cost is incurred on rejecting projects of any type.
2. If $\delta_p^{\mathrm{M}} = 1 \ \forall p \in \mathcal{P}(\phi)$ – All projects of any type $p \in \mathcal{P}(\phi)$ are accepted such that no specific project types $p \in \mathcal{P}(\phi)$ need to be distinguished. Thus, it is optimal to postpone MPP if $k_\phi^{\mathrm{M}} \geq k_\phi^{\mathrm{PM}}$ and only a fixed cost k_ϕ^{PM} is incurred. If $k_\phi^{\mathrm{M}} < k_\phi^{\mathrm{PM}}$ (theoretical case) MPP should be performed such that a fixed cost k_ϕ^{M} is incurred.
3. $\delta_p^{\mathrm{M}} = 0$ for some $p \in \mathcal{P}(\phi)$ and $\delta_{p'}^{\mathrm{M}} = 1$ for some $p' \in \mathcal{P}(\phi)$ with $p' \neq p$ – MPP must be performed regularly before OA in order to determine the specific type $p \in \mathcal{P}(\phi)$ of an order. Hence, on acceptance or rejection of a project of type p a fixed cost k_ϕ^{M} is incurred.

Then, the fixed cost associated with the acceptance of an order of general type ϕ is given by

$$k^{\mathrm{A}}(\boldsymbol{\delta}^{\mathrm{M}}, \phi) = \begin{cases} 0 & \delta_p^{\mathrm{M}} = 0 \ \forall p \in \mathcal{P}(\phi) \\ \min\left\{k_\phi^{\mathrm{PM}}, k_\phi^{\mathrm{M}}\right\} & \delta_p^{\mathrm{M}} = 1 \ \forall p \in \mathcal{P}(\phi) \\ k_\phi^{\mathrm{M}} & \text{otherwise} \end{cases} \tag{8.4}$$

8.1.2.2 Capacity Planning Decisions

The vector $\boldsymbol{\delta}^{\mathrm{E}} = \left(\delta_1^{\mathrm{E}}, \delta_2^{\mathrm{E}}, \ldots, \delta_{|\mathcal{P}|}^{\mathrm{E}}\right)$ is a vector of binary variables δ_p^{E} that refers to the projects to be scheduled as follows

$$\delta_p^{\mathrm{E}} = \begin{cases} 1 \text{ Schedule a project of type } p. \\ 0 \text{ Do not schedule a project of type } p. \end{cases} \tag{8.5}$$

A *feasible* decision must be in line with a *non-idling* (cf. Meyn [93]) OA and resource allocation policy such that exactly one project must be scheduled if there are waiting projects and the resource is idle. Furthermore, a project in process may not be preempted.

The variable $\delta^{\mathrm{C}}(a)$ is defined to be the fraction of non-regular capacity used and refers to the project already scheduled (where $n_p^{\mathrm{E}}(s) = 1$) or the project scheduled by decision a (where $\delta_p^{\mathrm{E}}(a) = 1$ for some $p \in \mathcal{P}$ with $n_p^{\mathrm{W}}(s) \geq 1$). The cost per unit of time, after a decision a in system state $s \in \mathcal{S}$ is made, becomes

$$c(s, a) = \sum_{p \in \mathcal{P}} w_p \left(n_p^{\mathrm{W}}(s) + n_p^{\mathrm{E}}(s)\right) + w^{\mathrm{C}} \delta^{\mathrm{C}}(a) \tag{8.6}$$

8.1.3 Exogenous Information Process

Exogenous information arrives when an event occurs after making a decision in a system state. For our problem, we identify two kinds of events.

- Arrival of a new order of type p – This kind of event occurs at a rate λ_p. By means of its type we know its parameters such as expected duration, holding cost etc.
- Completion of a project of type p – This kind of event occurs at rate $(1 + \delta^{\mathrm{C}} z_p)\mu_p$.

8.1.4 Transition Function

A transition from one system state s to a subsequent system state s' takes place if an event occurs after making a decision a in system state s. The time to the next event

is exponentially distributed with rate $\beta(s,a)$ which is given by

$$\beta(s,a) = \sum_{p\in\mathcal{P}} (n_p^{\mathrm{E}}(s) + \delta_p^{\mathrm{E}}(a))\mu_p \cdot (1 + \delta^{\mathrm{C}}(a)z_p) + \sum_{p\in\mathcal{P}} \lambda_p \tag{8.7}$$

Next, we state formally the mappings to the subsequent states on an event of the two kinds. In the following, we use $\boldsymbol{e}_p$ for a unity vector of dimension $|\mathcal{P}|$ having a value of 1 at position p and zero for all other positions.

1. Arrival of an order – On arrival of a new order of type $p \in \mathcal{P}$ that is accepted the subsequent system state is given by

$$s^{\mathrm{A}}(s, \boldsymbol{\delta}^{\mathrm{E}}, p) = \left(\boldsymbol{n}^{\mathrm{W}}(s) + \boldsymbol{e}_p - \boldsymbol{\delta}^{\mathrm{E}}, \boldsymbol{n}^{\mathrm{E}}(s) + \boldsymbol{\delta}^{\mathrm{E}}\right) \tag{8.8}$$

   On arrival, an accepted order of type p becomes a project and is added to the waiting projects such that $n_p^{\mathrm{W}}(s^{\mathrm{A}}(s, \boldsymbol{\delta}^{\mathrm{E}}, p)) = n_p^{\mathrm{W}}(s) + 1$. Furthermore, a project of type $p' \in \mathcal{P}$ may be scheduled by decision a and remains scheduled on arrival of an new project such that $n_{p'}^{\mathrm{E}}(s^{\mathrm{A}}(s, \boldsymbol{\delta}^{\mathrm{E}}, p)) = n_{p'}^{\mathrm{E}}(s) + \delta_p^{\mathrm{E}}(a)$. In case that a project arrives that is rejected, the subsequent state given by

$$s^{\mathrm{R}}(s, \boldsymbol{\delta}^{\mathrm{E}}) = \left(\boldsymbol{n}^{\mathrm{W}}(s) - \boldsymbol{\delta}^{\mathrm{E}}, \boldsymbol{n}^{\mathrm{E}}(s) + \boldsymbol{\delta}^{\mathrm{E}}\right) \tag{8.9}$$

   is entered. In this case, the only change of the system state is due to scheduling of a project of type $p \in \mathcal{P}$.
2. Completion of a project – The subsequent state is given by

$$s^{\mathrm{E}}(s, \boldsymbol{\delta}^{\mathrm{E}}, p) = \left(\boldsymbol{n}^{\mathrm{W}}(s) - \boldsymbol{\delta}^{\mathrm{E}}, \boldsymbol{n}^{\mathrm{E}}(s) + \boldsymbol{\delta}^{\mathrm{E}} - \boldsymbol{e}_p\right) \tag{8.10}$$

   In this case, either the project already scheduled in state s is removed from the system or the project scheduled by decision a in state s. At the end of the transition as fixed payoff y_p is obtained.

8.1.5 Objective Function

In order to evaluate a given policy π, we consider the long term average reward per time unit as given by

$$g(\pi) = \liminf_{N\to\infty} \frac{E\left[\sum_{n=0}^{N-1} y(s_n, \pi(s_n), s_{n+1}) - \sum_{n=0}^{N} c(s_n, \pi(s_n))\tau_n\right]}{E\left[\sum_{n=0}^{N} \tau(s_n, \pi(s_n))\right]} \tag{8.11}$$

n denotes the number of the visited state, $\tau(s,a)$ is the transition time being exponentially distributed with rate $\beta(s,a)$ and $y(s,a,s')$ the fixed reward incurred on a transition from state s to state s'. On arrival of a new order of general project type $\phi \in \Phi$, we have $y(s,a,s') = -k^{\mathrm{A}}(\boldsymbol{\delta}^{\mathrm{PM}}(a),\phi)$, and on completion of a project of type $p \in \mathcal{P}$, we have $y(s,a,s') = y_p$. Furthermore, we assume holding cost w_p per time unit a project is in the system which corresponds to the average weighted flow time per project objective. Thus, the holding cost being linear in time serve as an approximation for the cost incurred due to not meeting due dates or due to delayed completion. Then, the cost rate $c(s,a)$ per time unit the system is in state s after selecting decision a is given by

$$c(s,a) = \sum_{p\in\mathcal{P}} w_p \left(n_p^{\mathrm{W}}(s) + n_p^{\mathrm{E}}(s)\right) + w^{\mathrm{C}}\delta^{\mathrm{C}}(a) \tag{8.12}$$

Then, an optimal policy π^* is given by

$$\pi^* = \arg\max_{\pi\in\Pi} \{g(\pi)\} \tag{8.13}$$

8.2 Solution Methodology

In this section, we address how the structure of the problem can be exploited in order to determine efficiently an optimal policy. At first, we make sure that there exists, for any given stationary policy π, a single $g(\pi)$ independent from the starting state of a sample path by establishing that the CTDMP is *unichain* (cf. Puterman [108]).

Theorem 8.2.1. *Under any stationary policy π performing OA and capacity planning decisions, the CTMDP is* unichain.

Proof. For the proof, it is sufficient to restrict the consideration to the scheduling decisions made by a policy. At first, we note that, by assuming non-idling policies (cf. Meyn [93]), the resource is always busy after having made a decision except the system is empty in state s^0. Hence, the resource is busy for all states $s \in \mathcal{S}\backslash\{s^0\}$. Furthermore, with non-zero probability, no new project arrives or is accepted until the system becomes empty such that system state s^0 is *accessible* from any system state $s \in \mathcal{S}\backslash\{s^0\}$. Conversely, under a policy π a subset $\mathcal{S}(\pi) \subseteq \mathcal{S}\backslash\{s^0\}$ is *accessible* from s^0 such that all $s \in \mathcal{S}(\pi)$ communicate with s^0 and $\mathcal{S}(\pi) \cup \{s^0\}$ is a *recurrent* class of states. Since s^0 is accessible from all *transient* states $s \in \mathcal{S}\backslash\mathcal{S}(\pi)$, it is the only recurrent class. □

We note that if we allow idleness the CTMDP is not necessarily unichain but weakly communicating as for some policies the CTMDP may be multichain (for details cf. Puterman[108]). Thus, we have modify the solution procedures. Furthermore, according to some preliminary tests, allowing idleness does not seem to have a benefit on the performance of optimal policies.

As the CTMDP is unichain under any policy $\pi \in \Pi$, the formulation of the optimality equations can be simplified as there exist a single $g(\pi)$ for policy π that is independent from the starting state of a sample path. Furthermore, the methodologies outlined in Chap. 4 can be applied without modifications.

Then, the set of optimality equations (Bellman equations) $\forall s \in \mathcal{S}$ is given by

$$h(s) = \max_{a \in \mathcal{A}(s)} \{Q(s,a)\} \, \forall s \in \mathcal{S} \tag{8.14}$$

where $Q(s,a)$ is the value of decision $a \in \mathcal{A}(s)$.

The optimal policy π^* is given by

$$\pi^*(s) = a^* = \arg \max_{a \in \mathcal{A}(s)} \{Q(s,a)\} \, \forall s \in \mathcal{S} \tag{8.15}$$

where the value $Q(s,a)$ of a decision a in state s is given by

$$Q(s,a) = Q^{\mathrm{R}}(s,a) + Q^{\mathrm{NR}}(s,a) \tag{8.16}$$

$Q^{\mathrm{R}}(s,a)$ is the value of a decision if $\delta^{\mathrm{C}} = 0$ and is given by

$$\begin{aligned} Q^{\mathrm{R}}(s,a) = &- \frac{\sum_{p \in \mathcal{P}} w_p \left(n_p^{\mathrm{W}}(s) + n_p^{\mathrm{E}}(s) \right) - g^*}{\beta(s,a)} - \sum_{p \in \mathcal{P}} \frac{\lambda_p}{\beta(s,a)} k^{\mathrm{A}}(\boldsymbol{\delta}^{\mathrm{M}}(a), \phi_p) \\ &+ \sum_{p \in \mathcal{P}} \frac{\lambda_p}{\beta(s,a)} \delta_p^{\mathrm{M}}(a) h\left(s^{\mathrm{A}}(s,a,p) \right) \\ &+ \sum_{p \in \mathcal{P}} \frac{\lambda_p}{\beta(s,a)} \left(1 - \delta_p^{\mathrm{M}}(a) h\left(s^{\mathrm{R}}(s,a) \right) \right) \\ &+ \sum_{p \in \mathcal{P}} \frac{(n_p^{\mathrm{E}}(s) + \delta_p^{\mathrm{E}}(a)) \mu_p}{\beta(s,a)} \left(y_p + h(s^{\mathrm{E}}(s,a,p)) \right) \end{aligned} \tag{8.17}$$

$Q^{\mathrm{NR}}(s,a)$ the change of the decision value due to δ^{C} and is given by

$$Q^{\mathrm{NR}}(s,a) = - \frac{w^{\mathrm{C}} \delta^{\mathrm{C}}(a)}{\beta(s,a)} + \sum_{p \in \mathcal{P}} \frac{(n_p^{\mathrm{E}}(s) + \delta_p^{\mathrm{E}}(a)) z_p \delta^{\mathrm{C}} \mu_p}{\beta(s,a)} \left(y_p + h(s^{\mathrm{E}}(s,a,p)) \right) \tag{8.18}$$

For the solution of (8.16), we can apply standard methodologies as presented in Sect. 4. However, the subproblem of determining $a^* \in \mathcal{A}(s); \forall s \in \mathcal{S}$ is difficult as we have infinite $|\mathcal{A}(s)|$. This is due to the continuous variable δ^{C}. In addition, we have $2^{|\mathcal{P}|}$ possible OA alternatives and at most $|\mathcal{P}|$ alternatives for scheduling

projects on the resource. Fortunately, we are able to exploit the structure of the optimality equations which leads to an efficient procedure for determining a^*.

At first, all scheduling decisions represented by $\boldsymbol{\delta}^{\mathrm{E}}$ need to be fully enumerated since all successor states $s^{\mathrm{A}}(s, \boldsymbol{\delta}^{\mathrm{E}}, p)$, $s^{\mathrm{R}}(s, \boldsymbol{\delta}^{\mathrm{E}})$ and $s^{\mathrm{E}}(s, \boldsymbol{\delta}^{\mathrm{E}}, p)$ depend on $\boldsymbol{\delta}^{\mathrm{E}}$.

The optimal values for $\boldsymbol{\delta}^{\mathrm{M}}$ given $\boldsymbol{\delta}^{\mathrm{E}}$ can be determined in two steps. In the first step, we determine the optimal order acceptance decisions $\boldsymbol{\delta}^{\mathrm{MPP}}$ for the case that MPP is regularly performed before OA using the following lemma.

