MACHINE SCHEDULING WITH CHANGING PROCESSING TIMES AND RATE-MODIFYING ACTIVITIES

Kabir Rustogi

A thesis submitted in partial fulfilment of the requirements of the University of Greenwich for the degree of Doctor of Philosophy

March 2013

School of Computing and Mathematical Sciences,
University of Greenwich,
London, U.K.

Dedicated to my grandparents, Mr. B. Lal Rustogi and Mrs. Manmohini Rastogi

DECLARATION

I certify that this work has not been accepted in substance for any degree, and is not concurrently being submitted for any degree other than that of Doctor of Philosophy being studied at the University of Greenwich. I also declare that this work is the result of my own investigations except where otherwise identified by references and that I have not plagiarised the work of others.

Kabir Rustogi (Student)	Prof. Vitaly Strusevich (1st Supervisor)

ACKNOWLEDGEMENTS

First, I would like to extend my sincerest gratitude to my supervisor, Prof. Vitaly Strusevich, for his constant support and encouragement. Without his guidance, I could have easily fit the description of those, whose misadventures with PhD research are documented in comic strips.

I am grateful to the CMS staff, for their timely help and advice; especially Lauren Quinton for her robotic efficiency in handling the huge amount of paper work that is required to get till this stage.

I would like thank my friends, especially LL, for making my life easier/harder during these years. I would also like to thank the years, for its characteristic nature to simply, pass, without much damage.

Finally, I am thankful to my family, for their patience and undying support.

ABSTRACT

In classical scheduling models, it is normally assumed that the processing times of jobs are fixed. However, in the recent years, there has been a growing interest in models with variable processing times. Some of the common rationales provided for considering such models, is as follows: the machine conditions may deteriorate as more jobs are processed, resulting in higher than normal processing times, or conversely, the machine's operator may gain more experience as more jobs are processed, so he/she can process the jobs faster. Another direction of improving the practical relevance of models is by introducing certain rate-modifying activities, such as maintenance periods, in the schedule.

In this thesis, we mainly focus on the study of integrated models which allow changing processing times and rate-modifying activities. When this project was started, it was felt that there was a major scope of improvement in the area, both in terms of creating more general, practically relevant models and developing faster algorithms that are capable of handling a wide variety of problems. In this thesis, we address both these issues.

We introduce several enhanced, practically relevant models for scheduling problems with changing times that allow various types of rate-modifying activities, various effects or a combination of effects on the processing times. To handle these generalised models, we develope a unified framework of algorithms that use similar general principles, through which, the effects of rate-modifying activities can be systematically studied for many different scenarios.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

Scheduling problems, in general, consist in the allocation of scarce resources over time in order to perform a set of tasks. Depending on the situation, resources and the tasks to be performed can take on many different forms. Resources may be machines in an assembly plant, CPU, memory and I/O devices in a computer system, runways at an airport, mechanics in an automobile repair shop, etc. On the other hand, activities may be various operations in manufacturing processes, execution of a computer program, landings and take-offs at an airport, car repair in an automobile repair shop, and so on.

Traditionally, scheduling problems are formulated in terms of jobs that need to be allocated to some machines. Each job is associated with a processing time. Any allocation that meets all requirements concerning the jobs and machines is called a feasible schedule. The quality of a schedule is measured by a criterion function which usually depends on the completion times of jobs. The objective of a scheduling problem is to find, among all feasible schedules, a schedule that optimises the criterion function. For instance, an objective may be to minimise the completion time of the last job (makespan), while another objective may be to minimise the number of tardy jobs. A solution to an arbitrary scheduling problem consists in giving a polynomial-time algorithm that either generates an optimal schedule, or if the problem is proven to be computationally intractable, generates a schedule that is close to optimal.

The first scheduling algorithms can be traced back to the 1950s and are attributed to Johnson (1954), Jackson (1956) and Smith (1956). The main motivation behind most of the early scheduling literature is related to efficient management of various activities occurring in a workshop. Good scheduling algorithms can lower the production cost in a manufacturing process, enabling the company to stay competitive. Beginning in the late 1960s computer scientists also started to encounter scheduling problems in the

development of operating systems. Computational resources such as CPU, memory and I/O devices were found to be scarce and efficient utilisation of these resources was essential in order to lower the cost of executing computer programs. In the last 20 years scheduling has found another important application in grid/cloud computing, where users from around the world send computational requests to a grid/cloud server for processing. Such servers handle millions of requests every minute and scheduling algorithms help in optimising the allocation of a request (job) to a server (machine) in quick time.

As expected, over the years the problems faced in scheduling have become much complicated. The classical scheduling problems are deterministic in nature and deal with constant parameters. They typically deal with situations in which the number of jobs and their processing times are known in advance. The number of machines to be used and their speeds are also fixed. Although in the past, an astounding number of studies have been carried out on such models, their underlying assumptions, however, do not always hold. This limits the applicability of these models to most of the present day scheduling problems. To cater to the growing requirements of advanced scheduling models, scheduling research has branched out into several speciality areas, e.g., stochastic scheduling, online scheduling, etc. The practical relevance of deterministic scheduling models is often increased by the introduction and study of enhanced models that combine scheduling and logistic decision-making. These include models that address the issues of machine non-availability, transportation and delivery, resource management, controlling the processing characteristics of jobs and machines, and many others.

One of the current trends in deterministic scheduling research is to investigate scheduling problems with variable processing times. Many different forms of variation have been studied, including learning effects, deterioration effects, start-time dependent processing times and resource dependent processing times. As early as in the 1930s, Wright (1936) noticed that in the aircraft industry the working costs per unit decreased with an increase in production output. He formulated the so-called 80% hypothesis, stating that with every redoubling of output the processing time for each unit decreases by 20%. This learning effect has a deep impact in many mass production systems. In scheduling terms, the existence of a learning effect means that the actual processing time of a job gets shorter, provided that the job is scheduled later. A common rationale behind this phenomenon is that as more jobs are processed, the operator gains the required experience to process the following jobs more efficiently. An effect which is antonymous to the learning effect is a deterioration effect, in which it is assumed that the later a job starts, the longer it takes to process it. Such an effect can be attributed

to the ageing/deterioration of machine conditions. As more jobs are processed the machine undergoes wear and tear and its processing efficiency decreases. One of the first papers that explored such an effect in scheduling theory is due to Browne and Yechiali (1990). Another branch of scheduling that deals with variable processing times is scheduling with resource allocation. In this it is assumed that the processing time of a job is resource dependent, so that each job is allocated a certain amount of resource, and jobs with more resources benefit from faster processing. We do not study this class of problems in this thesis. For various aspects of models with resource-dependent processing times, we refer to the recent reviews by Shabtay and Steiner (2007) and Różycki and Węglarz (2012).

In the past two decades, scheduling problems with learning and deterioration effects have received considerable attention. Although these effects are antonymous to each other, and in practically all prior papers these two phenomena are discussed separately, these effects can be modelled in similar ways. A variety of different models have been created to incorporate these effects: in a position-dependent model, the processing time of a job changes with respect to its position in the schedule, in a time-dependent model the processing time of a job changes with respect to its start-time in a schedule, in a cumulative model, the processing time of a job depends on the sum of normal processing times of the jobs scheduled earlier, and so on. Active research in these areas has lead to a vast body of knowledge on scheduling with changing processing times with learning/deterioration effects. We refer to Cheng, Ding and Lin (2004), Biskup (2008), and Gordon et al. (2008) for recent state-of-the-art reviews in these areas, as well as for references to practical applications of these models. Also see a book by Gawiejnowicz (2008), which discusses these effects specifically for a time-dependent model.

There are several common drawbacks shared by most of the publications on scheduling with learning/deterioration. First, in many papers, similar algorithmic ideas are applied to problems with minor differences in formulation, while the common features of these problems are overlooked. Most of the problems reduce to solving a linear assignment problem or are solvable by simple priority rules. Yet, no effort has been made to consolidate these vast number of results under a single umbrella. Second, the practical impact of research on scheduling models with a learning/deterioration effect alone is somewhat questionable. In practical situations it is often observed that the machines/operators are subject to periodic maintenance activities or replacements. These activities modify the rate of change of processing times, so that the learning/deterioration process is disrupted. Indeed, for a large number of jobs, if the processing conditions are not altered by some sort of a rate-modifying activity, the

processing times will either reduce to zero in the case of a learning effect or will grow to unacceptably large values in the case of deterioration. Such a situation is unrealistic.

Thus, planning a schedule with rate-modifying activities such as machine maintenance is necessary, and its importance for production enterprises and service organisations is widely recognised by both practitioners and management scientists; see for example, popular books on maintenance by Palmer (1999) and Nyman and Levitt (2001), and various Internet emporiums such as www.plant-maintenance.com, www.maintenanceworld.com, www.maintenanceresources.com. The following quotation from the influential paper by Gopalakrishnan, Ahire and Miller (1997) is especially close to the spirit of this issue:

> "Industrial systems used in the production of goods are subject to deterioration and wear with usage and age. System deterioration results in increased breakdowns leading to higher production costs and lower product quality. A well-implemented, planned preventive maintenance (PM) program can reduce costly breakdowns... Deciding what PM tasks to do, and when, constitutes a critical resource allocation and scheduling problem."