Lemma 8.2.1. *Given $\boldsymbol{\delta}^{E}$ and the assumption that MPP is performed before OA, the optimal $\boldsymbol{\delta}^{MPP}$ for system state $s \in \mathcal{S}$ is obtained as follows*

$$\delta_p^{MPP} = \begin{cases} 0 & h(s^A(s, \boldsymbol{\delta}^E, p)) - h(s^R(s, \boldsymbol{\delta}^E)) \leq 0 \\ 1 & \textit{otherwise} \end{cases} \tag{8.19}$$

Proof. To remove $\beta(s, a)$ from the denominators, we apply *uniformization* where the CTMDP is transformed into an equivalent CTMDP by adding fictitious transitions to system state s without any cost such that a constant total rate c is attained (cf. Chap. 4). Thus, we obtain

$$h(s) = \frac{\max_{a \in \mathcal{A}(s)} \left\{ \tilde{Q}(s, a) \right\}}{c} + h(s) \tag{8.20}$$

where

$$\tilde{Q}(s, a) = \tilde{Q}^{\mathrm{R}}(s, a) + \tilde{Q}^{\mathrm{NR}}(s, a) \tag{8.21}$$

and

$$\begin{aligned} \tilde{Q}^{\mathrm{R}}(s, a) = &- g^* - \sum_{p \in \mathcal{P}} w_p \left(n_p^{\mathrm{W}}(s) + n_p^{\mathrm{E}}(s) \right) - \sum_{p \in \mathcal{P}} \lambda_p k^{\mathrm{A}}(\boldsymbol{\delta}^{\mathrm{M}}(a), \phi_p) \\ &+ \sum_{p \in \mathcal{P}} \lambda_p \delta_p^{\mathrm{M}}(a) \left(h\left(s^{\mathrm{A}}(s, \boldsymbol{\delta}^{\mathrm{E}}(a), p) \right) - h(s) \right) \\ &+ \sum_{p \in \mathcal{P}} \lambda_p \left(1 - \delta_p^{\mathrm{M}}(a) \right) \left(h\left(s^{\mathrm{R}}(s, \boldsymbol{\delta}^{\mathrm{E}}(a)) \right) - h(s) \right) \\ &+ \sum_{p \in \mathcal{P}} (n_p^{\mathrm{E}}(s) + \delta_p^{\mathrm{E}}(a)) \mu_p \left(y_p + h(s^{\mathrm{E}}(s, \boldsymbol{\delta}^{\mathrm{E}}(a), p)) - h(s) \right) \end{aligned} \tag{8.22}$$

$$\begin{aligned} \tilde{Q}^{\mathrm{NR}}(s, a) = &- w^{\mathrm{C}} \delta^{\mathrm{C}}(a) + \\ &\sum_{p \in \mathcal{P}} (n_p^{\mathrm{E}}(s) + \delta_p^{\mathrm{E}}(a)) z_p \delta^{\mathrm{C}}(a) \mu_p \left(y_p + h(s^{\mathrm{E}}(s, \boldsymbol{\delta}^{\mathrm{E}}(a), p)) - h(s) \right) \end{aligned} \tag{8.23}$$

By rearranging (8.22) according to general project types we obtain

$$\begin{aligned}\tilde{Q}^{\mathrm{R}}(s,a) = &- g^* - \sum_{\phi\in\Phi}\sum_{p\in\mathcal{P}(\phi)} w_p\left(n_p^{\mathrm{W}}(s) + n_p^{\mathrm{E}}(s)\right) - \sum_{\phi\in\Phi}\sum_{p\in\mathcal{P}(\phi)} \lambda_p k^{\mathrm{A}}(\boldsymbol{\delta}^{\mathrm{M}}(a),\phi)\\ &+ \sum_{\phi\in\Phi}\sum_{p\in\mathcal{P}(\phi)} \lambda_p \delta_p^{\mathrm{M}}(a)\left(h\left(s^{\mathrm{A}}(s,\boldsymbol{\delta}^{\mathrm{E}}(a),p)\right) - h(s)\right)\\ &+ \sum_{\phi\in\Phi}\sum_{p\in\mathcal{P}(\phi)} \lambda_p \left(1-\delta_p^{\mathrm{M}}(a)\right)\left(h\left(s^{\mathrm{R}}(s,\boldsymbol{\delta}^{\mathrm{E}}(a))\right) - h(s)\right)\\ &+ \sum_{\phi\in\Phi}\sum_{p\in\mathcal{P}(\phi)} (n_p^{\mathrm{E}}(s) + \delta_p^{\mathrm{E}}(a))\mu_p\left(y_p + h(s^{\mathrm{E}}(s,\boldsymbol{\delta}^{\mathrm{E}}(a),p)) - h(s)\right)\end{aligned} \tag{8.24}$$

such that we can decompose (8.24) into $|\Phi|$ Independent terms. For each general project type $\phi \in \Phi$ a term is given by

$$\begin{aligned}&- \frac{g^*}{|\Phi|} - \sum_{p\in\mathcal{P}(\phi)} w_p\left(n_p^{\mathrm{W}}(s) + n_p^{\mathrm{E}}(s)\right) - \sum_{p\in\mathcal{P}(\phi)} \lambda_p k^{\mathrm{A}}(\boldsymbol{\delta}^{\mathrm{M}}(a),\phi)\\ &+ \sum_{p\in\mathcal{P}(\phi)} \lambda_p \delta_p^{\mathrm{M}}(a)\left(h\left(s^{\mathrm{A}}(s,\boldsymbol{\delta}^{\mathrm{E}}(a),p)\right) - h(s)\right)\\ &+ \sum_{p\in\mathcal{P}(\phi)} \lambda_p \left(1-\delta_p^{\mathrm{M}}(a)\right)\left(h\left(s^{\mathrm{R}}(s,\boldsymbol{\delta}^{\mathrm{E}}(a))\right) - h(s)\right)\\ &+ \sum_{p\in\mathcal{P}(\phi)} (n_p^{\mathrm{E}}(s) + \delta_p^{\mathrm{E}}(a))\mu_p\left(y_p + h(s^{\mathrm{E}}(s,\boldsymbol{\delta}^{\mathrm{E}}(a),p)) - h(s)\right)\end{aligned} \tag{8.25}$$

which is independent from the terms of other general project types.

In the following, we show how an optimal $\boldsymbol{\delta}^{\mathrm{M}}$ given $\boldsymbol{\delta}^{\mathrm{E}}$ can be determined. As this involves a selection of a decision from those having the same values for $\boldsymbol{\delta}^{\mathrm{E}}$, we drop the decision a as parameter.

Rearranging (8.25) and removing the part that is independent from $\boldsymbol{\delta}^{\mathrm{M}}$ gives

$$-\lambda_\phi k^{\mathrm{A}}(\boldsymbol{\delta}^{\mathrm{M}},\phi) + \sum_{p\in\mathcal{P}(\phi)} \lambda_p \delta_p^{\mathrm{M}}\left(h\left(s^{\mathrm{A}}(s,\boldsymbol{\delta}^{\mathrm{E}},p)\right) - h\left(s^{\mathrm{R}}(s,\boldsymbol{\delta}^{\mathrm{E}})\right)\right) \tag{8.26}$$

With the assumption that MPP is always performed before OA, we determine an optimal $\boldsymbol{\delta}^{\mathrm{MPP}}$ (which replaces $\boldsymbol{\delta}^{\mathrm{M}}$). Then, (8.26) turns into

$$-\lambda_\phi k_\phi^{\mathrm{M}} + \sum_{p\in\mathcal{P}(\phi)} \lambda_p \delta_p^{\mathrm{MPP}}\left(h\left(s^{\mathrm{A}}(s,\boldsymbol{\delta}^{\mathrm{E}},p)\right) - h\left(s^{\mathrm{R}}(s,\boldsymbol{\delta}^{\mathrm{E}})\right)\right) \tag{8.27}$$

Thus, the term can be maximized by letting $\delta_p^{\mathrm{MPP}} = 1$ for any $p \in \mathcal{P}(\phi)$ if

$$h\left(s^{\mathrm{A}}(s, \boldsymbol{\delta}^{\mathrm{E}}(a), p)\right) - h\left(s^{\mathrm{R}}(s, \boldsymbol{\delta}^{\mathrm{E}})\right) > 0 \tag{8.28}$$

and $\delta_p^{\mathrm{MPP}} = 0$ otherwise. □

The difference $h(s^{\mathrm{A}}(s, \boldsymbol{\delta}^{\mathrm{E}}, p)) - h(s^{\mathrm{R}}(s, \boldsymbol{\delta}^{\mathrm{E}}))$ can be interpreted as the change of the long term expected total reward if on arrival an order of type $p \in \mathcal{P}$ is accepted. Thus, we accept a project of type $p \in \mathcal{P}$ only if the expected long term total reward is increased.

Then, we obtain $\boldsymbol{\delta}^{\mathrm{M}}$ from the following lemma.

Lemma 8.2.2. *Given $\boldsymbol{\delta}^{E}$, $\boldsymbol{\delta}^{MPP}$, the optimal $\boldsymbol{\delta}^{M}$ for system state $s \in \mathcal{S}$ is obtained as follows. For any $\phi \in \Phi$, we set $\forall\, p \in \mathcal{P}(\phi)$*

$$\begin{array}{ll} \delta_p^M = \delta_p^{MPP} & Q^M(s, \boldsymbol{\delta}^E, \boldsymbol{\delta}^{MPP}, \phi) > Q^{PM}(s, \boldsymbol{\delta}^E, \phi) \wedge Q^M(s, \boldsymbol{\delta}^E, \boldsymbol{\delta}^{MPP}, \phi) > 0 \\ \delta_p^M = 1 & Q^M(s, \boldsymbol{\delta}^E, \boldsymbol{\delta}^{MPP}, \phi) \leq Q^{PM}(s, \boldsymbol{\delta}^E, \phi) \wedge Q^{PM}(s, \boldsymbol{\delta}^E, \phi) > 0 \\ \delta_p^M = 0 & \textit{otherwise} \end{array} \tag{8.29}$$

where

$$Q^M(s, \boldsymbol{\delta}^E, \boldsymbol{\delta}^{MPP}, \phi) = \sum_{p \in \mathcal{P}(\phi)} \lambda_p \delta_p^{MPP} \left(h(s^A(s, \boldsymbol{\delta}^E, p)) - h(s^R(s, \boldsymbol{\delta}^E))\right) - \lambda_\phi k_\phi^M \tag{8.30}$$

$$Q^{PM}(s, \boldsymbol{\delta}^E, \phi) = \sum_{p \in \mathcal{P}(\phi)} \lambda_p \left(h(s^A(s, \boldsymbol{\delta}^E, p)) - h(s^R(s, \boldsymbol{\delta}^E))\right) - \lambda_\phi \min\left\{k_\phi^M, k_\phi^{PM}\right\} \tag{8.31}$$

Proof. We obtain $Q^{\mathrm{M}}(s, \boldsymbol{\delta}^{\mathrm{E}}, \boldsymbol{\delta}^{\mathrm{MPP}}, \phi)$ from (8.27) for given $\boldsymbol{\delta}^{\mathrm{MPP}}$ and $\boldsymbol{\delta}^{\mathrm{E}}$.

If we do not perform MPP before OA and $\delta_p^{\mathrm{M}} = 1\ \forall\, p \in \mathcal{P}(\phi)$ we obtain

$$-\sum_{p \in \mathcal{P}(\phi)} \lambda_p \min\left\{k_\phi^{\mathrm{M}}, k_\phi^{\mathrm{PM}}\right\} + \sum_{p \in \mathcal{P}(\phi)} \lambda_p \left(h\left(s^{\mathrm{A}}(s, \boldsymbol{\delta}^{\mathrm{E}}, p)\right) - h\left(s^{\mathrm{R}}(s, \boldsymbol{\delta}^{\mathrm{E}})\right)\right) \tag{8.32}$$

which corresponds to $Q^{\mathrm{PM}}(s, \boldsymbol{\delta}^{\mathrm{E}}, \phi)$. If $\delta_p^{\mathrm{M}} = 0\ \forall\, p \in \mathcal{P}(\phi)$ we obtain 0.

Thus, comparing the three alternatives gives the optimal order acceptance decision for general project type ϕ. □

$Q^{\mathrm{M}}(s, \boldsymbol{\delta}^{\mathrm{E}}, \boldsymbol{\delta}^{\mathrm{MPP}}, \phi)$ can be interpreted as the change of the expected long term total reward if MPP is performed before OA and order acceptance decisions are made for the orders of general type ϕ according to $\boldsymbol{\delta}^{\mathrm{MPP}}$. $Q^{\mathrm{PM}}(s, \boldsymbol{\delta}^{\mathrm{E}}, \phi)$ can be interpreted as the change of the expected long term total reward if orders of

general type ϕ are accepted and MPP is postponed. Thus, if $Q^{\mathrm{M}}(s, \boldsymbol{\delta}^{\mathrm{E}}, \boldsymbol{\delta}^{\mathrm{M}}, \phi) > Q^{\mathrm{PM}}(s, \boldsymbol{\delta}^{\mathrm{E}}, \phi)$ and $Q^{\mathrm{M}}(s, \boldsymbol{\delta}^{\mathrm{E}}, \boldsymbol{\delta}^{\mathrm{M}}, \phi) > 0$ MPP is performed before OA.

If $Q^{\mathrm{PM}}(s, \boldsymbol{\delta}^{\mathrm{E}}, \phi) \geq Q^{\mathrm{M}}(s, \boldsymbol{\delta}^{\mathrm{E}}, \boldsymbol{\delta}^{\mathrm{M}}, \phi)$ and $Q^{\mathrm{PM}}(s, \boldsymbol{\delta}^{\mathrm{E}}, \phi) > 0$ projects of general ϕ are accepted while MPP is postponed. Otherwise any project of general type ϕ will be rejected.

Next, δ^{C} is determined using the following lemma.

Lemma 8.2.3. *Given $\boldsymbol{\delta}^{M}$, $\boldsymbol{\delta}^{E}$ we have $\delta^{C} = 1$ if*

$$-w^{C} + \sum_{p \in \mathcal{P}} (n_p^E(s) + \delta_p^E) z_p \mu_p \left(y_p + h(s^E(s, \boldsymbol{\delta}^E, p) - h(s)) \right) > 0 \tag{8.33}$$

and $\delta^{C} = 0$ otherwise.

Proof. Given given $\boldsymbol{\delta}^{\mathrm{PM}}$, $\boldsymbol{\delta}^{\mathrm{E}}$, we observe that (8.23) is maximized by $\delta^{\mathrm{C}} = 1$ if

$$-w^{\mathrm{C}} + \sum_{p \in \mathcal{P}} (n_p^{\mathrm{E}}(s) + \delta_p^{\mathrm{E}}) z_p \mu_p \left(y_p + h(s^{\mathrm{E}}(s, \boldsymbol{\delta}^{\mathrm{E}}, p)) - h(s) \right) > 0$$

and $\delta^{\mathrm{C}} = 0$ otherwise. □

The left hand side of (8.33) can be interpreted as change of the long term expected total reward if non-regular capacity is fully used to crash the project in process. Thus, either non-regular capacity is used 100 % if the left hand side (8.33) is positive or 0 % otherwise. This corresponds to the behavior of a *bang bang* control (cf. Stidham and Weber [121]) where the optimal service rate for a system state is always one of the two extreme values from an interval of feasible values.

Thus, in order to determine the decision that minimizes $Q(s, a)$, we have to consider at most $|\mathcal{P}|$ possible scheduling decisions where for each scheduling decision, we determine optimal values w.r.t. the remaining decision variables. Thus, we have to evaluate $|\mathcal{P}|$ terms for the OA decisions with MPP before OA. Furthermore, we have $|\Phi|$ terms for the OA decisions with postponed MPP and $|\Phi|$ for the OA decisions with MPP before OA where each term in composed of $O(|\mathcal{P}|)$ terms. Finally, we evaluate one term for the decision concerning the usage of non-regular capacity. Then, for each scheduling decision, we have to compute the total value $Q(s, a)$ in order to compare the $|\mathcal{P}|$ scheduling decisions. In total, we have to evaluate $O(|\mathcal{P}|^2)$ terms. Even if we ignore the usage of non-regular capacity, the procedure is more efficient than full enumeration of all decisions as we would have to evaluate $O(|\mathcal{P}| \cdot 2^{|\mathcal{P}|})$ decisions.