As seen from the quotation above, in the planning of rate-modifying activities in a processing sequence, the decision-maker is faced with a trade-off between two processes: (i) change of the processing conditions, and (ii) allocation of a rate-modifying period in order to control the changing conditions. However, until very recently the processes (i) and (ii) have not been fully integrated in the models studied in the scheduling literature. There is a long list of papers on process (i) alone, in which scheduling problems with changing processing times have been analysed. On the other hand, several papers consider the problem of including certain fixed machine non-availability periods, during which a machine is subject to compulsory maintenance. However, these maintenance periods (MPs) are only included to satisfy certain requirements (e.g., the number of MPs to be scheduled, or the deadline for an MP to start or to finish, or periodicity of MPs), and do not affect the processing times of the jobs. Such problems mainly concentrate on packing the jobs into gaps of a fixed duration which are created between subsequent MPs. MPs of such a kind cannot be seen as rate-modifying periods as the processing times of the jobs remain unaffected by maintenance. We refer to the two recent surveys, Lee (2004); Ma, Chu and Zuo (2010), which review such a class of problems. Other models of periodic maintenance focus on minimising functions that include operational and maintenance costs; see Bar-Noy et al. (2002); Grigoriev, van de Klundert and Spieksma (2006). Under another maintenance scenario initiated by Kubzin and Strusevich (2005, 2006), the machines are subject to a compulsory

maintenance, and its duration depends on its start time. Still, in these papers, the processing times of the jobs do not depend on their place in a schedule with respect to an MP.

One of the first papers that study the effect of a rate-modifying activity on processing conditions is that by Lee and Leon (2001). However, in this and in several follow-up papers only one rate-modifying period is introduced and the issue of changing processing times is not fully incorporated, which makes the impact of these results limited.

Studies that consider integrated scheduling models with changing processing times and rate-modifying periods have started to appear only very recently. As an example of such a model, we mention the problem studied by Kuo and Yang (2008a), who solve the problem of minimising the makespan on a single machine which undergoes a positional deterioration effect. The processing times of the jobs are known to increase as a polynomial function of their positions in the schedule. Additionally, they allow the inclusion of identical maintenance activities, so that after each MP the machine is fully restored to its "as good as new" condition and the deterioration process starts afresh. Such an MP can be seen as a rate-modifying period, as after its completion the machine is able to process the jobs faster. For this problem, the authors provide a polynomial-time algorithm which delivers the optimal number of MPs to include in the schedule, the times at which these MPs should be performed and the optimal permutation of jobs. Other papers also exist, which solve similar problems with minor differences in formulation, see, e.g., Zhao and Tang (2010), Yang and Yang (2010a,b) and Lodree and Geiger (2010). The last paper combines a single MP of a rate-modifying nature with time-dependent deterioration.

Rate-modifying periods (RMPs) to be inserted into a schedule should not be limited to maintenance periods only. It is possible that introducing an RMP in fact slows down the processing, which happens, e.g., if the RMP is related to replacement of an experienced operator by a trainee. Another possible effect of an RMP is introduced by Ji and Cheng (2010), who consider problems with a learning effect and use RMPs to further enhance the learning capabilities of the machines/operators. We review all of these papers, among others, in more detail in Chapter 3.

Given the high practical relevance of such integrated models, one may expect that research in this field may become one of the major directions in deterministic machine scheduling in the near future. However, at present only a handful of results exist in this area, and the problem scenarios considered are very specific and often unrealistic. For example, the use of a specific function (polynomial, exponential, etc.) to model a positional effect, severely limits the scope of such models. Moreover, an assumption

that an MP fully restores the machine to its default state every time it is performed, is also unrealistic. Besides the inadequacies of the models being studied, the existing solution algorithms for solving problems of this type are also very restrictive and do not permit finding solutions to more general problems. Some authors have attempted to solve somewhat general problems, e.g., Yang and Yang (2010a) consider a problem in which a polynomial positional deterioration effect is combined with maintenance activities, where unlike the model considered by Kuo and Yang (2008a), the duration of the MPs is not given by a constant, but is dependent on their start-times. However, the provided solution algorithm is cumbersome and although polynomial, requires a relatively large running time. In fact, even the algorithm provided by Kuo and Yang (2008a) for solving a very specific problem, is not the fastest possible.

Although being an active research topic of scheduling theory, the bulk of publications exhibit limitations, either at the modelling level or from the point of view of using a rather restricted set of algorithmic ideas. As an anonymous associate editor of one of our papers put it:

> "...this topic attracts a huge number of publications, most of which are minor modifications of each other".

When this project was started, it was felt that there was a major scope of improvement in the area, both in terms of creating more general, practically relevant models and developing faster algorithms that are capable of handling a wide variety of problems. In this thesis, we address both these issues. We have developed a unified framework through which the effects of rate-modifying activities can be systematically studied for many different scenarios. We have critically revised the prior results and found several instances in which other authors have unnecessarily restricted the scope of their models and have employed unsuitable algorithms to solve the resulting problems, leading to inflated running times. In fact, we noticed that after making appropriate adjustments to the models and the associated solution algorithms, we can solve a wider range of problems in faster time. Some of our observations and improvements are summarised in an invited survey Rustogi and Strusevich (2012b).

In particular, by introducing models with RMPs of a distinct nature, by introducing group-dependent effects, by combining position-dependent and time-dependent effects, and by combining learning and deterioration effects, we have ruined certain myths that this area of research had created and had been taken for granted in prior studies. The main reason most of our predecessors are not able to handle such general models, is due to the fact that most prior studies on scheduling with changing processing times

have attempted to obtain a solution by applying standard algorithmic techniques which are known for classical scheduling models, e.g., solution by simple priority rules (LPT, SPT), use of a standard linear assignment problem, etc.

We hope that this project has seriously changed the area of scheduling with changing processing times, from a collection of disjoint but similar results to a methodologically sound study, with a wide range of enhanced models and clear understanding of the conditions under which a particular algorithmic tool is applicable. We have a feeling that both the introduced enhanced models and the range of applicability of certain algorithms have almost been stretched to their limits. The immediate but not the most important implication of our study is that it (quoting the same associate editor again)

> "...may limit the flow of similar publications on the topic."

1.1 Aims and Objectives

The main goals of this thesis are:

- To introduce enhanced, practically relevant models for scheduling problems with changing processing times that allow various types of rate-modifying activities, various effects or a combination of effects on the processing times.
- Identify and develop algorithmic tools for solving scheduling problems for these generalised models.
- Set up a common framework for developing the algorithms that use similar general principles for handling various types of the generalised models.

Reaching the following objectives would form a pathway to achieving the listed goals:

- Introduce and study models with generalised positional effects represented by an arbitrary function, rather than particular functions (polynomial, exponential, etc.) that were used in most of prior studies.
- Introduce and study models in which the rate-modifying activities do not necessarily restore the machine to its default state. Such RMPs divide the schedule into a finite number of non-identical groups, thereby making the processing times of the jobs dependent on the group a job is scheduled in.

- Further generalise the model by dropping the assumption that the RMPs are identical in nature. Each RMP can have a different duration and can have a different effect on the machine conditions. The latter effect can be incorporated by assuming group-dependent effects.

- Consider models in which the durations of the RMPs are dependent on their start-times. Explore how this can be incorporated in the existing model, given that the RMPs are distinct.

- For a single machine, verify whether existing techniques can be adapted to handle problems with group-dependent positional effects. If necessary, design a new set of algorithms which are capable of solving these enhanced problems, while ensuring that the running times of these algorithms compare favourably with those of earlier algorithms used for solving less general problems.

- If the RMPs inserted in a schedule create k groups, check under which conditions the objective function of the problems with group-independent effects forms a convex sequence with respect to k, so that an optimal number of RMPs can be found by an efficient binary search algorithm. This would enable us to solve various problems faster than our predecessors.

- Identify different versions of problems with positional effects and RMPs, distinguishing them based on three criteria: (i) RMPs are identical or distinct, (ii) RMPs result in a group-independent or group-dependent effect, and (iii) duration of the RMPs are constant or start-time dependent. For each version identify the best solution approach, so that an optimal solution can be achieved in the fastest time.

- Verify whether the tool-kit developed to handle problems with positional effects and RMPs can be adapted to solve problems with time-dependent effects.

- Consider models that combine deterioration and learning effects, so that the resulting effect is possibly non-monotone in behaviour.

- Consider models that combine positional and time-dependent effects. Explore the effect of rate-modifying activities on such combined models. Formulate an enhanced model with these combined effects and solve the resulting problems by the developed methods or their modifications.

- Extend the obtained results for single machine problems to problems with parallel machines.

- Introduce and study models with cumulative effects and rate-modifying activities and study its approximability by linking it to problems of Boolean programming.
- Explore other scheduling problems, possibly without changing processing times and/or RMPs, which can be solved efficiently by means of the developed tool-kit. In particular, perform a study of an impact that adding extra machines may have on the performance of a scheduling system with parallel identical machines.

Most of the results related to the key objectives outlined above have been disseminated in the following journal publications and conferences:

Journals:

1. Rustogi, K., & Strusevich, V.A. (2011). Convex and V-shaped sequences of sums of functions that depend on ceiling functions. *Journal of Integer Sequences, 14*, Article 11.1.5, http://www.cs.uwaterloo.ca/journals/JIS/VOL14/Strusevich/strusevich2.html.
2. Rustogi, K., & Strusevich, V. A. (2012). Single machine scheduling with general positional deterioration and rate-modifying maintenance. *Omega, 40*, 791–804.
3. Rustogi, K., & Strusevich, V. A. (2012). Simple matching vs linear assignment in scheduling models with positional effects: A critical review. *European Journal of Operational Research, 222*, 393–407.
4. Kellerer, H., Rustogi, K., & Strusevich, V. A. (2012). Approximation schemes for scheduling on a single machine subject to cumulative deterioration and maintenance. *Journal of Scheduling,* DOI:10.1007/s10951-012-0287-8.
5. Rustogi, K., & Strusevich, V. A. (2013). Parallel machine scheduling: Impact of adding extra machines. Accepted for publication in *Operations Research.*
6. Rustogi, K., & Strusevich, V. A. (2013). Combining time and position dependent effects on a single machine subject to rate-modifying activities. Accepted for publication in *Omega.*
7. Rustogi, K., & Strusevich, V. A. (2013). Single machine scheduling with time-dependent deterioration and rate-modifying maintenance. Submitted to *Journal of the Operational Research Society.*

Conferences:

1. Rustogi, K., & Strusevich, V. A. (2010). Single machine scheduling with deteriorating jobs and rate-modifying maintenance activities, *23rd Conference of the European Chapter on Combinatorial Optimisation (ECCO)*, Malaga, Spain.

2. Rustogi, K., & Strusevich, V. A. (2011). Using a rectangular assignment problem for single machine scheduling with deterioration and maintenance, *24th Conference of the European Chapter on Combinatorial Optimisation (ECCO)*, Amsterdam, Netherlands.

3. Rustogi, K., & Strusevich, V. A. (2011). Enhanced models of single machine scheduling with deterioration effect and maintenance activities, *10th Workshop on Models and Algorithms for Planning and Scheduling Problems (MAPSP)*, Nymburk, Czech Republic.

4. Kellerer, H., Rustogi, K., & Strusevich, V. A. (2012). Approximation Schemes for Scheduling on a Single Machine Subject to Cumulative Deterioration and Maintenance, *25th European Conference on Operational Research (EURO)*, Vilnius, Lithuania.

5. Rustogi, K., & Strusevich, V. A. (2012). Adding an Extra Machine to a Shop Parallel Identical Machines, *25th European Conference on Operational Research (EURO)*, Vilnius, Lithuania.

6. Rustogi, K., & Strusevich, V. A. (2012). Combining time and position dependent effects with multiple maintenance periods on a single machine, *International Symposium on Combinatorial Optimisation (CO2012)*, Oxford, United Kingdom.

7. Kellerer, H., Rustogi, K., & Strusevich, V. A. (2012). Scheduling on a Single Machine Subject to Cumulative Deterioration and a Single Maintenance, *International Symposium on Combinatorial Optimisation (CO2012)*, Oxford, United Kingdom.

We use these results to systematically achieve the main goals of this thesis.

1.2 Structure of the Thesis

Due to the variety of aspects addressed in this thesis, its remainder is split into four parts.

Part I provides a literature review of the classical scheduling models and results used in this thesis. In Chapter 2, we provide a general background of the theory of scheduling. We define terminology and notation that is used throughout the thesis and introduce some basic scheduling models that are related to this study. Furthermore, we briefly discuss some of the important solution procedures that are often used to solve problems arising in combinatorial optimisation. In Chapter 3, we review some prior scheduling models that deal with changing processing times. We discuss the main rationales that have been proposed for such a phenomena and systematically classify the past studies into different subject areas. Further, we explore the subject areas most relevant to this thesis in greater detail and present some important prior results.

Part II describes the general principles consistently applied in this study to various enhanced scheduling models with changing processing times. In Chapter 4, we provide a general framework which is used to systematically study the effect of rate-modifying activities throughout the thesis. We give a formal description of the main problem that we address in this piece of research. In Chapter 5, we discuss discrete convexity of particular sequences that are widely applied in the thesis to different models with changing processing times, and which is applicable to other areas as well. The results of this chapter are used in several different sections of this thesis.

Part III addresses single machine problems with changing processing times and rate-modifying activities. In each of the chapters included in this part, we study a different model for changing processing times and combine it with rate-modifying activities. In Chapters 6 and 7, we study models with positional deterioration effects, the former dealing with job-independent effects and the latter dealing with job-dependent effects. In Chapter 8, we study models with a simple linear time-dependent deterioration effect. In Chapter 9, we study a very general model, in which deterioration and learning effects are combined and positional and time-dependent effects occur simultaneously. In Chapter 10, we study models with cumulative deterioration effects.

Part IV is dedicated to parallel machine scheduling. In Chapter 11, we perform an analytical study of the effect of adding extra machines to a system. In this chapter, we do not study models with changing processing times, but many of the techniques that we develop for solving problems considered earlier in the thesis, have been applied to obtain positive results. In Chapter 12, we study enhanced models for parallel machine scheduling, with changing processing times and rate-modifying activities.

Finally, in Chapter 13, we provide a brief summary of the results obtained in the thesis and give some concluding remarks. The logical flow of the thesis is presented in Figure 1.1.

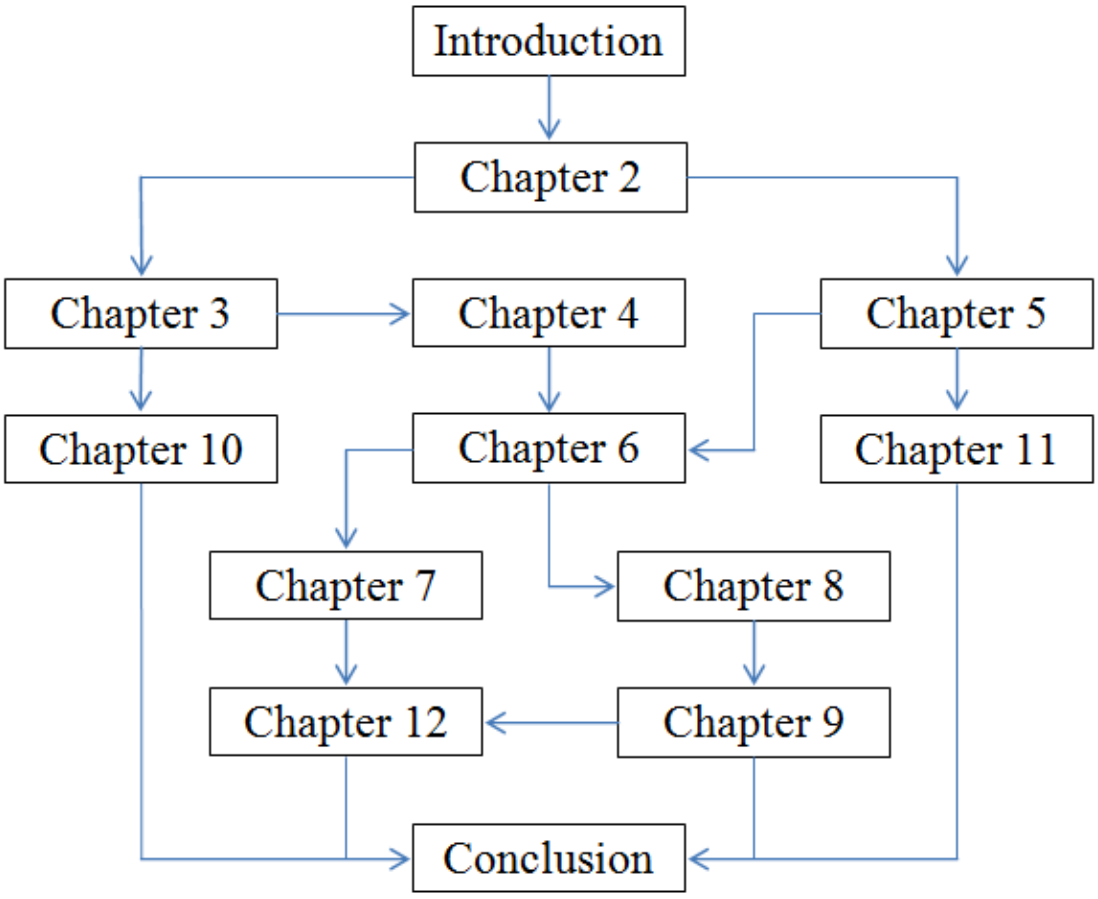


Figure 1.1: Logical flow of thesis

Part I

Literature Review

CHAPTER 2

Classical Scheduling Models and Algorithms

In this chapter, we provide a general background of the theory of scheduling. We define terminology and notation that is used throughout the thesis and introduce some basic scheduling models that are related to this study. Furthermore, we briefly discuss some of the important solution procedures that are often used to solve problems arising in scheduling. We refer to the reviews by Chen, Potts and Woeginger (1998) and Leung (2004), and the monographs by Brucker (2007) and Gawiejnowicz (2008), for most of the content presented in this chapter.

2.1 Classical Scheduling Models

The models that we study in this thesis are extensions of the following family of classical scheduling problems. In the most general setting, the jobs of set $N = \{1, 2, \ldots, n\}$ have to processed on $m \geq 1$ machines $M_1, M_2, \ldots, M_m$, and the following information is typically known for each job:

- **Processing time** (p_{ij})**:** If a job $j \in N$ is assigned to machine M_i, $1 \leq i \leq m$, then the processing time of job j is equal to p_{ij}, where we assume that all p_{ij} values are non-negative integers. The subscript i is omitted if the processing time of a job j is independent of the machine assignment.

- **Release date** (r_j)**:** The release date r_j specifies the time at which a job $j \in N$ becomes available for processing.

- **Due date** (d_j)**:** The due date d_j specifies the date at which a job $j \in N$ is expected to complete. Completion of a job after its due date is allowed but a

cost is incurred.

- **Deadline** $(\overline{d}_j)$**:** The deadline $\overline{d}_j$ specifies the time by which a job $j \in N$ must be completed. Unlike the due date a deadline is a hard constraint.

- **Weight** (w_j)**:** The weight w_j of a job $j \in N$ reflects the importance of the job.