Finally, the cardinality of the state space $|\mathcal{S}|$ that has a high impact on the computational burden and memory requirements is given by the following theorem.

Theorem 8.2.2.

$$|\mathcal{S}| = \begin{pmatrix} K^{\max} - 1 + |\mathcal{P}| \\ \mathcal{P}| \end{pmatrix} (|\mathcal{P}| + 1) \tag{8.34}$$

Proof. In order to compute $|\mathcal{S}|$, we ignore the order acceptance decisions and assume that any project is accepted until $K^{\max}$ is reached. This is justified by the fact under such a policy each state $s \in \mathcal{S}$ is accessible as long as $K^{\max}$ is not exceeded. This can be seen by considering the following scenario that may occur with non-zero probability. Starting with s^0, the first project of type $p \in \mathcal{P}$ with $n_p^{\mathrm{E}}(s) = 1$ arrives and is immediately scheduled as we require policies to be non-idling. Afterwards, we assume that no project is rejected as long as $K^{\max}$ is not reached. The project in process is not completed while $n_p^{\mathrm{W}}(s)$ projects arrive for each project type $p \in \mathcal{W}$. Thus, each state $s \in \mathcal{S}$ is accessible from s^0, and by completion of the project in project, each state $s' \in \mathcal{S}$ with $\boldsymbol{n}^{\mathrm{E}}(s') = \mathbf{0}$ is accessible from s^0 as well.

To simplify the analysis, we transform the semi-open system (cf. Buzacott and Shantikumar [25]) with a maximum number of projects in the system into an equivalent closed system with a constant number $K^{\max}$ of projects in the system by introducing the variable $n^{\mathrm{NIS}}(s) = K^{\max} - \sum_{p \in \mathcal{P}} (n_p^{\mathrm{W}}(s) + n_p^{\mathrm{E}}(s))$ for the projects not in the system. An interpretation is as follows. On completion, projects turns into generic orders without specific type that are queued in front of the system. Then, they wait for admission to the system at a rate $\lambda = \sum_{p \in \mathcal{P}} \lambda_p$ where the time between two admissions is exponentially distributed. On admission, a generic order turns into an order of type $p \in \mathcal{P}$ with probability $\frac{\lambda_p}{\lambda}$ which can be accepted or rejected.

Then, we obtain the cardinality of the state space in two steps. In the first step, we consider the states where no project is in process on the bottleneck resource. We observe that such states can only be entered subsequent to project completions. Thus, if we allow a maximum of $K^{\max}$ projects to be in the system at most $K^{\max} - 1$ projects may be waiting which are of the $|\mathcal{P}|$ project types. The rest of the projects is waiting for admission. Then, the number of states with no projects in process amounts to $\binom{K^{\max} - 1 + |\mathcal{P}|}{|\mathcal{P}|}$ which corresponds to the number of possibilities of distributing n identical items ($K^{\max} - 1$) among k ($|\mathcal{P}| + 1$) buckets as given by $\binom{n + k - 1}{k - 1}$. In order to obtain, in the second step, the number of states having a project in process, we observe that each state s with a project of any type $p \in \mathcal{P}$ in process has a transition to a state s' with $\boldsymbol{n}^{\mathrm{W}}(s') = \boldsymbol{n}^{\mathrm{W}}(s)$ and $\boldsymbol{n}^{\mathrm{E}}(s') = \mathbf{0}$ on completion of the project. Thus, s' must be accessible via one transition from $|\mathcal{P}|$ states having one project in process such that there number is $\binom{K^{\max} - 1 + |\mathcal{P}|}{|\mathcal{P}|} |\mathcal{P}|$.

Then, the sum of the number of states without a project in process and the number of states with one project in process gives the assertion. □

8.3 Computational Investigation

In this section, we investigate computationally the effect of the two sources of flexibility that are the option of postponing MPP and the option of using non-regular capacity for crashing.

For the analysis, we distinguish w.r.t. MPP between three cases.

1. Regular MPP – MPP is always performed *before* OA, except all projects of a general project type are rejected. For this case, we set k_{ϕ}^{PM} to a large value for all $\phi \in \Phi$ such that MPP is never postponed.
2. Postponed MPP – MPP is always performed *after* OA on arrival of a project such that acceptance/rejection decisions are only based on information from the general project type. For this case, we set k_{ϕ}^{M} to a large value for all $\phi \in \Phi$ such that MPP is always postponed.
3. Flexible MPP – Whether MPP is postponed or performed regularly is subject to decisions based on the system state.

Concerning the usage of non-regular capacity for crashing projets, we distinguish between two cases.

1. No crashing – Crashing using non-regular capacity is not allowed. To model this case, we set $z_p = 0 \ \forall p \in \mathcal{P}$ such that the usage of non-regular capacity has no benefit.
2. Crashing – Crashing using non-regular capacity is allowed.

The investigation is guided by two research questions.

1. What is the structure of optimal policies?
2. Under which system conditions it is worthwhile to consider postponing of MPP and usage of non-regular capacity?

By the first research question, we intend to find out whether and when policies have a simple structure which makes them more amenable to simple approximations. The second research question is aimed at finding out for which values of system parameters both sources of flexibility should be considered simultaneously. This implies a more complicated planning problem which can be simplified by omitting one or both aspects.

In the following, Sect. 8.3.1 is dedicated to the first research question and Sect. 8.3.2 to the second one.

8.3.1 Structure of Optimal Policies

As a starting point, we consider a problem instance referred to as *Base case 1* with two project types $p \in \mathcal{P}(\phi) = \mathcal{P}$ such that both are from the same general project type. The parameters are given in Table 8.1.

Table 8.1 Problem parameters for Base case 1

Parameter	Value
$(\overline{d}_1, \overline{d}_2)$	(0.6, 1.4)
(w_1, w_2)	(10, 10)
(y_1, y_2)	(200, 200)
(λ_1, λ_2)	(0.5, 0.5)
(z_1, z_2)	(0.429, 0.429)
k_1^{M}	10
k_1^{PM}	5
w^{C}	50

The two project types differ only w.r.t. their expected durations. Furthermore, we have set the expected durations and the arrival rates such that utilization u of the bottleneck resource, given by

$$u = \rho = \sum_{p \in \mathcal{P}} \lambda_p \overline{d}_p \tag{8.35}$$

has a value of 1. Note that u is the utilization without crashing if all orders were accepted.

Special attention is payed to differences and similarities to optimal policies obtained for related problems. The analysis is divided according to the three parts of decisions. Order acceptance, scheduling and usage of non-regular capacity.

8.3.1.1 Order Acceptance

Essentially, our problem is generalization of the problem considered by De Serres [42] such that we are confronted with the same difficulties when applying proof techniques for establishing structural results. Thus, in order to explore in more detail the structure of an optimal policy w.r.t. OA we use computational experiments instead. Figure 8.1 shows the optimal OA decisions for Base case 1 when no crashing is allowed and regular MPP or flexible MPP is used.

For the presentation of the policy, we have aggregated the system states according to the number of projects of each type in the system.

We can clearly identify *acceptance regions* highlighted by different shades of grey where projects of both types are accepted, only projects of type p_1 are accepted and no projects are accepted. We also obtain some regions indicated by the intermediate shades of grey where order acceptance decisions may differ between the states. For some states projects of both types are accepted and for some states projects only from one type are accepted.

Obviously, for regular MPP and for flexible MPP, the optimal policy is nearly monotonic w.r.t. OA decisions which is in line with the findings of De Serres [42]. Furthermore, we observe that the fact whether regular MPP or flexible MPP is used does not affect the structure of the optimal policy. However, for flexible

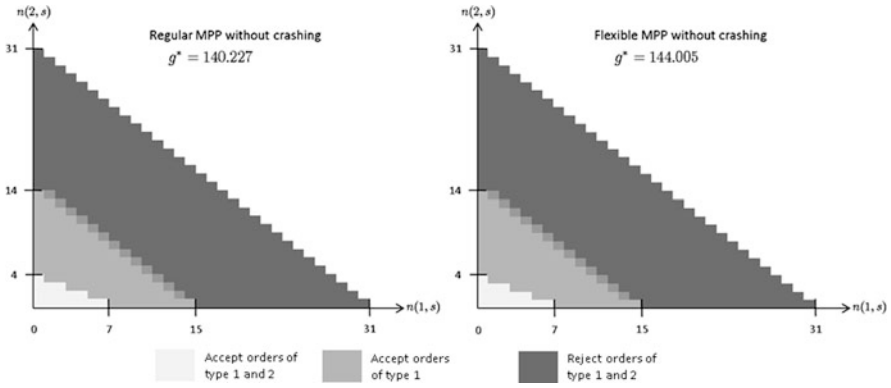

Fig. 8.1 Optimal order acceptance decisions depending on the number of projects in the system if no crashing is allowed

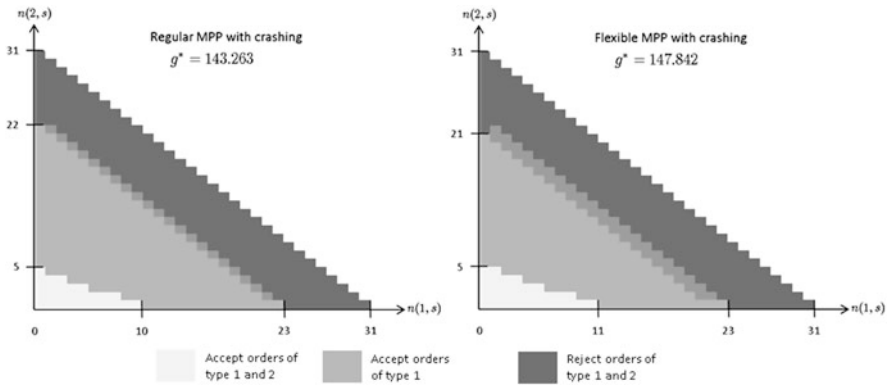

Fig. 8.2 Optimal order acceptance decisions depending on the number of projects in the system if crashing is allowed

MPP, decisions in the light grey region are always associated with postponed MPP. Thus, the main benefit from allowing flexible MPP results from the cost saved by postponed MPP when projects of both types are accepted (light grey region).

Next, we consider the optimal order acceptance decisions when crashing is allowed.

Figure 8.2 reveals that the optimal policies are again nearly monotonic and are less selective as the regions where projects are accepted become larger. Again, the structure is hardly affected by allowing flexible MPP.

In an extensive computational study, we could verify that also for other problem instances where project types differ w.r.t. payoffs, holding cost or crashing factors optimal policies are nearly monotonic w.r.t. order acceptance decisions.

Next, we consider another interesting property that is especially useful to narrow down the space of approximations of the optimal policy w.r.t. OA decisions. The

region where projects of both types are accepted is contained in the acceptance region for projects of type p_1 having a shorter expected duration. Next, we generalize the finding to cases where project types may differ w.r.t. other parameters. As a starting point, we could establish the following result for project types have different holding costs and payoffs.

Theorem 8.3.1. *Be given two project types $p_1, p_2 \in \mathcal{P}(\phi)$ for some $\phi \in \Phi$ with the following properties.*

- $y_{p_1} \geq y_{p_2}$
- $w_{p_1} \leq w_{p_2}$
- $\overline{d}_{p_1} = \overline{d}_{p_2}$
- *No crashing is possible ($z_{p_1} = z_{p_2} = 0$)*

Then, under an optimal policy, at each decision time where it is optimal to accept a project of type p_2 it is also optimal to accept a project of type p_1.

Proof. For the proof, we consider a finite stream of projects indexed by $j = 1, \ldots, J$ that arrive at the system. We represent a project by a tuple (j, p) and refer to the type of a project by $\boldsymbol{p}_j$. Then, the system state $s(t)$ at time t is given by $s(t) = (J^a, \mathcal{W}(t), \mathcal{E}(t))$ where J^a is the number of order that already have arrived at the system, $\mathcal{W}(t)$ is the set of waiting projects and $\mathcal{E}(t)$ the set of projects in process. The variable J^a is needed to account for the limited number of orders that arrive at the system. It is incremented by 1 if an order of any type arrives (irrespectively whether the order will be accepted or not) and as soon as $J^a = J$ no further order arrives. A decision $a \in \mathcal{A}(s)$ is given by

$$a = \left(\boldsymbol{\delta}^{\mathrm{M}}, \mathcal{B}\right) \tag{8.36}$$

where $\boldsymbol{\delta}^{\mathrm{M}}(a)$ is a vector of binary variables $\delta_p^{\mathrm{M}}(a)$ for the order acceptance decisions as defined in Sect. 8.1 and $\mathcal{B}(a)$ is the set of projects to be scheduled (as the bottleneck resource can process only one project at a time, the set contains at most one element). We define $s^{\mathrm{A}}(\mathcal{B}, s, p)$ to be the state that is entered on arrival of a project of type p if projects in $\mathcal{B}(a)$ are scheduled and $s^{\mathrm{R}}(\mathcal{B}, s)$ the state entered when a project is rejected and projects in $\mathcal{B}$ are scheduled. $V^*(s)$ refers to the future expected total reward if after state s an optimal policy is followed. Furthermore, we define $\boldsymbol{F}(j, s)$ to be a random variable for the remaining flow time of project j if the system is in state s and an optimal policy is followed.

For the proof, we assume that the system is in state s where j projects already have arrived and decision a^* is the optimal decision. As project types p_1 and p_2 are from the same general type, we can ignore, in the following analysis, fixed cost for doing regular or postponed MPP. Next, we assume that it is optimal to accept project $j + 1$ if it turns out to be of type p_2 such that $V^*(s^{\mathrm{A}}(\mathcal{B}(a^*), s, p_2)) - V^*(s^{\mathrm{R}}(\mathcal{B}(a^*), s)) > 0$. Then, the expected remaining flow time of project $j + 1$ is given by $E[\boldsymbol{F}(j + 1, s^{\mathrm{A}}(\mathcal{B}(a^*), s, p_2))]$.

Next, we consider the case where a project is accepted if it turns out to be of type p_1.

If decisions are made afterwards as if it were of type p_2 we must have $E[\boldsymbol{F}(j+1, s^{\mathrm{A}}(\mathcal{B}(a^*), s, p_1))] = E[\boldsymbol{F}(j+1, s^{\mathrm{A}}(\mathcal{B}(a^*), s, p_2))]$ as the same resource allocation and OA decisions are made and p_1 has the same distribution for its duration as p_2. We would have for the expected remaining holding cost

$$E\left[\boldsymbol{F}(j+1, s^{\mathrm{A}}(\mathcal{B}(a^*), s, p_2))\right] w_{p_1} \leq E\left[\boldsymbol{F}(j+1, s^{\mathrm{A}}(\mathcal{B}(a^*), s, p_2))\right] w_{p_2}$$

and for the payoffs $y_{p_1} \geq y_{p_2}$. Thus, the expected future reward would be

$$V^*(s^{\mathrm{A}}(\mathcal{B}(a^*), s, p_2)) - E\left[\boldsymbol{F}(j+1, s^{\mathrm{A}}(\mathcal{B}(a^*), s, p_2))\right](w_{p_1} - w_{p_2}) + (y_{p_1} - y_{p_2}) \geq V^*(s^{\mathrm{A}}(\mathcal{B}(a^*), s, p_2))$$

As, by assumption, an optimal policy is followed after state $s^{\mathrm{A}}(p_1, s)$ which makes optimal decisions we must have

$$V^*(s^{\mathrm{A}}(\mathcal{B}(a^*), s, p_1)) \geq V^*(s^{\mathrm{A}}(\mathcal{B}(a^*), s, p_2)) - E\left[\boldsymbol{F}(j+1, s^{\mathrm{A}}(\mathcal{B}(a^*), s, p_2))\right](w_{p_1} - w_{p_2}) + (y_{p_1} - y_{p_2})$$

Thus, the assertion follows that if it is optimal to accept a project of type p_2 it is also optimal to accept a project of type p_1. The assertion remains also valid for $J \longrightarrow \infty$. □

This, result implies that if the conditions hold the regions where projects of both types are accepted is contained in the acceptance region for p_1.