A *schedule* specifies, for each machine M_i and each job j, one or more time intervals throughout which processing is performed on job j by machine M_i. A schedule is said to be *feasible* if it ensures that at any given time, each machine processes no more than one job and no job is assigned to more than one machine. Besides, a feasible schedule must satisfy all the conditions laid out by the specific problem type. Typically, a problem type is specified by three criteria: machine environment, job characteristics and some optimality criterion. It is convenient to denote the resulting problem by the representation scheme of Graham et al. (1979), in which a three-field descriptor $\alpha|\beta|\gamma$ is used to define the problem: α represents the machine environment, β defines the job characteristics, and γ is the optimality criterion.

2.1.1 Machine Environment

A machine environment is defined by the configuration in which the machines are available. Machine configurations can be broadly divided in two classes: *single-stage* systems and *multi-stage* systems. In a single-stage system each job requires exactly one operation, whereas in a *multi-stage* system jobs require operations at different stages.

Single-stage systems involve either a single machine, or m machines operating in parallel. In the case of parallel machines, each machine has the same function. The possible machine environments in the α field for a single-stage system are as follows:

- **Single machine** (1)**:** All jobs are processed on a single machine and the processing time of a job $j \in N$ is simply given by p_j.

- **Identical parallel machines** (Pm)**:** Jobs are simultaneously processed on $m \geq 1$ identical machines and the processing time of a job $j \in N$ is given by p_j, irrespective of the machine assignment.

- **Uniform parallel machines** (Qm)**:** Jobs are simultaneously processed on $m \geq 1$ identical machines, however, each of them operate at different speeds. The speed of a machine M_i is given by s_i, so that the processing time of a job $j \in N$ assigned to machine M_i is given by $p_{ij} = p_j/s_i$, $1 \leq i \leq m$.

- **Unrelated parallel machines** (Rm)**:**Jobs are simultaneously processed on m distinct machines. The processing time of each job depends on the machine assignment, so that the processing time of a job $j \in N$ assigned to machine M_i is given by p_{ij}.

For each case, it is assumed that all machines become available to process jobs at time zero. In this thesis we only study single-stage systems. However, for completeness we give a brief description of multi-stage systems as well.

There are three main types of multi-stage systems. All such systems comprise of s stages, each having a different function. In a *flow shop* with s stages, the processing of each job goes through the stages $1, \ldots, s$ in that order. In an *open shop*, the processing of each job also goes once through each stage, but the routing (that specifies the sequence of stages through which a job must pass) can differ between jobs and forms part of the decision process. In a *job shop*, each job has a prescribed routing through the stages, and the routing may differ from job to job. There are also multiprocessor variants of multi-stage systems, where each stage comprises several (usually identical) parallel machines.

2.1.2 Job Characteristics

Each job $j \in N$, may be characterised by a set of properties/constraints. If the jobs are associated with more than one property, then all properties can be denoted in the three-field notation by defining $\beta := \beta_1, \beta_2, \ldots, \beta_z$, where $\beta_1, \beta_2, \ldots, \beta_z$ denote the properties $1, 2, \ldots, z$ of the jobs. If no properties are defined for the jobs then define $\beta := \varnothing$. Below we list out few of the common properties that are often associated with jobs in classical scheduling theory.

- **Processing Times** (p_j)**:** The processing time of a job is usually given by a non-negative integer that is proportional to the processing requirements of the job. In normal circumstances a special mention of the processing times of the jobs is not required in the three-field notation. However, if the processing times are defined by some special rule, then such a rule must be denoted by appropriately updating the β field, e.g., if a system has all jobs with unit processing times, then we include $p_j = 1$ in the β field.

- **Preemptions** ($pmtn$)**:** Jobs with preemption allow the processing of any operation to be interrupted and resumed at a later time on the same or on a different

machine. If preemptions are allowed, we include *pmtn* in the β field, otherwise, not.

- **Release dates** (r_j)**:** The release date r_j defines the earliest time at which a job $j \in N$ can begin processing. If the symbol r_j is not present in the β field, then a job j can begin processing at any time.

- **Deadline** ($\overline{d}_j$)**:** If the symbol $\overline{d}_j$ is present in the β field, a job $j \in N$ must be completed by time $\overline{d}_j$, otherwise, no such restrictions are imposed.

- **Precedence constraints** (*prec*)**:** The precedence constraints specify the scheduling constraints of the jobs, in the sense that certain jobs must be completed before some other jobs can begin processing, i.e., if job j has precedence over job k, then k cannot start its processing until j is completed. Precedence constraints are usually specified by a directed acyclic precedence graph G with vertices $1, \ldots, n$. There is a directed path from vertex j to vertex k if and only if job j has precedence over job k. If *prec* is not specified in the β field, the jobs are not subject to any precedence constraints.

There are some other job characteristics commonly found in the scheduling literature, such as *no-wait*, *restrictions on the number of jobs*, etc., but we do not mention them here as they are associated with multi-stage systems. Moreover in this thesis, we do not consider problems with release dates, deadlines or precedence constraints. Our main focus is on the study of changing processing times of jobs. In Chapter 11, we also discuss some parallel machine problems with preemption.

2.1.3 Optimality Criteria

The optimality criterion of a scheduling problem or the objective function to be minimised is always a non-decreasing function of the completion time of the jobs. For a given schedule π, let $C_j(\pi)$ denote the *completion time* of a job $j \in N$. Some commonly used objective functions in the γ field include:

- **Makespan** ($C_{\max}$)**:** The makespan is the maximum completion time of a job in the schedule, i.e., $C_{\max}(\pi) = \max\{C_j(\pi) | j \in N\}$. For a single machine environment, the makespan is simply given by the completion time of the last job in the schedule.

- **Sum of completion times** ($\sum C_j$)**:** This objective function denotes the sum of the times each job has to wait before being fully processed by the system. It

follows that the quantity $\left(\sum_{j\in N} C_j\right)/n$ represents the average time a job has to wait before being completed. A more general form of this objective function is the *sum of weighted completion times*, or $\sum_{j\in N} w_j C_j$, where w_j is the weight of job j.

- **Total flow time** ($\sum F_j$)**:** This objective function is relevant if the problem specifies the release dates r_j of each job $j \in N$. The flow time of a job j is defined by the total time it has to wait before being completed, i.e., $F_j = C_j - r_j$, $j \in N$. The significance of the total flow time $\sum_{j\in N} F_j$ is the same as that of the sum of completion times. In fact, if it is known that all jobs are available for processing at time zero, it is common to denote the sum of completion times, simply as total flow time. Notice that in this thesis, we do not consider problems with release dates, so we often use the term flow time in the context of $\sum C_j$.

If for a job $j \in N$, the due date is given as d_j, we define the *lateness* $L_j(\pi) = C_j(\pi) - d_j$; the *earliness* $E_j(\pi) = \max\{d_j - C_j(\pi), 0\}$; the *tardiness* $T_j(\pi) = \max\{C_j(\pi) - d_j, 0\}$; and the *unit penalty* $U_j(\pi) = 1$, if $C_j(\pi) > d_j$, and $U_j(\pi) = 0$, otherwise. Some commonly found objective functions related to problems with due dates are: the *maximum lateness* $L_{\max}(\pi) = \max\{L_j(\pi)|j \in N\}$; the *total (weighted) tardiness* $\sum_{j\in N}(w_j)T_j$; the *(weighted) number of late jobs* $\sum_{j\in N}(w_j)U_j$; the *total (weighted) earliness* $\sum_{j\in N}(w_j)E_j$. Also, some situations require more than one of these criteria to be considered.

In this thesis, we mainly concentrate on problems without due dates or release dates, so our prime focus is on the objectives $C_{\max}$ and $\sum C_j$. In Sections 3.2.1 and 6.2, we study some specially structured problems with due dates and prove that they can be solved using a similar methodology as rest of the problems considered in this thesis.

2.2 Classical Scheduling Algorithms

In this section, we outline the common methods and techniques that are used to analyse and solve scheduling problems. We mainly focus on classical problems such as those defined in Section 2.1, in which the input is deterministic and is known in advance.

2.2.1 Time Complexity of Algorithms

Most scheduling problems are written in terms of an *optimisation problem,* in which we need to search for a solution that optimises a certain objective function. A typical

optimisation problem can be viewed as a function f that maps every instance or *input* $\mathbf{x}$ to an *output* $f(\mathbf{x})$. In complexity theory, however, we usually deal with a so-called *decision problem*, in which the output $f(\mathbf{x})$ can take only two values: "true" and "false". By means of binary search one can represent every optimisation problem as a sequence of decision problems.

The input $\mathbf{x}$ to any problem is usually assumed to have a *binary encoding*, and the size of the input is the number of binary digits needed to describe the instance. For example, the number of binary digits needed to represent the positive integer k is $1 + \lfloor \log_2 k \rfloor$. Another encoding scheme is the *unary encoding*, which is mostly used only for theoretical reasoning. In unary encoding an integer is represented as a string of 1's, e.g., the number 5 would be represented as 11111.

For a given input $\mathbf{x}$, a step-by-step procedure that generates the correct output $f(\mathbf{x})$ after a finite number of steps, is known as an *algorithm*. Each step, in turn, can be decomposed into a finite number of *elementary operations*, such as additions, multiplications and comparisons. The *time complexity* or *running time* of an algorithm is expressed as the total number of elementary operations, that are required by the algorithm to solve the given problem. This is normally based on a worst-case instance and expressed asymptotically as a function (say, $T(n)$) of the size of the input (say, n). Often, the function $T(n)$ can be quite cumbersome and in practice, is usually represented in the *Big-O notation* as $O(g(n))$. Formally, we write $T(n) = O(g(n))$, if there exists a constant c such that $T(n) \leq cg(n)$. For example, for a function $T(n) = 5n^4 + 100n^3 + n \log n$ we can write $T(n) = O(n^4)$.

If the function $g(n)$ is polynomial in the input size n, the algorithm is said to be a *polynomial-time* algorithm. On the other hand, if the function $g(n)$ is exponential in n, the algorithm is said to be an *exponential-time* algorithm. In complexity theory, an algorithm is considered to be "good" if it is polynomial-time; otherwise, it is considered to be "bad". If a problem is known to be solvable by a polynomial-time algorithm, then it is considered *easy* and is said to be *polynomially solvable*. The class of all polynomially solvable problems (no matter optimisation or decision) is called class P.

Next, the class of all decision problems for which a given solution ("true" or "false") can be verified as being correct or incorrect by a deterministic polynomial-time algorithm, is called class NP. Alternatively, the class NP is a class of all decision problems for which an optimal solution ("true" or "false") can be found and verified by a *non-deterministic polynomial-time* (NP) algorithm. Obviously, a solution to the decision problems belonging to class P can be verified in polynomial time by a deterministic algorithm, thus, $P \subseteq NP$. It is conjectured that $P \neq NP$, although no proof is known.

The most difficult problems in class NP are the *NP-complete* problems. Unless $P = NP$, which is considered unlikely, an NP-complete problem does not possess a polynomial-time algorithm. Informally, a problem X in NP is NP-complete if any other problem in NP can be solved in polynomial time by an algorithm that makes a polynomial number of calls to a subroutine that solves problem X. Note that this implies that, if an NP-complete problem allows a polynomial algorithm, then all problems in NP would allow polynomial algorithms. Researchers on complexity theory have identified a huge body of NP-complete problems.

A popular NP-complete problem, which is also relevant to the content of this thesis (see, e.g., Chapter 10) is the Subset-sum problem, which is defined as follows:

SUBSET-SUM PROBLEM: Given a positive integer Z, a set $R = \{1, 2. \ldots, r\}$, and a positive integer u_j, for each $j \in R$, does there exist a subset $R' \subseteq R$ such that $\sum_{j \in R'} u_j = Z$?

Note that the concepts of polynomial solvability and NP-completeness crucially depend on the encoding scheme used. If one changes the encoding scheme from binary to unary, the problem may become easier, as the input becomes longer and hence the restrictions on the running time of a polynomial algorithm are less stringent. A problem that can be solved in polynomial time under the unary encoding scheme is said to be *pseudo-polynomially* solvable. If a problem is NP-complete even under the unary encoding scheme it is known as strongly NP-complete or NP-complete in the strong sense, otherwise, it is simply referred to as NP-complete or NP-complete in the ordinary sense. The Subset-sum problem defined above is known to be NP-complete in the ordinary sense (see Karp (1972)).

An optimisation problem is known as (strongly) *NP-hard* if its decision version is (strongly) NP-complete. NP-hard problems can be informally defined as problems which are at least as hard as the hardest problems in NP. Typically, there are two ways of solving NP-hard optimisation problems. First, we can try to find an optimal solution using an exponential-time algorithm, which obviously is not a very acceptable approach. Second, we can use some sort of an approximation scheme which searches for a feasible solution, close enough to the optimum, in reasonably quick time. We review some basic concepts regarding approximation algorithms in the next section.

2.2.2 Approximation Algorithms

The NP-hardness of an optimisation problem suggests that it is not always possible to find an optimal solution quickly. However, instead of searching for an optimal

solution with enormous effort, we may instead use an *approximation algorithm* to generate approximate solutions that are close to the optimum with considerably less computational effort.

Consider a scheduling problem in which the objective is to minimise a cost function $F(S) \geq 0$, where S denotes any feasible schedule for the given problem. Let S^* denote an optimal schedule so that for any feasible schedule S, the inequality $F(S^*) \leq F(S)$ holds. Further, assume that there exists an approximation algorithm H that generates a schedule S^H. The algorithm H is called a *ρ-approximation algorithm* if $F(S^H) \leq \rho F(S^*)$, where $\rho \geq 1$. We refer to ρ as a *ratio guarantee* for the algorithm H since it provides a bound on the performance under any circumstances. If the ratio ρ is the smallest possible, then it is called the *worst-case ratio bound* of algorithm H. Often, an approximation algorithm is also known as a *heuristic*.

A family of ρ-approximation algorithms is called a *polynomial-time approximation scheme* (PTAS) if $\rho = 1 + \varepsilon$ for any $\varepsilon > 0$, and the running time of every algorithm H_ε in such a family is polynomial with respect to the length of the problem input. Furthermore, if the running time of every algorithm H_ε is bounded by a polynomial in the input size and $1/\varepsilon$, then the family is called a *fully polynomial-time approximation scheme* (FPTAS). If a scheduling problem is strongly NP-hard, then it neither allows an FPTAS nor a pseudo-polynomial algorithm unless $P = NP$.

There are other criteria for evaluating the performance of an approximation algorithm, which include *empirical* analysis and *probabilistic* analysis. Empirical analysis of an algorithm involves running the algorithm on a large number of test problem instances, preferably with known or estimated optimal solutions, and then evaluating the performance of the algorithm statistically. Probabilistic analysis of an algorithm can be regarded as the analytical counterpart of empirical analysis. The main goal of the analysis is to provide a probabilistic characterisation for the average-case performance of the heuristics, under some assumptions on the probability distribution of the problem parameters.

2.2.3 Reduction to Boolean Programming Problems

Very often it is observed that scheduling problems can be reduced to Boolean programming problems. In this section, we review some of the most popular Boolean programming problems which are relevant to this thesis.

Linear Assignment Problem

A meaningful interpretation of the *Linear Assignment Problem* (LAP) is as follows. Suppose that n jobs need to be assigned to $m \geq n$ candidates, exactly one job per candidate. It is known that c_{ji} is the cost of assigning job j to candidate i. Our objective is to find an assignment so that the total cost is minimised.

It is convenient to arrange all costs as an $n \times m$ cost matrix $C = (c_{ji})$, where the j-th, $j \in \{1, \ldots, n\}$, row contains the elements $c_{j1}, c_{j2}, \ldots, c_{jm}$, and the i-th, $i \in \{1, \ldots, m\}$, column contains the elements $c_{1i}, c_{2i}, \ldots, c_{mi}$. It is required to select n elements, exactly one from each row and exactly one from each column, so that their sum is minimised. The $n \times m$ LAP is also known as a *rectangular assignment problem* and can be formulated as a Boolean programming problem in the following way:

$$\begin{aligned} \min \quad & \sum_{j=1}^{n} \sum_{i=1}^{m} c_{ji} x_{ji} \\ \text{subject to} \quad & \sum_{i=1}^{m} x_{ji} = 1, \qquad j \in \{1, \ldots, n\}; \\ & \sum_{j=1}^{n} x_{ji} = 1, \qquad i \in \{1, \ldots, m\}; \\ & x_{ji} \in \{0, 1\}, \quad j \in \{1, \ldots, n\}, \, i \in \{1, \ldots, m\}. \end{aligned} \tag{2.1}$$

The algorithm to solve a rectangular assignment problem of the form (2.1) has been outlined by Bourgeois and Lassale (1971). The running time of this algorithm is $O(n^2 m)$, $m \geq n$. Below we reproduce the main steps of this algorithm. Later on in the thesis in Chapter 7, we shall perform an in-depth analysis of this algorithm and modify some of its steps to suit our requirements.

Let either a row or a column of the matrix C be called a *line*. The algorithm given by Bourgeois and Lassale (1971) manipulates with the cost matrix, reduces the original (positive) elements on a line-by-line basis, so that some of them become zeros. Two zeros that do not belong to the same line are called *independent*. There are two types of labels applied to a zero: it can be *starred* to become 0^* or *primed* to become $0'$. During the run of the algorithm, some lines are said to be *covered*. In all iterations of the algorithm, the starred zeros are independent, and their number is equal to the number of the covered lines, with each covered line containing exactly one 0^*. The algorithm stops having found n starred zeros in the current matrix. The primed zeros in a current partial solution are seen as potential candidates to become starred zeros.

Algorithm BourLas (see Bourgeois and Lassale (1971))

Step 0. Consider a row of the matrix C, subtract the smallest element from each element in the row. Do the same for all other rows.

Step 1. Search for a zero, Z, in the matrix. If there is no starred zero in its row or column, star Z. Repeat for each zero in the matrix. Go to Step 2.

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

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

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

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

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

A special case of the $n \times m$ LAP is the $n \times n$ LAP with a square cost matrix. One of the most studied problems of integer programming, the $n \times n$ LAP is known to be solvable in $O(n^3)$ time by the so-called Hungarian algorithm due to Kuhn (1955).