Unfortunately, as the distribution of the durations has an impact on state and transition probabilities of the CTMDP, we could not extend the result to the case where $\overline{d}_1 \leq \overline{d}_2$ and $z_{p_1} \geq z_{p_2}$. However, we could verify in a number of computational experiments that, at least for the cases we have tested, the result extends to expected durations and crashing factors. To summarize the findings we introduce the concept of dominance.

Definition 8.3.1. Be $p_1, p_2 \in \mathcal{P}(\phi)$ for some $\phi \in \Phi$ two project types with $\phi(p_1) = \phi(p_2)$. Then p_1 dominates pointwise p_2 if $\overline{d}_1 \leq \overline{d}_2$, $w_1 \leq w_2$, $y_1 \geq y_2$, $z_1 \geq z_2$ and at least one inequality is strict.

In short, we write $p_1 \succ_d p_2$ to say that p_1 dominates pointwise p_2.

Then, our conclusion from the experiments is the following: If $p_1 \succ_d p_2$ the region where projects of both types are accepted is contained in the acceptance region of p_1.

From a practitioners point of view, the monotonic structure of optimal order acceptance decisions may serve as a starting point to find good heuristic policies for order acceptance decisions. For example, as the boundaries of the acceptance

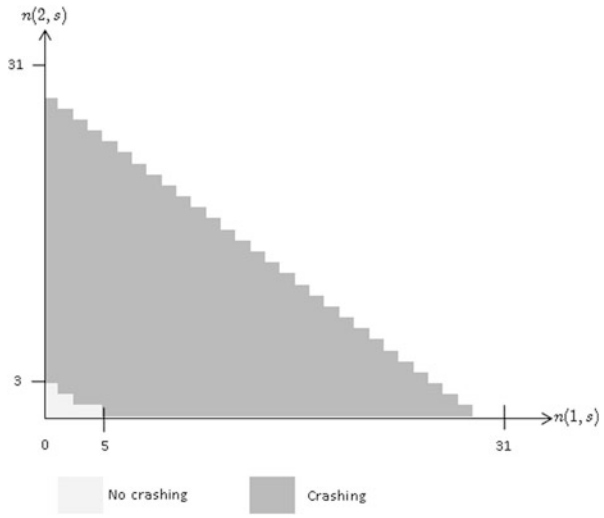

Fig. 8.3 Usage of non-regular capacity at states where project type 1 is scheduled

regions correspond almost to a linear function, *window flow controls* may be an option to approximate the optimal policy w.r.t. OA decisions (cf. De Serres [42]). Dominance relations between project types help to further reduce that search space of possible policies w.r.t. OA decisions.

8.3.1.2 Scheduling

For Base case 1 as well as for many other instances, the optimal policy corresponds w.r.t. scheduling decisions to the $c\mu$-policy. Except some cases where the priorities $w_p\mu_p$ are very similar (cf. De Serres [41] for an example where the $c\mu$-policy is not optimal) we could verify that *without* crashing the $c\mu$-policy delivers optimal scheduling decisions. If crashing is allowed the following condition should be met.

$$w_{p_1}\mu_{p_1} \geq w_{p_2}\mu_{p_2}(1 + z_{p_2}) \tag{8.37}$$

According to our observations, the optimal policy corresponds to a $c\mu$-policy if for two project types $p_1 > p_2$ the priority of a project of type p_2 can never exceed the priority of a project of type p_1 even if its crashed.

8.3.1.3 Usage of Non-regular Capacity

Next, we consider the usage of non-regular capacity when flexible MPP is allowed. Figure 8.3 shows the usage of non-regular capacity for states where no project is in process and it is optimal to schedule a project of type 1.

Table 8.2 Problem parameters for Base case 2

Parameter	Value
$(\overline{d}_1, \overline{d}_2)$	(1.0, 1.0)
(w_1, w_2)	(10, 10)
(y_1, y_2)	(200, 200)
(λ_1, λ_2)	(0.5, 0.5)
(z_1, z_2)	(0.429, 0.429)
k_1^{M}	10
k_1^{PM}	5
w^{C}	50

Obviously, the policy is monotonic w.r.t. usage of non-regular capacity. Non-regular capacity is used for crashing if the numbers of projects of each type waiting in the system exceed a certain number. The more projects are waiting in the system the higher are the holding cost due to waiting. Thus, the pressure to reduce holding cost by processing projects in less time increases which explains the structure of the policy. Obviously, for the case where only projects of type 2 are waiting the threshold is slightly lower (3) than for the case where more projects of type 1 are waiting. This can be explained by the longer expected duration of project type 2. As the flow times of such projects are expected to be longer the pressure to crash increases.

Monotonicity w.r.t. usage of non-regular capacity is typical and can also be observed for other parts of the state space where a project is already in process. Furthermore, we could verify in an extensive computational study that this monotonicity exists also for other problem instances.

The decision on the usage of non-regular capacity is a kind of service rate control. Then, the observation is similar to the results of Crabill [30] who has shown that for an $M/M/1$-system with a single project type the optimal policy has multiple thresholds on the number of projects in the system. If a threshold is exceeded a larger service rate is selected.

We conclude that the optimal policy has in many cases a simple structure w.r.t. capacity planning decisions which may guide the search for good heuristic policies.

8.3.2 Benefit of Crashing and Flexible MPP

In order to explore the benefit of the two sources of flexibility we carry out a ceteris paribus analysis with a base case referred to as *Base case 2* with two identical project types $p \in \mathcal{P}(\phi) = \mathcal{P}$ which are from the same general project type. The parameters are shown in Table 8.2.

Again, we have set the expected project durations and the arrival rates such that without crashing the utilization of the bottleneck resource has a value of 1 if all orders were accepted. We will briefly address the impact of utilization later.

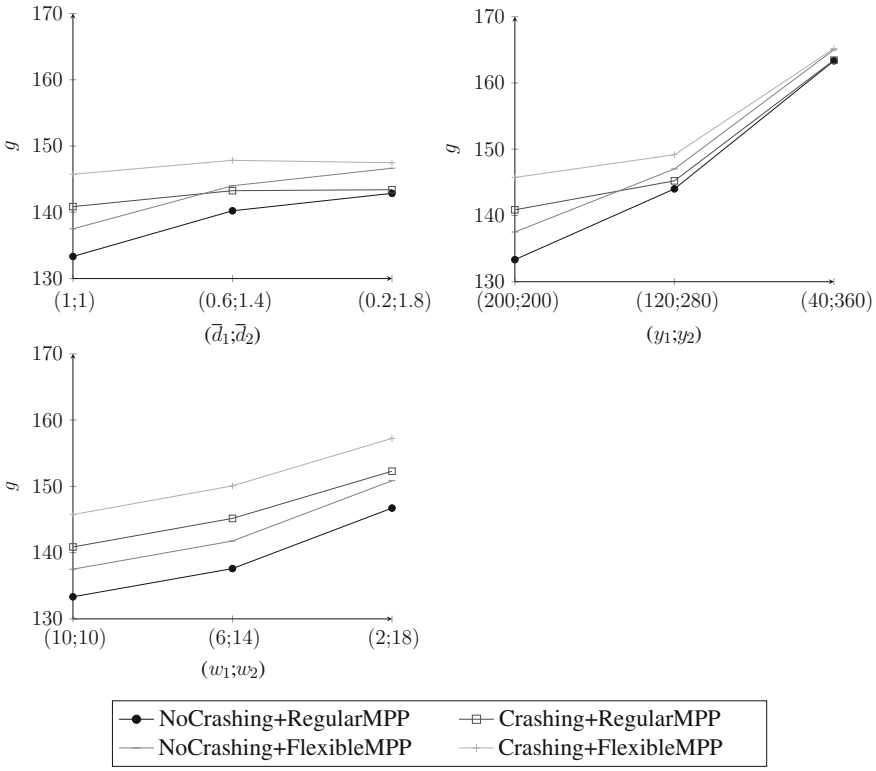

Fig. 8.4 Benefit of crashing

For the ceteris paribus analysis, we have considered scenarios where projects have different degrees of heterogeneity. Heterogeneity refers to the extend the values w.r.t. a parameter differ between the project types. For example, two project types with expected durations $(0.6, 1.4)$ are more heterogeneous that two project types having the same expected duration. For our analysis, we have set the expected durations $(\overline{d}_1, \overline{d}_2)$ to $(0.6, 1.4)$ and $(0.2, 1.8)$, the payoffs (y_1, y_2) to $(120, 280)$ and $(40, 360)$ and the holding costs per time unit (w_1, w_2) to $(6, 10)$ and $(2, 18)$.

8.3.2.1 Benefit of Crashing

In this section, we analyze the benefit of crashing using non-regular capacity. Figure 8.7 shows firstly the average reward for *regular MPP* without crashing and with crashing in the base case and different scenarios concerning the expected durations $(\overline{d}_1, \overline{d}_2)$, the payoffs (y_1, y_2) and the holding cost per time unit (w_1, w_2). Secondly, it shows the average reward for *flexible MPP* without crashing and with crashing in the different scenarios (Fig. 8.4).

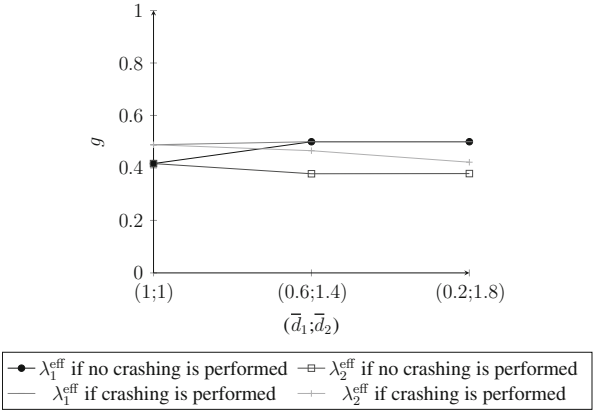

Fig. 8.5 Effective arrival rates with and without crashing in case of flexible MPP

We observe that crashing is most beneficial for the base case as we obtain the largest increase of the average reward. This holds for regular MPP as well as for flexible MPP. The larger the difference between the expected durations, payoffs or holding costs per time unit becomes the smaller becomes the increase of the average reward due to crashing. The observation can be explained by the fact that for more heterogenous projects the policy becomes more selective in the sense that projects of the type with the higher expected duration, lower payoff and higher holding cost per time unit are rejected more frequently. An indication delivers a consideration of the effective arrival rate λ_p^{eff} being the arrival rate of project type $p \in \mathcal{P}$ when rejections of projects are taken into account. Figure 8.5 shows the effective arrival rates for both project types at different levels of heterogeneity w.r.t. expected durations when crashing is performed or not and flexible MPP is applied.

We observe that crashing leads to an increase of the effective arrival rates. Due to shorter project durations, average holding costs decrease such that more projects can be accepted. Furthermore, for more heterogenous projects (implying a shorter the expected duration of project type 1), the effective arrival rate of project type 1 increases up to the maximum arrival rate of 0.5. Thus, crashing does not lead to a further increase of the effective arrival rate of project type 1. At the same time crashing leads to an increase of the effective arrival rate of project type 2. However, the increase becomes less the higher the expected duration of project type 2 becomes as crashing compensates the negative effect of a longer expected duration less well.

The observations made for the payoffs and the holding costs per time unit can be explained by a similar argument.

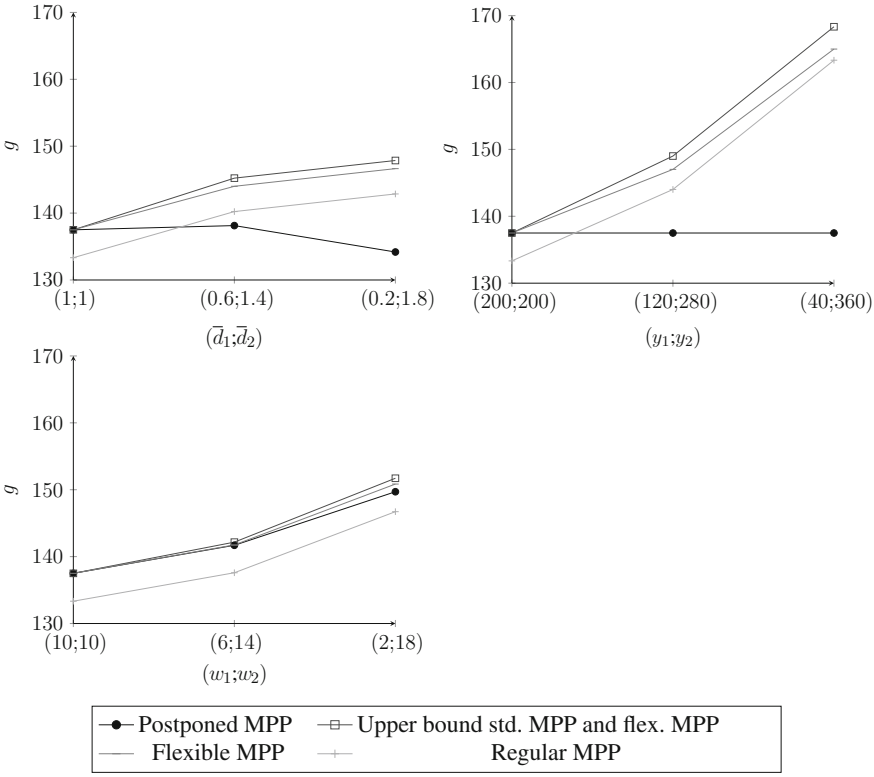

Fig. 8.6 Benefit of MPP without crashing

8.3.2.2 Benefit of MPP

In this section, we analyze the benefit of MPP in more detail. At first, we consider the effect of MPP when no crashing is performed. Figure 8.6 shows the average rewards for the different scenarios with postponed MPP, regular MPP and flexible MPP. In addition, we compute an upper bound for the average reward obtained from regular MPP and flexible MPP by setting $k_1^{\mathrm{M}} = k_1^{\mathrm{PM}} = 5$. As a consequence, the difference between the upper bound and the average reward for postponed MPP can be interpreted as the *maximum additional average reward resulting from regular MPP*.

At first, we make the intuitive observation that regular MPP becomes more beneficial for more heterogeneous projects as MPP helps to identify the project type which should be rejected. Furthermore, we observe that the average reward can be considerably increased and nearly attains the upper bound when flexible MPP is performed instead of regular MPP. However, the benefit of flexible MPP decreases for more heterogeneous projects which can be explained by the fact that a large part of the benefit is due to savings of the additional fixed cost $k_\phi^{\mathrm{M}} - k_\phi^{\mathrm{PM}}$ in case of

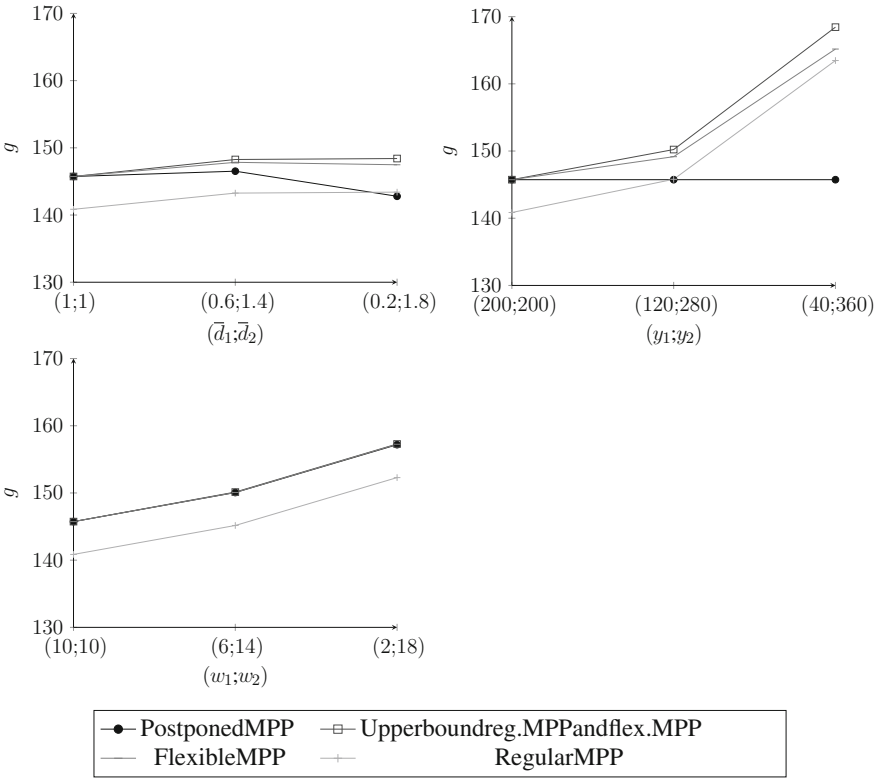

Fig. 8.7 Benefit of MPP with crashing

postponed MPP. The more heterogeneous projects become the less it is beneficial to postpone MPP.

Next, we consider the benefit of MPP when crashing is allowed as shown in Fig. 8.7.

We observe that, with crashing, the maximum additional average reward resulting from regular MPP decreases such that flexible MPP has a smaller benefit over postponed MPP. Furthermore, regular MPP is only beneficial over postponed MPP only for very heterogeneous projects in terms of expected durations or payoffs. The observation can be explained by the fact that crashing reduces project durations such that negative impact of longer projects is reduced.

8.3.2.3 Combined Benefit of MPP and Crashing

In this section, we analyze when it is worthwhile to consider flexible MPP and crashing simultaneously. Instead, the planning problem could be simplified by

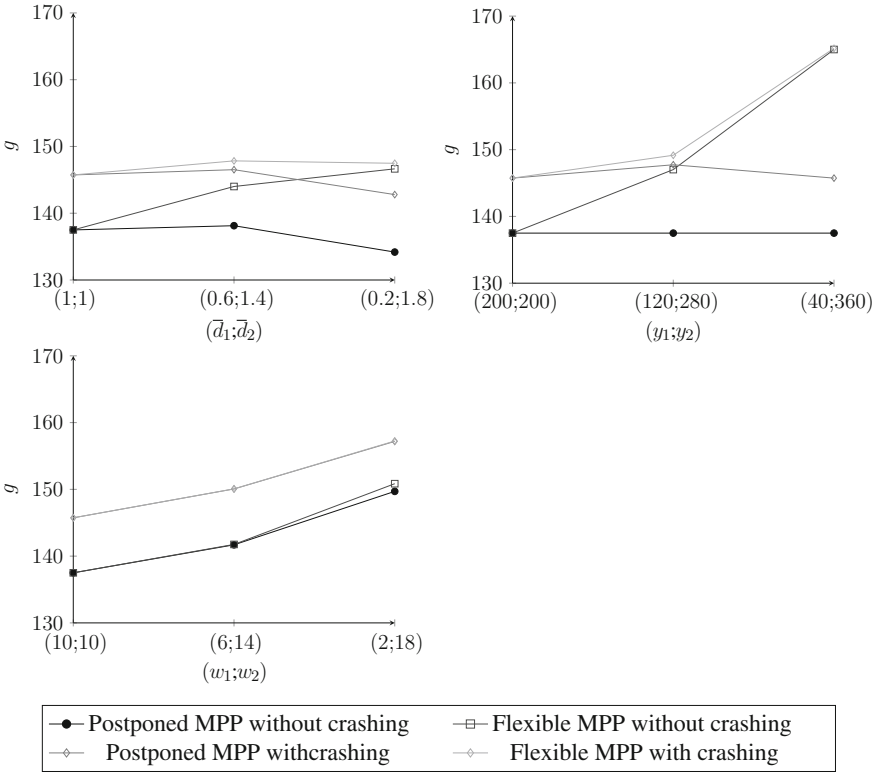

Fig. 8.8 Benefit of MPP and crashing for different expected durations

considering flexible MPP without crashing or crashing while MPP is postponed. The simplest problem is obtained when MPP is postponed and no crashing is allowed.[1]

Figure 8.8 shows the average rewards for different levels of heterogeneity w.r.t. expected durations for postponed MPP without crashing, flexible MPP without crashing, postponed MPP with crashing and flexible MPP and crashing.

We observe that a policy that uses flexible MPP and crashing performs best. Furthermore, for projects having the same expected durations its benefit over the policy with postponed MPP without crashing can be fully explained by usage of crashing. By contrast for projects having very heterogeneous expected durations its performance can be fully explained by the usage of flexible MPP.

This suggests that for similar projects of the same general type a policy that only takes into account crashing and postpones MPP is sufficient while for very

[1]Regular MPP is not considered in this analysis as it is dominated by flexible MPP and postponed MPP as seen in the previous section.

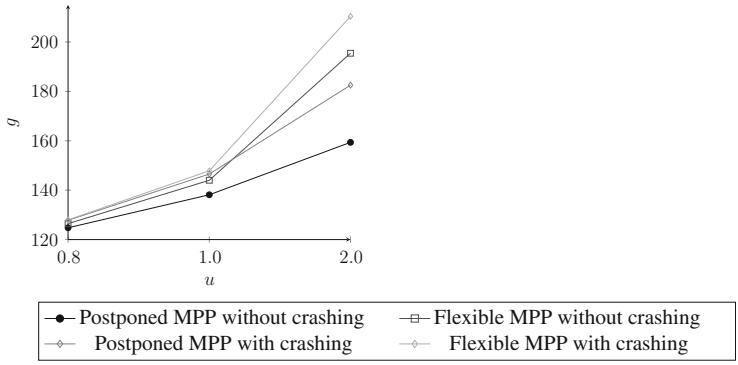

Fig. 8.9 Benefit of MPP and crashing for different levels of u

heterogeneous projects only flexible MPP without crashing should be taken into account.

Finally, we briefly address the effect of u which has been ignored so far. Figure 8.9 shows the effect for the base case when setting $(\overline{d}_1, \overline{d}_2) = (0.6, 1.4)$ when setting u to 0.8, 1.0 and 2.0.

It becomes obvious that for low levels of u, there is only a small benefit of MPP before OA and non-regular capacity for crashing. By contrast, for a high level of u, indicating an overly demanded system, the benefit of MPP and non-regular capacity become larger. In the face of higher demand the efficient usage of a scarce resource becomes more important.

8.3.2.4 General Insights

In the previous sections, observations have been made only for the ceteris paribus case where only one project parameter is varied at a time. Next, we draw some more general conclusions from the observations made in a more extensive computational study. As long as there is a dominance relation ($p_1 \succ_d p_2$ or $p_2 \succ_d p_1$) between two project types $p_1, p_2 \in \mathcal{P}(\phi)$ for some $\phi \in \Phi$ more heterogeneity w.r.t. project type parameters makes flexible MPP more beneficial while less heterogeneity makes crashing more beneficial.

Thus, in the intermediate regions where project type parameters are not extremely different a policy combining flexible MPP and crashing is most beneficial.

Next, we briefly address the effect when project types become more heterogeneous w.r.t. one parameter while no dominance relation exists. For the analysis we consider *Base case 1* where we have obviously a dominance relation as $\overline{d}_1 < \overline{d}_2$. Then, we firstly consider the scenario where have for the payoffs $(y_1, y_2) = (120, 280)$ such that there is no longer a dominance relation as $y_1 < y_2$ and $\overline{d}_1 < \overline{d}_2$. Secondly, we consider the scenario where $(y_1, y_2) = (280, 120)$ such that the dominance relation is strengthened. In order to study the benefit of flexible MPP

Table 8.3 Performance for different combinations of (y_1, y_2)

(y_1, y_2)	(200, 200)	(120, 280)	(280, 120)
Postponed MPP without crashing	138.132	138.132	138.132
Flexible MPP without crashing	144.005	138.133	154.900
Flexible MPP with crashing	147.842	146.523	155.067

and crashing we consider optimal policies with postponed MPP without crashing, flexible MPP without crashing and flexible MPP with crashing. Table 8.3 shows the average rewards for the base case and the two scenarios.

We observe that for the scenario where there is no dominance relation anymore flexible MPP has no benefit while crashing is most beneficial. This can be explained by the fact that the advantage of the shorter expected duration of project type 1 is neutralized in part by the smaller payoff y_1. Thus, both project types become more similar from an economic point of view.

Chapter 9
Conclusions and Future Work

In this chapter, we summarize, firstly, the results obtained from our research, and secondly point out possible options for future research.

After formal statements of the problems in Chap. 2 and a review of the relevant literature in Chap. 3, we have proposed in Chap. 5 a new procedure for the generation of problem instances with controlled parameters. The controlled parameters are related to the system such as number of resource types and to project types such as order strength or number of activity types.

The focus of Chap. 6 was on the performance of resource-based priority policies (RBPs) for scheduling under a weighted tardiness objective where preemptions are not allowed. The investigation was based on a simulation study where a number of problem parameters are controlled. We have found that the performance of RBPs strongly depends on the problem parameters such that no general recommendation, concerning the policy to be used, can be given. Thus, we have identified different scenarios where the ranking of the RBPs in terms of their performance has been relatively stable.

Chapter 7 was dedicated to the computation of optimal policies and near optimal policies for the scheduling problem with and without preemptions. To facilitate analysis, we have assumed a semi-open system where projects are rejected if the number of projects in the system exceeds a certain number. Furthermore, we have considered the weighted flow time instead of the weighted tardiness and fixed costs incurred when projects are rejected. As a first step, we have formulated two new models which represent both problems by continuous-time Markov decision processes (CTMDPs). For both models, optimal policies can be obtained using standard solution methodologies as outlined in Chap. 4. Furthermore, we have shown how efficiency can be improved and have characterized complexity in terms of state space cardinality. Then, a numerical example was given to point out important features of optimal policies. However, as the cardinalities of the state space becomes very large, especially when there are only few precedence relations and preemptions are not allowed, optimal policies can be obtained only for small examples. Thus, as a second step, we have proposed, based on the structural properties observed

P. Melchiors, *Dynamic and Stochastic Multi-Project Planning*, Lecture Notes in Economics and Mathematical Systems 673, DOI 10.1007/978-3-319-04540-5_9,

for optimal policies, a new policy class, namely the class of project state ordering policies (POPs), which helps to effectively reduce state space cardinality without much loss of performance. To remedy further the curse of dimensionality, we consider, in a third step, approaches of approximate dynamic programming. The first approach addresses large state spaces when preemptions are not allowed by exploiting the fact that the state space becomes much smaller when preemptions are allowed. The idea is to approximate the value function using the value function of the optimal policy for the problem with preemptions. The second approach is to approximate the value function using approximation architectures having a compact representation. The major advantages of using approximation architectures are the fact that the analysis carried out for semi-open systems can be extended to open systems without a limit on the number of projects in the system and that larger problem instances with many activity types and project types can be considered. We have proposed a number of approximation architectures and have considered a number of approaches for determining the free parameters. In a large computational study with controlled problem parameters we investigated the benefit of optimal policies over RBPs as considered in Chap. 6 and the performance of POPs and their potential to reduce state space cardinalities. We have found that optimal policies may have considerable benefits over RBPs. Furthermore, restricting the consideration to POPs does not mean a loss of performance in many cases while state space cardinality can be dramatically reduced. Thus, optimal policies can be obtained for a larger range of problem instances. For the non-preemptive problem, we additionally have investigated the performance of the value function approximation based on the value function for the preemptive problem. We have found, that, for many cases, near optimal policies can be obtained which clearly outperform RBPs. As we have numerous approximation architectures and algorithmic options for determining the free parameters, we carried out a computational study in order to narrow down the number of alternatives in terms of approximation architectures and algorithms to the most promising ones. Afterwards, we used a number of case studies to demonstrate the performance of policies obtained from approximation architectures. We have found that such policies have the potential to clearly outperform RBPs.

In Chap. 8, we have considered the joint optimization of order acceptance (OA) decisions and capacity planning decisions comprising scheduling decisions and decisions concerning the usage of non-regular capacity. An additional aspect, which has been neglected so far in the literature, is the option to postpone macro process planning where important information about an order is determined to a time after OA. As a starting point, we have presented a new tactical model based on a CTMDP for which decisions can be determined efficiently. Based on the model, we investigated the structure of optimal policies. We have found that optimal policies have in many cases a simple structure which may guide the search for simple heuristic policies. Furthermore, we considered the benefit of the option to postpone MPP or the usage of non-regular capacity in the model. We have identified conditions where one or both aspects can be neglected such that model and the corresponding planning problem can be simplified. Future work may refer to different issues. Firstly, the effect of different probability distributions

for interarrival times and activity durations on the performance of priority policies should be investigated. First tests have revealed that, essentially, the results remain valid also for other distributions. Secondly, optimal and near optimal policies that have been obtained under the assumption of the exponential distribution should be tested with other distributions as well. Again, preliminary experiments have shown that the policies may perform well also for other distributions. Thirdly, the models could be generalized in order to take into account further properties of R&D-projects such as iterations in case of failures. Finally, the investigation of order acceptance and capacity planning can extended to multiple bottleneck resources.

A
Abbreviations

ADP	Approximate dynamic programming.
BE	Bellman error.
BD	Bottleneck dynamics.
BD-MC	Bottleneck dynamics with myopic activity costing.
BD-GC-U	Bottleneck dynamics with global activity costing and uniform resource pricing.
BD-GC-U	Bottleneck dynamics with global activity costing and dynamic resource pricing.
CR	Critical ratio.
CRN	Common random numbers.
CV	Coefficient of variation.
CTMC	Continuous-time Markov chains.
CTMDP	Continuous-time Markov decision process.
FCFS	First-come first-served.
GC	Global activity costing.
LS	Least square.
MAXPEN	Maximum penalty.
MC	Myopic activity costing.
MDP	Markov decision process.
MPP	Macro process planning.
OA	Order acceptance
OS	Order strength.
PD	Product development.
PI	Policy iteration.
PO	Project state ordering.
POP	Project state ordering policy.
RAN	Random.
RNG	Random number generator.
RCCP	Rough cut capacity planning.
RCPSP	Resource-constrained project scheduling problem.