Below we give its formulation in terms of Boolean programming:

$$\begin{array}{lll} \min & \displaystyle\sum_{j=1}^{n}\sum_{i=1}^{n} c_{ji}x_{ji} & \\ \text{subject to} & \displaystyle\sum_{i=1}^{n} x_{ji} = 1, & j \in \{1,\ldots,n\}\,; \\ & \displaystyle\sum_{j=1}^{n} x_{ji} = 1, & i \in \{1,\ldots,n\} \\ & x_{ji} \in \{0,1\}\,, & j \in \{1,\ldots,n\}\,,\, i \in \{1,\ldots,n\}\,. \end{array} \tag{2.2}$$

A special case of the LAP of the form (2.2) can be solved faster if $c_{ji} = \alpha_i\beta_j$, so that the input of the problem is determined by two arrays $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ and $\beta = (\beta_1, \beta_2, \ldots, \beta_n)$. Such a problem is known as the linear assignment problem with a *product matrix*. It is also sometimes referred to as the problem of minimising a linear form $\sum_{j=1}^{n} \alpha_{\pi(j)}\beta_j$ over a set of all permutations. Provided that $\beta_1 \le \beta_2 \le \cdots \le \beta_n$, the problem reduces to finding a permutation $\varphi = (\varphi(1), \varphi(2), \ldots, \varphi(n))$ of the components of array α, such that for any permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$, the inequality

$$\sum_{j=1}^{n} \alpha_{\varphi(j)}\beta_j \le \sum_{j=1}^{n} \alpha_{\pi(j)}\beta_j \tag{2.3}$$

holds. In terms of the original Boolean decision variables in (2.2), we have that $x_{j,\varphi(j)} = 1$, $j \in \{1,\ldots,n\}$, while all other x_{ji} are zero. The classical result by Hardy, Littlewood and Polya (1934) leads to the following algorithm that finds the required permutation φ.

Algorithm Match

INPUT: Two arrays $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ and $\beta = (\beta_1, \beta_2, \ldots, \beta_n)$

OUTPUT: A permutation $\varphi = (\varphi(1), \varphi(2), \ldots, \varphi(n))$ that satisfies (2.3)

Step 1. Renumber the components of the array β so that

$$\beta_1 \le \beta_2 \le \cdots \le \beta_n.$$

Step 2. Output a permutation φ such that

$$\alpha_{\varphi(1)} \ge \alpha_{\varphi(2)} \ge \cdots \ge \alpha_{\varphi(n)}.$$

As follows from Hardy, Littlewood and Polya (1934), the following statement holds.

Lemma 2.1. *A permutation φ found by Algorithm Match satisfies (2.3) for an arbitrary permutation π.*

Finding permutation φ requires the sorting of two arrays of n numbers and can be done in $O(n \log n)$ time. Informally, this permutation matches the larger components of one of the two given arrays with smaller components of the other array. Another way of arriving at Lemma 2.1 is as follows.

Assume that the arrays $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ and $\beta = (\beta_1, \beta_2, \ldots, \beta_n)$ are ordered such that $\alpha_1 \geq \alpha_2 \geq \cdots \geq \alpha_n$ and $\beta_1 \leq \beta_2 \leq \cdots \leq \beta_n$ hold. It is easy to verify that the resulting cost matrix $C = \left(c_{ji} = \alpha_i \beta_j\right)$ satisfies the following property

$$c_{ij} + c_{rs} \leq c_{is} + c_{rj}, \text{ for all } 1 \leq i < r \leq n,\ 1 \leq j < s \leq n.$$

The above property is known as the *Monge property* and any matrix C obeying the Monge property is called a *Monge matrix.* Monge matrices are known to permit a solution to the assignment problem of the form (2.2) very easily. If $C = (c_{ji})$ is a Monge matrix of dimension $n \times n$, then an optimal solution to the assignment problem is given by

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

See Burkard, Klinz and Rudolf (1996) for a proof of the above property and more applications of Monge matrices in combinatorial optimisation. Notice, that the above property is equivalent to Lemma 2.1.

In this thesis, we use different versions of the linear assignment problem on several occasions. We refer the reader to the recent monograph by Burkard, Dell'Amico and Martello (2009) for an excellent exposition of all aspects of the assignment problem.

Subset-Sum Problem

The decision version of the Subset-Sum problem was defined in Section 2.2.1. The optimisation version of the Subset-sum problem can be defined as

$$\begin{aligned} \max \quad & \sum_{j \in R} u_j x_j \\ \text{subject to} \quad & \sum_{j \in R} u_j x_j \leq Z \\ & x_j \in \{0, 1\},\ j \in R, \end{aligned} \tag{2.4}$$

where Z and u_j, $j \in R$, are all positive integers. Since the decision version of the above problem is an NP-complete problem, the Subset-sum problem of the form (2.4) is known to be NP-hard in the ordinary sense. This problem admits an FPTAS which, for a given positive ε, either finds an optimal solution $x_j^* \in \{0,1\}$, $j \in N$, such that

$$\sum_{j \in R} u_j x_j^* < (1-\varepsilon)Z,$$

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

$$(1-\varepsilon)Z \leq \sum_{j \in R} u_j x_j^\varepsilon \leq Z.$$

The fastest of known FPTASs requires no more than $O\left(\min\left\{n/\varepsilon, n + \frac{1}{\varepsilon^2}\log\left(\frac{1}{\varepsilon}\right)\right\}\right)$ time; see Kellerer et al. (2003) and Lemma 4.6.1 in Kellerer, Pferschy and Pisinger (2004).

Half-Product Problem

Let $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ denote a vector with n 0-1 components. Introduce the function

$$H(\mathbf{x}) = \sum_{1 \leq i < j \leq n}^{n} a_i b_j x_i x_j - \sum_{j=1}^{n} h_j x_j, \tag{2.5}$$

where for each j, $1 \leq j \leq n$, the coefficients a_j and b_j are non-negative integers, while h_j is an integer that can be either negative or positive. Problems of quadratic Boolean programming similar to (2.5) were introduced in 1990s as mathematical models for various scheduling problems by Kubiak (1995) and Jurisch, Kubiak and Józefowska (1997). The function $H(\mathbf{x})$ is called a *Half-product* since its quadratic part consists of roughly half of the terms of the product $\left(\sum_{j=1}^{n} a_j x_j\right)\left(\sum_{j=1}^{n} b_j x_j\right)$. This function and the term "Half-product" were introduced by Badics and Boros (1998), who considered the problem of minimising the function H with respect to Boolean decision variables with no additional constraints. Notice that we are only interested in the instances of the problem for which the optimal value of the function is strictly negative; otherwise, setting all decision variables to zero solves the problem. The problem of minimising function $H(\mathbf{x})$ of the form (2.5) is called the *Half-product problem*, which is known to be NP-hard in the ordinary sense. The first FPTAS for the Half-product problem that requires strongly polynomial time is due to Erel and Ghosh (2008), and the published running time is $O(n^2/\varepsilon)$. We refer the reader to a recent state-of-the-art review by

Kellerer and Strusevich (2012), for other aspects of the Half-product problem and for its scheduling applications.

2.2.4 Combinatorial Counting

Many scheduling problems reduce to enumeration of several feasible solutions and we are required to choose the best among them as an optimal solution. For completeness, we provide a brief review of some basic principles of combinatorics, which are often used in this thesis. We follow the terminology and notation of Flajolet and Sedgewick (2009).

- **Combinations:** A k-combination of a set S is a subset of k distinct elements of S. If the set has n elements the number of k-combinations is equal to $\binom{n}{k}$.
- **Arrangements:** A k-arrangement of a set S is an ordered subset of k distinct elements of S. If the set has n elements the number of k-arrangements is equal to $\binom{n}{k}k!$.
- **Partitions:** A partition of a positive integer n into exactly k positive summands is a sequence $(z_1, z_2, ..., z_k)$ of integers such that $n = z_1 + z_2 + \cdots + z_k$ and $z_1 \geq z_2 \geq \cdots \geq z_k \geq 1$. The total number of partitions of n into at most k positive summands is denoted by $P_n^{(\leq k)}$ and can be approximated by $\frac{n^{k-1}}{k!(k-1)!}$. The total number of partitions of n into exactly k positive summands is $P_n^{(k)} = P_n^{(\leq k)} - P_n^{(\leq k-1)}$.
- **Compositions:** A composition of an integer n made of k summands is a sequence $(z_1, z_2, ..., z_k)$ of positive integers such that $n = z_1 + z_2 + \cdots + z_k$. The total number of compositions $C_n^{(k)}$ in exactly k summands is given by $\binom{n-1}{k-1}$, which can be approximated as $\frac{n^{k-1}}{(k-1)!}$. The number of compositions $C_n^{(\leq k)}$ into at most k positive summands is $\binom{n+k-1}{k-1}$, which can be approximated by $\frac{(n+k)^{k-1}}{(k-1)!}$.

2.2.5 Polynomial-Time Algorithms for Classical Scheduling

In this section, we review some of the most well known polynomial-time scheduling algorithms. We shall only concentrate on the problems relevant to this thesis, i.e., single-stage problems without release dates, due dates, deadlines or precedence constraints. Our main focus will be on the makespan and the total flow time objectives.

Minimising Makespan

We focus on the problems $1 \,||\, C_{\max}$, $Pm \,||\, C_{\max}$ and $Pm \,|pmtn|\, C_{\max}$.

Problem $1 \,||\, C_{\max}$ is trivial as the makespan of a schedule π is simply given as the sum of processing times of all jobs, i.e., $C_{\max}(\pi) = \sum_{j \in N} p_j$. Thus, any feasible permutation of jobs is optimal.

Problem $Pm \,||\, C_{\max}$ is difficult to solve to optimality. Let us first consider an easier problem with two machines, which is denoted by $P2 \,||\, C_{\max}$. Intuitively, the smallest makespan for this problem is achieved if both machines have an equal load, i.e., the sum of the processing times of all jobs scheduled on each machine should be equal. This means that the lower bound on the makespan for problem $P2 \,||\, C_{\max}$ is equal to $\left(\sum_{j \in N} p_j\right)/2$. As a result, the problem of minimising the makespan on two parallel machines can be seen as a Subset-Sum problem of the form (2.4) with $Z = \left(\sum_{j \in N} p_j\right)/2$, which is an NP-hard problem. Indeed, the problems $P2 \,||\, C_{\max}$ and $Pm \,||\, C_{\max}$ are known to be NP-hard and NP-hard in the strong sense, respectively; see Garey and Johnson (1978, 1979). Thus, unless $P = NP$, there is no polynomial time algorithm to solve the problem of minimising the makespan on parallel machines without preemption. However, there exist several efficient approximation algorithms that solve such problems with good worst-case bounds. Here, we only give a description of the list scheduling algorithm which was introduced by Graham (1966, 1967) for solving problem $Pm \,||\, C_{\max}$.