P. Melchiors, *Dynamic and Stochastic Multi-Project Planning*, Lecture Notes in Economics and Mathematical Systems 673, DOI 10.1007/978-3-319-04540-5,

RBP	Resource-based priority policy.
RBPOP	Resource-based project state ordering policy.
SASP-DD	Due-date modified shortest activity from shorted project.
VI	Value iteration.
W(CR + SPT)	Weighted critical ratio and shortest processing time.
WEDD	Weighted earliest due date.
WMINSLK	Weighted minimum slack.
WSPT	Weighted shortest processing time.

B
Symbols

B.1 General

B.1.1 System

$\mathbb{1}\{\cdot\}$	Indicator function being 1 if the condition $\cdot$ is met. Otherwise it is 0.
$K^{\max}$	Maximum number of projects which are allowed to be in the system.
λ	Total arrival rate.
t	Current system time.

B.1.2 Markov Decision Processes

$\mathcal{A}(s)$	Set of alternative decisions in system state s.
a	Decision.
$\beta(s,a)$	Transition rate resulting from decision a in system state s.
$\mathcal{C}$	Closed set (class) of states.
c	Uniformization constant.
$c(s,a)$	Cost rate per time unit resulting from decision a in system state s.
$\tilde{c}(s,a)$	Expected transition cost resulting from decision a in system state s in the uniformized CTMDP.
e	Stochastic event.
$g(\pi)$	Long term average cost incurred per time unit under policy π.
$g^{\mathrm{o}}(\pi)$	Long term average cost incurred per time unit under policy π for an open system.
$\tilde{g}(\pi)$	Long term average cost incurred per transition under policy π in the uniformized CTMDP.
g^*	Long term average cost incurred per time unit under an optimal policy π^*.

P. Melchiors, *Dynamic and Stochastic Multi-Project Planning*, Lecture Notes in Economics and Mathematical Systems 673, DOI 10.1007/978-3-319-04540-5,

$\tilde{g}^*$	Long term average cost incurred per transition under an optimal policy π^* in the uniformized CTMDP.
$H(s)$	Hash function.
$h(s)$	Relative value function depending on system state s.
$k(s, a, s')$	Additional fixed cost incurred on a transition from system state s to system state s' after making decision a.
$k^{\mathrm{A}}(a, p)$	Fixed cost incurred on arrival of a project of type p if decision a is selected.
ω	Sample path.
π	Policy.
$\pi(s)$	Decision from policy π in system state s.
Π	Set of all stationary policies.
$Succ(s)$	Position of the next state having the same hash key $H(s)$.
s	System state.
$\tau(s, a)$	Transition time after system state s if decision a is selected.
$V(s)$	Estimate for relative value function for system state s.

B.1.3 Projects and Project Types

$\mathcal{A}_p$	Set of precedence relations between the activity types of project type p.
$\overline{D}_p^{\max}$	Mean maximum flow time of project type p.
$\boldsymbol{D}_j^{\max}$	Maximum flow time of project j.
$\mathbf{d}_{ij}$	Duration of activity i of project j.
$\overline{d}_{ip}$	Expected duration of activity type (i, p).
$\overline{d}_p$	Expected total workload of project type p.
$\boldsymbol{\delta}_j$	Time between the arrivals of project $j-1$ and j.
$\boldsymbol{F}_j$	Flow time of project j.
$\mathcal{G}_p$	Graph representing the network of project type p with nodes representing activity types and arcs representing precedence relations.
i	Index for activities and activity types.
j	Index for projects.
(i, j)	Activity i of project j.
(i, p)	Activity type i of project type p.
$\mathcal{J}(t)$	Set of projects in the system at time t.
λ_p	Arrival rate of project type p.
μ_{ip}	Service rate of activity type (i, p).
OS_p	Order strength of the network of project type p.
$\mathcal{P}$	Set of project types.
p	Index for project types.
$\boldsymbol{p}_j$	Type of project j.
r_{ip}	Resource demanded by activity type (i, p).
$\boldsymbol{t}_j^{\mathrm{a}}$	Arrival time of project j at the system.

t_j^{x}	Completion time of project j at the system.
t^{next}	Next decision time.
$\mathcal{V}_p$	Set of activities types of project type p.
$\mathcal{V}_p^{\mathrm{Pred}}(i)$	Set of immediate predecessors of activity type (i, p).
$\mathcal{V}_p^{\mathrm{Succ}}(i)$	Set of immediate successors of activity type (i, p).
$\mathcal{V}_{pr}$	Set of activities types of project type p to be processed on resource type r.
$\mathcal{W}(j,t)$	Set of waiting activities of project j at time t.
w_p	Weight/cost per time unit of project type p.
y_p	Rejection cost/payoff of project type p.

B.1.4 Resources and Resource Types

c_r	Number of resources of resource type r.
$\overline{\boldsymbol{d}}_r$	Expected duration of any activity arriving at resource type r.
$CV_{\overline{\boldsymbol{d}}_r}$	Coefficient of variation related to the expected durations of the activities processed on resource type r.
$\mathcal{E}(r,t)$	Set of activities in process on resource type r at time t.
ρ_r	Traffic intensity for resource type r.
r	Index for resource types.
$\mathcal{R}$	Set of resource types.
$\mathcal{W}(r,t)$	Set of activities waiting for resource type r at time t.
u_r	Utilization of a resource of type r.

B.2 Generation of Problem Instances

B.2.1 Problem Parameters

α_p	Percent of tardy projects of project type p.
a_p	Fraction of the total arrival rate for project type p.
β	Parameter for the variation of the maximum flow times.
$CV^{\overline{d},\min}$	Minimum value for the coefficient of variation related to expected durations.
$CV^{\overline{d},\max}$	Maximum value for the coefficient of variation related to expected durations.
δ_{ij}	Binary variable indicating that there exists a path in the AoN–network from node i to j.
ϵ^{wi}	Tolerance parameter for workload proportions.
$\lambda^{\max}$	Maximum total arrival rate.
ν_r	Utilization multiplier of resource type r.

N^{PI}	Number of problem instances to be generated with given parameters.
n_p^{nw}	Number of the network sample of project type p.
OS_p	Order strength of the network for project type p.
$\lvert\mathcal{R}\rvert$	Number of resource types.
sd	Seed for the RNG used in the generation process.
u	Utilization per resource.
$\mathcal{V}$	Set of nodes.
$\lvert\mathcal{V}_p\rvert$	Number of nodes of the network for project type p.
wi_p	Workload index for project type p.

B.2.2 Generation Procedure

a_r	Expected number of activities to be processed by resource type r due to a project of any type having entered the system.
$\tilde{d}_{ip}$	Temporary value for $\overline{d}_{ip}$.
ρ_{ip}	Traffic intensity due to activity type (i, p).
$\tilde{\rho}_{ip}$	Temporary value for ρ_{ip}.
ρ_{ip}	Traffic intensity due to activity type (i, p).
ρ_r	Traffic intensity for resource type r.
$\tilde{\rho}_r$	Temporary value for ρ_r.

B.3 Scheduling

B.3.1 General

$\boldsymbol{s}_{ij}$	Start time of activity (i, j).
$\mathcal{B}^{\mathrm{S}}(r, t)$	Set of activities scheduled on resource type r at time t.
$\mathcal{B}^{\mathrm{B}}(r, t)$	Set of activities preempted on resource type r at time t.

B.3.2 Scheduling Using Priority Policies

$\overline{D}_p^{\mathrm{CP}}$	Expected length of the critical path of project type p.
$\overline{\overline{\boldsymbol{d}}}_r(t)$	Average expected duration of the activities waiting for resource type r at time t.
κ	Lookahead parameter.
$\overline{l}_{ij}$	Expected latest start time of activity (i, j).
$\overline{l}_{ij}^{\mathrm{CP}}$	Expected latest start time of activity (i, j) where the due date is determined based on the expected critical path length.

$\pi_{ij}(t)$	Marginal opportunity cost of activity (i, j) at time t.
$\pi_r(t)$	Price for resource type r at time t.
$\pi_r^{\mathrm{U}}(t)$	Price for resource type r at time t estimated using uniform resource pricing.
$\pi_r^{\mathrm{D}}(t)$	Price for resource type r at time t estimated using dynamic resource pricing.
$\overline{\boldsymbol{R}}^{\mathrm{n}}(\pi)$	Mean rank of policy π.
$\boldsymbol{t}_{ij}^{\mathrm{a}}$	Arrival time of activity (i, j) at resource type $r_{i\boldsymbol{p}_j}$.
$\boldsymbol{U}_{ij}(t)$	Urgency factor of activity (i, j) at time t.
$\boldsymbol{Z}(\pi)$	Average weighted tardiness under policy π.
$\boldsymbol{Z}^{\mathrm{n}}(\pi)$	Normalized average weighted tardiness under policy π.
$\overline{\boldsymbol{Z}}^{\mathrm{n}}(\pi)$	Mean normalized average weighted tardiness under policy π.

B.3.3 Markov Decision Process for the Non-preemptive Problem

$\mathcal{A}(r, s)$	Set of all feasible activity sets that can be scheduled on resource type r in system state s.
$\mathcal{B}^{\mathrm{S}}(k, s, a)$	Set of all activities from project k scheduled by decision a in system state s.
$\mathcal{E}(\sigma)$	Set of activities in process of a project in state σ.
$f(r, s)$	Number of idle resources of type r in system state s.
$K(s)$	Number of projects in the system in system state s.
$k^{\mathrm{A}}(s, p)$	Fixed cost incurred on arrival of a project of type p when the last pre-decision state is s.
$n(\sigma, s)$	Number of projects of project state σ in system state s.
$n^{\mathrm{Tr}}(\sigma, s)$	Truncated number projects of project state σ in system state s.
$\hat{n}(\sigma, s, a)$	Number of projects in project state σ while the system is in the post-decision state resulting from decision a in system state s.
$\pi^{*,y}$	Optimal policy computed for the case with rejection cost y for all project types.
$p(\sigma)$	Type of a project in project state σ.
σ	Project state.
$\sigma(j, t)$	State of project j at time t.
$\sigma(k, s)$	State of project k in system state s.
σ_p^{I}	Initial project state of a project of type p immediately after project arrival.
σ_p^{F}	Absorbing project state of a project of type p entered on completion of the last activity.
$\sigma^{\mathrm{C}}(i, \sigma)$	Project state entered from σ on completion of activity $i \in \mathcal{E}(\sigma)$.
$\sigma^{\mathrm{S}}(\mathcal{B}, \sigma)$	Project state entered from σ subsequent to scheduling all activities given by index set $\mathcal{B}$.

$\Sigma(p,s)$	Set of states of the projects of type p that are in the system at state s.
Σ	Set of all project states.
$\mathcal{S}(\pi)$	Set of system states that is accessible from s^0 under policy π.
s	System state.
s^0	System state with no projects in the system.
$s^{\mathrm{Tr}}(s)$	System state based on truncated numbers of project states from s.
$s^{\mathrm{C}}(s,a,i,\sigma)$	System state entered on completion of activity i of a project in state σ subsequent to decision a in system state s.
$\mathcal{W}(\sigma)$	Set of waiting activities of a project in state σ.
$\mathcal{W}^{\mathrm{R}}(i,\sigma)$	Set of activities of a project in state σ that become ready for execution on completion of activity i.

B.3.4 Markov Decision Process for the Preemptive Problem

$\mathcal{C}(r)$	Set of tuples (i,σ) with $r_{ip(\sigma)} = r$ and $\sigma \in \Sigma(s)$.
$n(i,\sigma)$	Number of activities i of projects in state σ to be scheduled.
$n(a,i,\sigma)$	Number of activities i of projects in state σ to be scheduled by decision a.
$q^{\mathrm{A}}(s,a,p)$	Probability of a transition in system state s on arrival of a project of type p if decision a is selected.
$q^{\mathrm{C}}(s,a,i,\sigma)$	Probability of a transition in system state s on completion of activity i of a project in state σ if decision a is selected.
$Q(s,i,\sigma)$	Value indicating the change of the long term expected total costs by scheduling activity i of a project in state σ.
$\sigma^{\mathrm{CPW}}(i,\sigma)$	Project state entered from σ on completion of activity $i \in \mathcal{W}(\sigma)$.
σ^{NIS}	State of the projects that are not in the system.
Σ^{P}	Set of all project states that may occur in pre-decision states of the preemptive problem.
$\mathcal{S}^{\mathrm{P}}$	Set of system states that may occur at decision times.
$s^{\mathrm{AP}}(s,p)$	System state entered from s on an arrival of a project of type p.
$s^{\mathrm{CP}}(s,i,\sigma)$	System state entered from s on completion of activity i of a project in state σ.
$\mathcal{W}^{\mathrm{R}}(i,\sigma)$	Set of activities of a project in state σ that become ready for execution on completion of activity i.

B.3.5 Optimal Policy for the Non-preemptive Problem with a Single Resource

$B(\omega,t)$	Cumulative distribution function for the duration of jobs of class ω.
$\gamma(\omega,\Omega_l)$	Expected remaining processing time until a job of class ω leaves the set Ω_l of job classes.

$\gamma(\sigma, \Sigma_l)$	Expected remaining processing time until a project in state σ leaves the set Σ_l of project states.
λ_ω	Arrival rate of job class ω.
$n(\omega, t)$	Length of the queue for jobs of class ω.
Ω	Set of all job classes.
Ω_l	Subset of Ω obtained from iteration l.
Ω_l^*	Subset of Ω_l where the elements in Ω_l^* have the minimum fraction $\frac{w_\omega}{\gamma(\omega, \Omega_l)}$ from those in Ω_l.
ω	Index for job classes.
$p(\omega, \omega')$	Probability that on completion a job of class ω becomes a job of class ω'.
$p(\sigma, \sigma')$	Probability that on completion of an activity a project in state σ becomes a project in state σ'.
Σ_l	Subset of Σ obtained from iteration l.
Σ_l^*	Subset of Σ_l where the elements in Σ_l^* have the minimum fraction $\frac{w_p(\sigma)}{\gamma(\sigma, \Sigma_l)}$ from those in Σ_l.
w_ω	Holding cost per time unit a job class ω is in the system.

B.3.6 Preemptive Project State Ordering Policies

$n(i, p, s)$	Number of activities of type (i, p) that are waiting in system state $s \in \mathcal{S}^{\mathrm{P}}$.
$n(i, p, s^{\mathrm{Q}})$	Number of activities of type (i, p) that are waiting in system state $s^{\mathrm{Q}} \in \mathcal{S}^{\mathrm{Q}}$.
Π^{PO}	Set of all stationary preemptive POPs.
$\sigma(\mathcal{W}, p)$	Project state obtained by the combination of the set of waiting activities $\mathcal{W}$ and project type p.
$\sigma^{\mathrm{o}}(l, p, s)$	Project state having the l-th rank in the order defined on the set $\Sigma(p, s)$.
$\mathcal{S}^{\mathrm{PPO}}$	Set of system states (state space) if only POPs are considered for the preemptive problem.
$\mathcal{S}^{\mathrm{Q}}$	Set of all system states (state space) of a queueing network.
s^{Q}	System state of a queueing network.
s_l^{Q}	System state of a queueing network after the l-th iteration of m^{-1}.
$\sigma_{lp}^{\min}$	Least advanced project state from system state s_l^{Q} obtained in the lth iteration of m^{-1}.
$\mathcal{W}(\mathcal{U}, p)$	Set of activities in $\mathcal{U}$ of which no predecessors are in $\mathcal{U}$.
$\mathcal{W}(p, s^{\mathrm{Q}})$	Index superset of waiting activities related to projects of type p in system state $s^{\mathrm{Q}} \in \mathcal{S}^{\mathrm{Q}}$.
$\mathcal{W}(p, s)$	Index superset of waiting activities related to projects of type p in system state $s \in \mathcal{S}^{\mathrm{PPO}}$.
$\mathcal{U}(\sigma)$	Set of unfinished activities of a project in state σ.

$\mathcal{U}(\mathcal{W}, p)$	Set of all direct and indirect successors of the activities in the set $\mathcal{W}$.
$\mathcal{U}(p, s^{\mathrm{Q}})$	Index superset of unfinished activities related to projects of type p in system state $s^{\mathrm{Q}} \in \mathcal{S}^{\mathrm{Q}}$.
$\mathcal{U}(p, s)$	Index superset of unfinished activities related to projects of type p in system state $s \in \mathcal{S}^{\mathrm{PPO}}$.

B.3.7 Non-preemptive Project State Ordering Policies

$\mathcal{C}(\sigma_1, \sigma_2)$	Set of activities waiting or in process that two projects in the states σ_1 and σ_2 have in common.
$\mathcal{E}_1^{\mathrm{C}}(\sigma_1, \sigma_2)$	Activities in $\mathcal{E}(\sigma_1)$ that are also in $\mathcal{C}(\sigma_1, \sigma_2)$.
$\mathcal{E}_2^{\mathrm{C}}(\sigma_1, \sigma_2)$	Activities in $\mathcal{E}(\sigma_2)$ that are also in $\mathcal{C}(\sigma_1, \sigma_2)$.
Π^{PO}	Set of all stationary non-preemptive POPs.
π^{PO}	Non-preemptive POP.
$\mathcal{S}^{\mathrm{PO}}$	Set of system states (state space) if only POPs are considered for the non-preemptive problem.

B.3.8 Approximate Dynamic Programming

$\mathbf{A}(\tilde{\mathcal{S}})$	Regression matrix for a given set $\tilde{\mathcal{S}}$ of representative states.
$\mathbf{D}$	Diagonal scaling matrix.
$f(\boldsymbol{v})$	Weighted squared Bellman error based on $\boldsymbol{v}$.
$\gamma(s)$	State relevance weight for system state s.
$\boldsymbol{\gamma}$	Vector of state relevance weights.
$g^{\min}$	Minimum average cost obtained so far.
$g^{\mathrm{o}}(\pi)$	Average cost per unit of time for an open system under policy π.
$h(s)$	Approximate value function for system state s.
$\tilde{h}^{\mathrm{Lin}}(s)$	Approximate value function for system state s based on a linear approximation architecture.
$\tilde{h}^{\mathrm{LinP}}(s)$	Approximate value function for system state s of the preemptive problem based on a linear approximation architecture.
$\tilde{h}^{\mathrm{P}}(s)$	Approximate value function for system state s of the non-preemptive problem based on $h^{*\mathrm{P}}(s)$.
$h^{*\mathrm{P}}(s)$	Value function for system state s resulting from the optimal policy for the preemptive problem.
K^{L}	Lower bound for $K(s)$.
K^{U}	Upper bound for $K(s)$.
$N^{\tilde{\mathcal{S}}}$	Number of sets containing representative states.
N^{ADPPIBE}	Number of policy improvements in ADP-PI-BE.
N^{ADPPILS}	Number of policy improvements in ADP-PI-LS.

$\phi(s)$	Basis function depending on system state s.
$\tilde{\mathcal{S}}$	Set of representative states.
$\lvert\tilde{\mathcal{S}}\rvert^{\max}$	Maximum number of representative states.
$\mathcal{V}$	Set of all activity types.
v	Weight for basis function $\phi(s)$ in a linear approximation architecture.
$\boldsymbol{v}$	Vector of weights for the basis functions.
$\boldsymbol{v}(\pi)$	Vector of weights for the basis functions that correspond to a policy π.
$\boldsymbol{v}^*$	Best vector of weights in terms of weighted mean squared error or weighted mean squared bellman error for the basis functions.
ξ	Step size for the gradient descent approach.

B.4 Order Acceptance and Capacity Planning

$\mathcal{B}$	Set of projects to be scheduled.
$\mathcal{B}(a)$	Set of projects to be scheduled by decision a.
δ^{C}	Share of non-regular capacity used.
$\boldsymbol{\delta}^{\mathrm{M}}$	Vector of binary variables δ_p^{M}.
$\boldsymbol{\delta}^{\mathrm{MPP}}$	Vector of binary variables δ_p^{MPP}.
δ_p^{M}	Binary variable indicating that the next of project is accepted if it turns out to be of type p.
δ_p^{MPP}	Binary variable indicating that the next of project is accepted after MPP if it turns out to be of type p.
$\boldsymbol{\delta}^{\mathrm{E}}$	Vector of binary variables δ_p^{E}.
δ_p^{E}	Binary variable indicating that a project of type p is scheduled.
$\boldsymbol{F}(j,s)$	Remaining flow time of project j when the system is in state s.
$k^{\mathrm{A}}(\boldsymbol{\delta}^{\mathrm{M}})$	Cost due to acceptance or rejection of orders.
k_ϕ^{M}	Fixed cost when MPP is performed before OA for a project of general type ϕ.
k_ϕ^{PM}	Fixed cost when MPP is performed after OA for a project of general type ϕ.
λ_ϕ	Arrival rate of general project type ϕ.
μ_p	Service rate for project type p.
$\boldsymbol{n}^{\mathrm{E}}(s)$	Vector of variables $n_p^{\mathrm{E}}(s)$.
$n_p^{\mathrm{E}}(s)$	Number of projects of type p in process in system state s.
$\boldsymbol{n}^{\mathrm{W}}(s)$	Vector of variables n_p^{W}.
$n_p^{\mathrm{W}}(\mathrm{s})$	Number of waiting projects of type p in system state s.
$n^{\mathrm{NIS}}(s)$	Number of projects outside the system in system state s.
ϕ	Index for general project types.
ϕ_p	General type of project type p.
Φ	Set of general project types.
$\mathcal{P}_\phi$	Set of project types being of general type ϕ.
$Q(s,a)$	Value of decision a.

$\tilde{Q}(s,a)$	Value of decision a in the uniformized CTMDP.
$\tilde{Q}^{\mathrm{M}}(\boldsymbol{\delta}^{\mathrm{E}}, \boldsymbol{\delta}^{\mathrm{M}}, \phi, s)$	Value of the OA decision given by $\boldsymbol{\delta}^{\mathrm{M}}$ for general project type ϕ is MPP is performed before OA.
$\tilde{Q}^{\mathrm{N}}(\boldsymbol{\delta}^{\mathrm{E}}, \boldsymbol{\delta}^{\mathrm{M}}, \phi, s)$	Value of the OA decision given by $\boldsymbol{\delta}^{\mathrm{M}}$ for general project type ϕ is MPP is postponed.
$Q^{\mathrm{R}}(s,a)$	Value of decision a if only regular capacity is used.
$\tilde{Q}^{\mathrm{R}}(s,a)$	Value of decision a in the uniformized CTMDP if only regular capacity is used.
$Q^{\mathrm{NR}}(s,a)$	Change of the value of decision a if non-regular capacity is used.
$\tilde{Q}^{\mathrm{NR}}(s,a)$	Change of the value of decision a in the uniformized CTMDP if non-regular capacity is used.
$s^{\mathrm{A}}(s, \boldsymbol{\delta}^{\mathrm{E}}, p)$	State entered on arrival of an accepted order of type p when projects are scheduled according to $\boldsymbol{\delta}^{\mathrm{E}}$.
$s^{\mathrm{A}}(s, \mathcal{B}, p)$	State entered on arrival of an accepted order of type p when projects are scheduled according to $\mathcal{B}$.
$s^{\mathrm{R}}(s, \delta^{\mathrm{E}}, p)$	State entered on arrival of a rejected order of type p when projects are scheduled according to $\boldsymbol{\delta}^{\mathrm{E}}$.
$s^{\mathrm{R}}(s, \mathcal{B}, p)$	State entered on arrival of a rejected order of type p when projects are scheduled according to $\mathcal{B}$.
$V^*(s)$	Future expected total reward (Value function for state s) under an optimal policy.
w^{C}	Cost per unit of time in case of full usage of non-regular capacity.
z_p	Maximum increase of the service rate of project type p in case of full usage of non-regular capacity.
$y(s,a,s')$	Fixed reward incurred on a transition from state s to state s'.

Bibliography

1. Adelman, D. (2004). A price-directed approach to stochastic inventory/routing. *Operations Research, 52*(4), 499–514.
2. Adler, P. S., Mandelbaum, A., Nguyen, V., & Schwerer, E. (1995). From project to process management: An empirically-based framework for analyzing product development time. *Management Science, 41*(3), 458–484.
3. Anavi-Isakow, S., & Golany, B. (2003). Managing multi-project environments through constant work-in-process. *International Journal of Project Management, 21*, 9–18.
4. Anderson, E. J., & Nyrenda, J. C. (1990). Two new rules to minimize tardiness in a job shop. *International Journal of Production Research, 28*(12), 2277–2292.
5. Ashtiani, B., Leus, R., & Aryanezhad, M.-B. (2011). New competitive results for the stochastic resource-constrained project-scheduling problem: Exploring the benefits of pre-processing. *Journal of Scheduling, 14*, 157–171.
6. Azaron, A., Katagiri, H., Kato, K., & Sakawa, M. (2006). Longest path analysis in networks of queues: Dynamic scheduling problems. *European Journal of Operational Research, 174*, 132–149.
7. Azaron, A., & Modarres, M. (2007). Project completion time in dynamic PERT networks with generating projects. *Scientia Iranica, 14*(1), 56–63.
8. Azaron, A., & Tavakkoli-Moghaddam, R. (2006). A multi-objective resource allocation in dynamic PERT networks. *Applied Mathematics and Computation, 181*, 163–174.
9. Baccelli, F., Liu, Z., & Towsley, D. (1993). Extremal scheduling of parallel processing with and without real-time constraints. *Journal of the Association for Computing Machinery, 40*(5), 1209–1237.
10. Ballestín, F. (2007). When it is worthwhile to work with the stochastic RCPSP? *Journal of Scheduling, 10*, 153–166.
11. Ballestín, F., & Leus, R. (2009). Resource-constrained project scheduling for timely project completion with stochastic activity durations. *Production and Operations Management, 18*(4), 459–474.
12. Bard, J., Balachandra, R., & Kaufman, P. (1988). An interactive approach to R&D project selection and termination. *IEEE Transactions on Engineering Management, 35*, 139–146.
13. Bellman, R. (1957). *Dynamic programming*. Princeton: Princeton University Press.
14. Berman, E. B. (1964). Resource allocation in a PERT network under continuous activity time-cost functions. *Management Science, 10*(4), 734–745.
15. Bertsekas, D. (2000). *Dynamic programming and optimal control* (Vol. I). Belmont: Athena Scientific.
16. Bertsekas, D. (2007). *Dynamic programming and optimal control* (Vol. II). Belmont: Athena Scientific.

P. Melchiors, *Dynamic and Stochastic Multi-Project Planning*, Lecture Notes in Economics and Mathematical Systems 673, DOI 10.1007/978-3-319-04540-5,

17. Bertsekas, D. P., & Tsitsiklis, J. N. (1996). *Neuro-dynamic programming*. Belmont: Athena Scientific.
18. Bhulai, S., & Koole, G. (2003). On the structure of value functions for threshold policies in queueing models. *Journal of Applied Probability, 40*, 613–622.
19. Blackburn, J. D. (1991). New product development: The new time wars. In *Time-based competition – The next battleground in American manufacturing*. Homewood: Business One Irwin.
20. Boctor, F. F. (1990). Some efficient multi-heuristic procedures for resource-constrained project scheduling. *European Journal of Operational Research, 49*, 3–13.
21. Brémaud, P. (1999). *Markov chains – Gibbs fields, Monte Carlo simulation, and queues*. New York: Springer.
22. Browning, T. R., & Yassine, A. A. (2010). A random generator of resource constrained multi-project network problems. *Journal of Scheduling, 13*, 143–161.
23. Brucker, P., Drexl, A., Möhring, R., Neumann, K., & Pesch, E. (1999). Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research, 112*, 3–41.
24. Buss, A. H., & Rosenblatt, M. J. (1997). Activity delay in stochastic project networks. *Operations Research, 45*(1), 126–139.
25. Buzacott, J. A., & Shantikumar, J. G. (1993). *Stochastic models and manufacturing systems*. Englewood Cliffs: Prentice Hall.
26. Choi, J., Realff, M. J., & Lee, J. H. (2004). Dynamic programming in a heuristically confined state space: A stochastic resource constrained project scheduling application. *Computers and Chemical Engineering, 28*, 1039–1058.
27. Choi, J., Realff, M. J., & Lee, J. H. (2007). A Q-learning-based method applied to stochastic resource constrained project scheduling with new project arrivals. *International Journal of Robust and Nonlinear Control, 17*, 1214–1231.
28. Cohen, I., Nguyen, V., & Shtub, A. (2004). Multi-project scheduling and control: A process-based comparative study of the critical chain methodology and some alternatives. *Project Management Journal, 35*(2), 39–50.
29. Cox, D. R., & Smith, W. L. (1961). *Queues*. London: Methuen.
30. Crabill, T. B. (1972). Optimal control of a service facility with variable exponential service times and constant arrival rate. *Management Science, 18*(9), 560–566.
31. Creemers, S., Leus, R., & Lambrecht, M. (2010). Scheduling Markovian PERT networks to maximize the net present value. *Operations Research Letters, 38*, 51–56.
32. Davis, E. W., & Patterson, J. H. (1975). A comparison of heuristic and optimum solutions in resource-constrained project scheduling. *Management Science, 21*, 944–955.
33. De Boer, R. (1998). Resource-constrained multi-project management. PhD thesis, Universiteit Twente.
34. De Farias, D. P., & Van Roy, B. (2003). Approximate linear programming for average-cost dynamic programming. *Advances in Neural Information Processing Systems, 15*, 1619–1626.
35. De Farias, D. P., & Van Roy, B. (2003). The linear programming approach to approximate dynamic programming. *Operations Research, 51*(6), 850–865.
36. De Farias, D. P., & Van Roy, B. (2004). On constraint sampling in the linear programming approach to approximate dynamic programming. *Mathematics of Operations Research, 29*(3), 462–478.
37. Defregger, F. (2009). Revenue management for manufacturing companies. PhD thesis, Katholische Universität Eichstätt-Ingolstadt.
38. Demeulemeester, E., Vanhoucke, M., & Herroelen, W. (2003). RanGen: A random network generator for activity-on-the-node networks. *Journal of Scheduling, 6*, 17–38.
39. Demeulemeester, E. L., & Herroelen, W. S. (2002). *A research handbook* (International series in operations research & management science). Boston: Kluwer Academic.
40. De Serres, I. (1991). Simultaneous optimization of flow control and scheduling in a single server queue with two job classes. *Operations Research Letters, 10*, 103–112.