List Scheduling: The List Scheduling algorithm (later referred to in the thesis as Algorithm LS) delivers a heuristic schedule $S_{LS}(m)$ for the problem $Pm \,||\, C_{\max}$, reasonably close to the optimum. In the most general setting of Algorithm LS, the jobs are arranged in an arbitrary sequence (a list) and every time that a machine becomes available, the next job from the list is assigned to it. If the optimal schedule to problem $Pm \,||\, C_{\max}$ is known to be $S^*(m)$, the worst-case analysis result by Graham (1966, 1967) states that

$$\frac{C_{\max}(S_{LS}(m))}{C_{\max}(S^*(m))} \leq 2 - \frac{1}{m}, \tag{2.6}$$

and this bound is tight. The running time of Algorithm LS for solving problem $Pm \,||\, C_{\max}$ is $O(nm)$.

Largest Processing Time first (LPT) rule: According to the LPT rule, the jobs are arranged in a *non-increasing order* of their processing times and are renumbered in a way such that

$$p_1 \geq p_2 \geq \cdots \geq p_n, \tag{2.7}$$

holds. The LPT rule is a very common strategy for solving several scheduling problems.

For solving problem $Pm \,||\, C_{\max}$, Graham (1966, 1967) further proposed that instead of taking the jobs from an arbitrary list, if the jobs are taken from an ordered list in which they are arranged in an LPT rule, then the worst-case bound of Algorithm LS can be improved and is given by

$$\frac{C_{\max}\left(S_{LPT}(m)\right)}{C_{\max}\left(S^{*}(m)\right)} \leq \frac{4}{3} - \frac{1}{3m}, \tag{2.8}$$

where $S_{LPT}(m)$ is heuristic schedule following the LPT rule. To sort a list of size n, the running time needed is $O\left(n \log n\right)$. Thus, the LPT list scheduling method can deliver an approximate solution to problem $Pm \,||\, C_{\max}$ in $O\left(n \log n + nm\right)$ time, with a worst-case ratio bound $\rho = 4/3 - 1/3m$.

We now consider preemptive scheduling. Unlike $Pm \,||\, C_{\max}$, problem $Pm \,|pmtn|\, C_{\max}$ can be solved optimally in linear time by the famous wrap-around rule by McNaughton (1959).

McNaughtons's Wrap-Around Rule: In any feasible schedule $S\left(m\right)$ on m identical machines, a job should not be assigned to more than one machine at a time. It follows that the makespan of a schedule $S\left(m\right)$ cannot be less than the largest processing time, i.e.,

$$C_{\max}(S\left(m\right)) \geq p_{\max} = \max\left\{p_j | j \in N\right\}. \tag{2.9}$$

Further, the makespan cannot be less than the average machine load, i.e.,

$$C_{\max}(S\left(m\right)) \geq \frac{\sum_{j \in N} p_j}{m}. \tag{2.10}$$

Based on these observations the value of the optimal makespan is given as

$$C_{\max}(S^{*}\left(m\right)) = D = \max\left\{p_{\max}, \frac{\sum_{j \in N} p_j}{m}\right\}.$$

To find an optimal schedule $S^{*}\left(m\right)$, assign the jobs in any order from time 0 to time D on machine M_1. If a job's processing extends beyond time D, preempt the jobs at time D, and continue its processing on machine M_2, starting at time 0. Repeat this process until all jobs are assigned. The running time of this algorithm is $O\left(n\right)$ as it requires $O\left(n\right)$ time to sum up all elements of a vector of length n and requires $O\left(n\right)$ comparisons to find the largest processing time.

If we denote the value of the optimal makespan for problems $Pm \,|pmtn|\, C_{\max}$ and

$Pm \,||\, C_{\max}$ as $C_{\max}(S^*_{np}(m))$ and $C_{\max}(S^*_{np}(m))$ respectively, then Braun and Schmidt (2003) and Lee and Strusevich (2005) have proved that the following inequality holds

$$\frac{C_{\max}(S^*_{np}(m))}{C_{\max}(S^*_{p}(m))} \leq 2 - \frac{2}{m+1}. \tag{2.11}$$

The ratio $C_{\max}(S^*_{np}(m))/C_{\max}(S^*_{p}(m))$ is known as the *power of preemption.* In particular, Braun and Schmidt (2003) show that the bound (2.11) holds if an optimal non-preemptive schedule $S^*_{np}(m)$ is replaced by a schedule $C_{\max}(S_{LPT}(m))$ found by an LPT list scheduling algorithm. Lee and Strusevich (2005) describe a large class of non-preemptive schedules for which the bound (2.11) holds.

Minimising Total Flow Time

We focus on the problems $1 \,||\, \sum C_j$, $1 \,||\, \sum w_j C_j$, $Pm \,||\, \sum C_j$, $Qm \,||\, \sum C_j$ and $Rm \,||\, \sum C_j$. Recall that since we do not consider problems with release dates, the objective of minimising the sum of completion times is equivalent to the objective of minimising the total flow time.

Let us begin with problem $1 \,||\, \sum C_j$. Given a permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ of jobs, the completion time of a job $j = \pi(r)$ scheduled in position r, $1 \leq r \leq n$, can be given as $C_{\pi(r)} = \sum_{k=1}^{r} p_{\pi(k)}$. Thus, the sum of completion times of all jobs can be written as

$$\sum_{r=1}^{n} C_{\pi(r)} = \sum_{r=1}^{n} \sum_{k=1}^{r} p_{\pi(k)} = \sum_{r=1}^{n} (n - r + 1)\, p_{\pi(r)}. \tag{2.12}$$

Notice that the sequence $n - r + 1$, $1 \leq r \leq n$, decreases with r. Thus, according to Lemma 2.1, the sum of completion times of all jobs can be minimised if the jobs are scheduled in a *non-decreasing order* of their processing times.

Shortest Processing Time first (SPT) rule: According to the SPT rule, the jobs are arranged in a *non-decreasing order* of their processing times and are renumbered in a way such that

$$p_1 \leq p_2 \leq \cdots \leq p_n. \tag{2.13}$$

holds. Similar to the LPT rule, the SPT rule is also a very common strategy for solving several scheduling problems. It follows that an optimal solution to problem $1 \,||\, \sum C_j$ is obtained if the jobs are scheduled by the SPT rule.

An optimal solution to problem $1 \,||\, \sum w_j C_j$ is obtained by applying Smith's ratio

rule.

Smith's Ratio rule: Schedule the jobs in a non-decreasing order of their ratios p_j/w_j. The optimality of this *ratio rule* is given by means of an interchange argument, which is a very useful technique in proving the optimality of several scheduling problems.

Let us now consider problem $Pm\,||\sum C_j$. Suppose that in a feasible schedule $S(m)$ with m identical machines, there are $h^{[i]}$ jobs assigned to machine M_i, so that $\sum_{i=1}^{m} h^{[i]} = n$, and these jobs are processed in accordance with a permutation $\pi^{[i]} = \left(\pi^{[i]}(1), \pi^{[i]}(2) \ldots, \pi^{[i]}(h^{[i]})\right)$, where $1 \leq i \leq m$. Then it follows from (2.12) that the sum of completion times of all jobs can be written as

$$\sum_{j=1}^{n} C_j\left(S(m)\right) = \sum_{i=1}^{m} \sum_{r=1}^{h^{[i]}} \sum_{k=1}^{r} p_{\pi^{[i]}(k)} = \sum_{i=1}^{m} \sum_{r=1}^{h^{[i]}} \left(h^{[i]} - r + 1\right) p_{\pi^{[i]}(r)}. \tag{2.14}$$

Conway, Maxwell and Miller (1967) present an algorithm for solving problem $Pm\,||\sum C_j$ which is essentially a version of Algorithm LS outlined earlier in this section. According to them, an optimal solution is obtained if the jobs are scanned in the SPT order, i.e., in non-decreasing order of their processing times, and are assigned to the first available machine. The running time required to obtain an optimal schedule $S^*(m)$ is $O\left(n \log n\right)$, which is equal to the time needed to sort the jobs in an SPT order. It is easy to verify that if $n = km + r$, where $k \geq 0$ and $0 \leq r \leq m-1$, then in the resulting optimal schedule $S^*(m)$, there will be $k+1$ jobs assigned to r machines and k jobs assigned to the remaining machines.

Further, Brucker (2007) proves in Chapter 2 of his book that the optimal value of the objective function for problem $Pm\,||\sum C_j$ can be given by a closed form formula

$$\sum_{j=1}^{n} C_j(S^*(m)) = \sum_{j=1}^{n} p_j \left\lceil \frac{j}{m} \right\rceil, \tag{2.15}$$

provided that the jobs are numbered in an LPT (2.7) order. To see that (2.15) holds, it is convenient to view the optimal schedule $S^*(m)$ in the following way. Assume that the jobs are scanned in an LPT order (2.7). Then, in schedule $S^*(m)$ each of the first m jobs takes the last position on one of the machines, each of the next m jobs takes the second from last position on one of the machines, etc. This implies in light of formula (2.14), that job j contributes its processing time into the objective function exactly $\left\lceil \frac{j}{m} \right\rceil$ times. As a result, the objective function can be written as (2.15).