41. De Serres, I. (1991). Simultaneous optimization of flow control and scheduling in queues. PhD thesis, McGill University, Montreal.
42. De Serres, Y. (1991). Simultaneous optimization of flow control and scheduling in a single server queue with two job classes: Numerical results and approximation. *Computers and Operations Research, 18*, 361–378.
43. Dumond, J., & Mabert, V. A. (1988). Evaluating project scheduling and due date assignment procedures: An experimental analysis. *Management Science, 34*(1), 101–118.
44. Easton, F. F., & Rossin, D. F. (1997). Overtime schedules for full-time service workers. *Omega, 25*, 285–299.
45. Ebben, M. J. R., Hans, E. W., & Olde Weghuis, F. M. (2005). Workload based order acceptance in job shop environments. *OR Spectrum, 27*, 107–122.
46. Elmaghraby, S. E. (2000). On criticality and sensitivity in activity networks. *European Journal of Operational Research, 127*, 220–238.
47. Feinberg, E. A., & Yang, F. (2010). Optimality of trunk reservation for an M/M/k/N queue with several customer types and holding costs. Technical report, State University of New York at Stony Brook.
48. Fernandez, A. A., Armacost, R. L., & Pet-Edwards, J. J. A. (1996). The role of the nonanticipativity constraint in commercial software for stochastic project scheduling. *Computers Industrial Engineering, 31*(1/2), 233–236.
49. Fernandez, A. A., Armacost, R. L., & Pet-Edwards, J. K. (1998). Understanding simulation solutions to resource-constrained project scheduling problems with stochastic task durations. *Engineering Management Journal, 10*(4), 5–13.
50. Fréville, A. (2004). The multidimensional 0–1 knapsack problem: An overview. *European Journal of Operational Research, 155*, 1–21.
51. Gittins, J. C., & Jones, D. M. (1972). A dynamic allocation index for the sequential design of experiments. In J. Bolyaim (Ed.), *Progress in statistics (European meeting of statisticians, Budapest)* (Vol. 9). Budapest: Colloquium Mathematical Society.
52. Goldratt, E. M. (1997). *Critical chain*. Great Barrington: The North River Press.
53. Graham, R. L. (1966). Bounds for certain multiprocessing anomalies. *The Bell System Technical Journal, 45*(9), 1563–1581.
54. Gross, D., & Harris, C. M. (1998). *Fundamentals of queueing theory*. New York: Wiley.
55. Hans, E. W. (2001). Resource loading by branch-and-price techniques. PhD thesis, University of Twente.
56. Hans, E. W., Herroelen, W., Leus, R., & Wullink, G. (2007). A hierarchical approach to multi-project scheduling under uncertainty. *OMEGA, 35*(5), 563–577.
57. Herbots, J., Herroelen, W., & Leus, R. (2007). Dynamic order acceptance and capacity planning on a single bottleneck resource. *Naval Research Logistics, 54*, 874–889.
58. Herroelen, W., & Leus, R. (2004). Robust and reactive project scheduling: A review and classification of procedures. *International Journal of Operations and Production Management, 42*(8), 1599–1620.
59. Herroelen, W., & Leus, R. (2005). Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research, 165*, 289–306.
60. Herroelen, W., De Reyck, B., & Demeulemeester, E. (1998). Resource-constrained project scheduling: A survey of recent developments. *Computers and Operations Research, 25*(4), 279–302.
61. Hopp, W. J., & Spearman, M. L. (2000). *Factory physics* (2nd ed.). Boston: Irwin McGraw-Hill.
62. Howard, R. (1960). *Dynamic programming and Markov processes*. Cambridge: MIT.
63. Ishii, N., Takano, Y., & Muraki, M. (2011). A bidding price decision process in consideration of cost estimation accuracy and deficit order probability for engineer-to-order manufacturing. Technical report, Tokyo Institute of Technology.
64. Ivanescu, C. V., Fransoo, J. C., & Bertrand, J. W. M. (2002). Makespan estimation and order acceptance in batch process industries when processing times are uncertain. *OR Spectrum, 24*, 467–495.

65. Ivanescu, V. C., Fransoo, J. C., & Betrand, J. W. M. (2006). A hybrid policy for order acceptance in batch process industries. *OR Spectrum, 28*, 199–222.
66. Kavadias, S., & Loch, C. H. (2003). Optimal project sequencing with recourse at a scarce resource. *Production and Operations Management, 12*(4), 433–442.
67. Kelley, J. E., Jr. (1963). The critical path method: Resource planning and scheduling. In *Industrial scheduling*. Englewood Cliffs: Prentice Hall.
68. Kelley, J. E., & Walker, M. R. (1959). Critical-path planning and scheduling. In *1959 Proceedings of the eastern joint computer conference*, Boston (pp. 160–173).
69. Kemppainen, K. (2005). Priority scheduling revisited – Dominant rules, open protocols and integrated order management. PhD thesis, Helsinki School of Economics.
70. Kleywegt, A. J., & Papastavrou, J. D. (1998). The dynamic and stochastic knapsack problem. *Operations Research, 1*, 17–35.
71. Kleywegt, A. J., & Papastavrou, J. D. (2001). The dynamic and stochastic knapsack problem with random sized items. *Operations Research, 49*(1), 26–41.
72. Klimov, G. P. (1974). Time-sharing service systems I. *Theory of Probability and its Applications, 19*(3), 532–551.
73. Knudsen, N. C. (1972). Individual and social optimization in a mutliserver queue with a general cost-benefit. *Econometrica, 40*(3), 515–528.
74. Kolisch, R. (1996). Efficient priority rules for the resource-constrained project scheduling problem. *Journal of Operations Management, 14*, 179–102.
75. Kolisch, R., & Sprecher, A. (1996). PSPLIB – A project scheduling problem library. *European Journal of Operational Research, 1*, 205–216.
76. Kolisch, R., Sprecher, A., & Drexl, A. (1995). Characterization and generation of a general class of resource-constrained project scheduling problems. *Management Science, 41*(10), 1693–1703.
77. Koole, G., & Pot, A. (2005). Approximate dynamic programming in multi-skill call centers. In M. E. Kuhl, N. M. Steiger, F. B. Armstrong, & J. A. Joines (Eds.), *Proceedings of the 2005 winter simulation conference*, Orlando.
78. Koole, G., & Pot, A. (2006). Workload minimization in re-entrant lines. *European Journal of Operational Research, 174*, 216–233.
79. Kulkarni, V. G., & Adlakha, V. G. (1986). Markov and Markov-regenerative PERT networks. *Operations Research, 34*, 769–781.
80. Kumar, P. R., & Seidman, T. I. (1990). Dynamic instabilities and stabilization methods in distributed real-time scheduling of manufacturing systems. *IEEE Transactions on Automatic Control, 35*(3), 289–298.
81. Kurtulus, I. S., & Davis, E. W. (1982). Multi-project scheduling: Categorization of heuristic rule performance. *Management Science, 28*(2), 161–172.
82. Kurtulus, I. S., & Narula, S. C. (1985). Multi-project scheduling: Analysis of project performance. *IIE Transactions, 17*(1), 58–66.
83. Kutanoglu, E., & Sabuncuoglu, I. (1999). An analysis of heuristics in a dynamic job shop with weighted tardiness objectives. *International Journal of Production Research, 37*(1), 165–187.
84. Law, A. (2007). *Simulation modeling and analysis* (4th ed.). Boston: McGraw-Hill.
85. Lawrence, S. R., & Morton, T. E. (1993). Resource-constrained multi-project scheduling with tardy costs: Comparing myopic, bottleneck, and resource pricing heuristics. *European Journal of Operational Research, 64*, 168–187.
86. Lee, H., & Suh, H.-W. (2008). Estimating the duration of stochastic workflow for product development process. *International Journal of Production Economics, 111*, 105–117.
87. Levy, N., & Globerson, S. (1997). Improving multiproject management by using a queueing theory approach. *Project Management Journal, 28*(4), 40–46.
88. Lippman, S. A. (1975). Applying a new device in the optimization of exponential queueing systems. *Operations Research, 23*, 687–710.
89. Loch, C. H., & Kavadias, S. (2002). Dynamic portfolio selection of NPD programs using marginal returns. *Management Science, 48*(10), 1227–1241.

90. Loch, C. H., Pich, M. T., Urbschat, M., & Terwiesch, C. (2001). Selecting R&D projects at BMW: A case study of adopting mathematical programming methods. *IEEE Transactions Engineering Management, 48*(1), 70–80.
91. Martello, S., & Toth, P. (1990). *Knapsack problems*. New York: Wiley.
92. McManus, M. I. (1977). Optimum use of overtime in post offices. *Computers and Operations Research, 4*, 271–278.
93. Meyn, S. (2008). *Control techniques for complex networks*. Cambridge: Cambridge University Press.
94. Moallemi, C. C., Kumar, S., & Van Roy, B. (2008). Approximate and data-driven dynamic programming for queueing networks. Technical report, Graduate School of Business, Columbia University.
95. Möhring, R. H., Radermacher, F. J., & Weiss, G. (1984). Stochastic scheduling problems I. *Zeitschrift für Operations Research, 28*, 193–260.
96. Morton, T. E., & Pentico, D. W. (1993). *Heuristic scheduling systems* (Wiley series in engineering & technology management). New York: Wiley.
97. Naor, P. (1969). The regulation of queue size by levying tolls. *Econometrica, 37*(1), 15–24.
98. Nguyen, V. (1993). Processing networks with parallel and sequential tasks: Heavy traffic analysis and brownian limits. *The Annals of Applied Probability, 3*(1), 28–55.
99. Nguyen, V. (1994). The trouble with diversity: Fork-join networks with heterogeneous customer populations. *The Annals of Applied Probability, 4*(1), 1–25.
100. Nino-Mora, J. (2005). Stochastic scheduling. In P. M. Pardalos (Ed.), *Encyclopedia of optimization* (Vol. V, pp. 367–372). Dordrecht: Kluwer Academic.
101. Patterson, J. H. (1984). A comparison of exact approaches for solving the multiple constrained resource project scheduling problem. *Management, 30*(7), 854–867.
102. Perry, T. C., & Hartman, J. C. (2004). Allocating manufacturing capacity by solving a dynamic, stochastic multiknapsack problem. Technical report ISE 04T-009, Lehigh University, Pennsylvania.
103. Pinedo, M. L. (2008). *Scheduling-theory, algorithms, and systems* (3rd ed.). New York: Springer.
104. Pot, A. (2006). Planning and routing algorithms for multi-skill contact centers. PhD thesis, Faculteit der Exacte Wetenschappen, Vrije Universiteit Amsterdam.
105. Powell, W. B. (2007). *Approximate dynamic programming*. Hoboken: Wiley.
106. Powell, W. B. (2011). *Approximate dynamic programming* (2nd ed.). Hoboken: Wiley.
107. Pritsker, A. A. B., Watters, L. J., & Wolfe, P. M. (1969). Multiproject scheduling with limited resources: A zero-one programming approach. *Management Science, 16*, 93–107.
108. Puterman, M. L. (1994). *Markov decision processes – Discrete stochastic dynamic programming*. New York: Wiley.
109. Ramasesh, R. (1990). Dynamic job shop scheduling: A survey of simulation research. *OMEGA International Journal of Operations and Production Management, 18*(1), 43–57.
110. Roemer, T. A., & Ahmadi, R. (2004). Concurrent crashing and overlapping in product development. *Operations Research, 52*(4), 606–622.
111. Ross, K. W., & Tsang, D. H. K. (1989). Optimal circuit access policies in an ISDN environment: A Markov decision approach. *IEEE Transactions on Communications, 37*(9), 934–939.
112. Roubos, D. (2010). The application of approximate dynamic programming techniques. PhD thesis, Vrije Universiteit Amsterdam.
113. Roubos, D., & Bhulai, S. (2007). Average-cost approximate dynamic programming for the control of birth-death processes. Technical report, VU University Amsterdam.
114. Roubos, D., & Bhulai, S. (2010). Approximate dynamic programming techniques for the control of time-varying queueing systems applied to call centers with abandonments and retrials. *Probability in the Engineering and Informational Sciences, 24*, 27–45.
115. Schweitzer, P. J., & Seidman, A. (1985). Generalized polynomial approximations in Markovian decision processes. *Journal of Mathematical Analysis and Applications, 110*, 568–582.

116. Schwindt, C. (1998). Generation of resource-constrained project scheduling problems subject to temporal constraints. Technical report WIOR-543, Universitaet Karlsruhe.
117. Sedgewick, R. (1992). *Algorithmen* (2nd ed.). München: Addison-Wesley.
118. Sennot, L. I. (1999). *Stochastic dynamic programming and the control of queueing systems* (Wiley series in probability and statistics). New York: Wiley.
119. Slotnick, S., & Morton, T. (2007). Order acceptance with weighted tardiness. *Computers and Operations Research, 34*, 3029–3042.
120. Sobel, M. J., Szmerekovsky, J. G., & Tilson, V. (2009). Scheduling projects with stochastic activity duration to maximize expected net present value. *European Journal of Operational Research, 198*, 697–705.
121. Stidham, S., & Weber, R. (1993). A survey of Markov decision models for control of networks of queues. *Queueing Systems, 13*, 291–314.
122. Stork, F. (2001). Stochastic resource-constrained project scheduling. PhD thesis, Technische Universität Berlin.
123. Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning*. Cambridge: MIT.
124. Talla Nobibon, F., Herbots, J., & Leus, R. (2009). Order acceptance and scheduling in a single-machine environment: Exact and heuristic algorithms. Technical report, Faculty of Business and Economics, KU Leuven.
125. Talla Nobibon, F., & Leus, R. (2011). Exact algorithms for a generalization of the order acceptance and scheduling problem in a single-machine environment. *Computers and Operations Research, 38*(1), 367–378.
126. Thesen, A. (1977). Measures of the restrictiveness of project networks. *Networks, 7*, 193–208.
127. Tijms, H. C. (2003). *A first course in stochastic models*. New York: Wiley.
128. Tsai, D. M., & Chiu, H. N. (1996). Two heuristics for scheduling multiple projects with resource constraints. *Construction Management and Economics, 14*, 325–340.
129. van Foreest, N. D., Wijngaard, J., & van der Vaart, J. T. (2010). Scheduling and order acceptance for the customized stochastic lot scheduling problem. *International Journal of Production Research, 48*(12), 3561–3578.
130. Veatch, M. H. (2009). Approximate dynamic programming for networks: Fluid models and constraint reduction. Technical report, Gordon College.
131. Veatch, M. H., & Walker, N. (2008). Approximate linear programming for network control: Column generation and subproblems. Technical report, Gordon College.
132. Vepsalainen, A. P. J., & Morton, T. E. (1987). Priority rules for job shops with weighted tardiness costs. *Management Science, 33*(8), 1035–1047.
133. Vyzas, E. (1997). Approximate dynamic programming for some queueing problems. Master's thesis, Massachusetts Institute of Technology.
134. Wester, F. A. W., Wijngaard, J., & Zijm, W. H. M. (1992). Order acceptance strategies in a production-to-order environment with setup time and due-dates. *International Journal of Production Research, 30*(6), 1313–1326.
135. Williams, H. P. (1999). *Model building in mathematical programming* (4th ed.). New York: Wiley.
136. Yaghoubi, S., Noori, S., Azaron, A., & Tavakkoli-Moghaddam, R. (2011). Resource allocation in dynamic PERT networks with finite capacity. *European Journal of Operational Research, 215*, 670–678.
137. Yechiali, U. (1969). On optimal balking rules and toll charges in the GI/M/1 queueing process. *Operations Research, 19*, 349–370.
138. Yechiali, U. (1972). Customers' optimal joining rules for the GI/M/s queue. *Manage, 18*, 434–443.
139. Zijm, W. H. M. (2000). Towards intelligent manufacturing planning and control systems. *OR Spektrum, 22*, 313–345.