Notice that McNaughton (1959) has proved, that if preemption is allowed, the value of the objective function of the resulting problem $Pm\,|pmtn|\sum C_j$ is no smaller than

that of problem $Pm \,||\, \sum C_j$.

Let us now consider problem $Qm \,||\, \sum C_j$, with machine speeds given as $s_1, s_2, \ldots, s_m$. This problem is more complicated than problem $Pm \,||\, \sum C_j$, as an optimal schedule need not use all machines, leaving the slower machines empty. For a feasible schedule $S(m)$ with m uniform machines, it follows from (2.14), that the sum of completion times of all jobs can be written as

$$\sum_{j=1}^{n} C_j\left(S(m)\right) = \sum_{i=1}^{m} \sum_{r=1}^{h^{[i]}} \sum_{k=1}^{r} p_{\pi^{[i]}(k)} = \sum_{i=1}^{m} \sum_{r=1}^{h^{[i]}} \frac{\left(h^{[i]} - r + 1\right) p_{\pi^{[i]}(r)}}{s_i}. \tag{2.16}$$

Problem $Qm \,||\, \sum C_j$ can be solved by an algorithm given by Conway, Maxwell and Miller (1967). This algorithm works similarly to the LPT version of the algorithm described above for solving problem $Pm \,||\, \sum C_j$. We fill the processing sequence on each machine from the last position forwards, starting from the jobs with the largest processing times. The next job is assigned in front of the current processing sequence. This approach does not require any advance knowledge of the number of jobs to be assigned to a machine. Below we outline the main steps of this algorithm as it appears in Chapter 5 of the book by Brucker (2007).

Algorithm QmSum

INPUT: An instance of problem $Qm \,||\, \sum C_j$

OUTPUT: An optimal schedule defined by the processing sequences $\pi^{[i]}, 1 \leq i \leq m$

Step 1. If required, renumber the jobs in the LPT order. For each machine M_i, $1 \leq i \leq m$, define an empty processing sequence $\pi^{[i]} := (\varnothing)$ and the weight $W^{[i]} = 1/s_i$.

Step 2. For each job j from 1 to n do

(a) Find the largest index v with $W^{[v]} = \min\left\{W^{[i]} | 1 \leq i \leq m\right\}$.

(b) Assign job j to machine M_v and place it in front of the current permutation $\pi^{[v]}$, i.e., define $\pi^{[v]} := (j, \pi^{[v]})$. Define $W^{[v]} := W^{[v]} + \frac{1}{s_v}$.

The running time of the Algorithm QmSum is $O(n \log n)$. The main idea behind this algorithm is derived from Lemma 2.1. The algorithm scans the jobs in the LPT order and matches the current job with the smallest available weight $W^{[v]}$ (as found in Step 2a). The current values of weights $W^{[i]}$ depend on both, the machine and the current available position on that machine, and so for each job there are at most m weights to choose from.

Let us now consider problem $Rm \,||\, \sum C_j$ with m unrelated parallel machines. This problem is solvable in $O(n^3 m)$ time by reducing it to a rectangular assignment problem of size $n \times nm$; see Horn (1973) and Bruno, Coffman and Sethi (1974). The j-th row of the cost matrix consists of the costs associated with all possible assignments of job $j \in N$. It is possible for a job j to be assigned to any of the n available positions on any of the m available machines, thus, leading to total of nm possibilities. Let $y_{j,(i,r)}$ be a binary decision variable that is equal to 1 if and only if job j is assigned to the r-th last position on machine M_i. The corresponding rectangular LAP can be written as

$$\begin{aligned}
\min \quad & \sum_{j=1}^{n}\sum_{i=1}^{m}\sum_{r=1}^{n} r p_{ij} y_{j,(i,r)} & & \\
\text{subject to} \quad & \sum_{i=1}^{m}\sum_{r=1}^{n} y_{j,(i,r)} = 1, & & j \in \{1,\ldots,n\}\,; \\
& \sum_{j=1}^{n} y_{j,(i,r)} \le 1, & & i \in \{1,\ldots,m\}\,,\ r \in \{1,\ldots,n\}\,; \\
& y_{j,(i,r)} \in \{0,1\}\,, & & j \in \{1,\ldots,n\}\,,\ i \in \{1,\ldots,m\}\,, \\
& & & r \in \{1,\ldots,n\}\,.
\end{aligned} \tag{2.17}$$

The above rectangular assignment problem is of the form (2.1) and can be solved by Algorithm BourLas outlined in Section 2.2.3.

CHAPTER 3

Scheduling with Changing Processing Times

In the last chapter, we discussed several classical scheduling problems and reviewed the algorithms that are used to solve them. In this chapter, we review the same class of problems, but for models that allow changing processing times. In the recent past, several papers have appeared that study such effects. We discuss the main rationales that have been proposed for such a phenomenon and systematically classify the prior studies into different subject areas. Further, we explore the subject areas most relevant to this thesis in greater detail and present some important results.

3.1 Brief Overview of Models

In the classical scheduling theory, it is assumed that the processing time of a job j is fixed and has a constant value p_j (or p_{ij}). In many real-life situations, however, the processing conditions may vary and the actual time taken by the machine to complete a given job, may be different from p_j.

In the context of models with changing processing times, the integer p_j is called the *normal processing time* of a job j. A meaningful interpretation of p_j is that it defines the processing duration of job j, provided that the machine is in its default condition. The actual processing time of job j, however, is not necessarily equal to p_j and may vary according to some rule. The phenomenon of changing processing times is traditionally attributed to one of the following causes: (i) deterioration, (ii) learning, (iii) rate modifying activities and (iv) resource allocation.

Informally, in scheduling with *deterioration,* we assume that the later a job starts, the longer it takes to process. The most common rationale for deterioration, probably

first stated by Gawiejnowicz (1996), is as follows: a machine is served by a human operator who gets tired or the machine loses the processing quality of its tools as more jobs are processed. On the other hand, in scheduling with *learning,* the actual processing time of a job gets shorter, provided that the job is scheduled later. A common rationale behind this is that as an operator processes more jobs or spends more time in the processing conditions, he becomes more aware of how the jobs must be processed, and as a result, the time required to complete later jobs shortens. Scheduling problems with these two effects have received considerable attention in the recent past; we refer to Cheng, Ding and Lin (2004), Biskup (2008), Gawiejnowicz (2008) and Gordon et al. (2008) for recent state-of-the-art reviews in these areas, as well as for references to more practical applications of these models. We give detailed accounts of these effects and how they are incorporated in scheduling models, in the later sections of this chapter.

The effects of learning and deterioration are essentially antonymous to each other, and in practically all prior papers these two phenomena are discussed separately. For both learning and deterioration, the corresponding effects most commonly found in the literature belong to one of the following three types:

- *Positional:* the actual processing time of job j depends on p_j and on the position of the job in the sequence;
- *Time-Dependent:* the actual processing time of job j depends on the start time of the job;
- *Cumulative:* the actual processing time of job j depends on p_j and on the sum of normal processing times of all jobs sequenced earlier.

Recently, there have been publications that consider enhanced models, that combine two of the above listed three effects. This gives rise to an additional wide range of problems, e.g., to models with time-dependent deterioration and positional learning (see Wang (2006)), or with positional deterioration and time-dependent learning (see Yang (2010)), or with cumulative deterioration and positional learning (see Wu and Lee (2008)), among other, often somewhat exotic models. A recent review by Janiak, Krysiak and Trela (2011) focuses on the variety of the models in the area, including those with combined effects.

Another way in which the processing times of jobs may change is due to the presence of certain *rate-modifying activities* in the schedule. A rate modifying activity can be defined as an activity scheduled in between sequences of jobs, which is responsible for

changing the processing conditions of the system. One of the first papers that study an effect of a rate-modifying activity on processing conditions is that by Lee and Leon (2001). They look at the problem of scheduling a single rate-modifying period (RMP) and assume that the processing time of a job j that is sequenced before the RMP is p_j, while if it is sequenced after the MP the processing time becomes $\lambda_j p_j$, where $0 < \lambda_j < 1$. Following this model, several other authors have developed polynomial-time algorithms for various scheduling problems, including due date assignment and due window assignment; see, e.g., Mosheiov and Oron (2006); Gordon and Tarasevich (2009); Mosheiov and Sarig (2009). A generalised model in which the duration of an RMP depends on its start time is studied by Mosheiov and Sidney (2010).

If it is known that the machine is undergoing a deterioration effect, the RMPs are essentially aimed at improving the processing conditions of the system (such as maintaining or repairing the machine or its parts, giving a rest to a human operator, etc.), so that the actual processing times of the jobs scheduled after such an RMP typically get smaller. Such RMPs can be referred to as maintenance periods (MPs), see, e.g., Kuo and Yang (2008a), Zhao and Tang (2010) and Yang and Yang (2010a), for models with positional deterioration and machine maintenance, and see, Lodree and Geiger (2010) for a model with time dependent deterioration and machine maintenance. On the other hand, if the machine is undergoing a learning effect, the RMPs can be associated with replacing a machine/operator, so that all learning advantages of the previous employee are lost, and the overall productivity of the system decreases. Alternatively, an RMP can be associated with an activity which further enhances the learning rate of the machine; see, e.g., Ji and Cheng (2010), who study this model for a job-dependent case.

Lastly, scheduling with *resource allocation* allows the processing time of a job to be resource dependent, so that each job is allocated a certain amount of resource, and jobs with more allotted resources benefit from faster processing. We do not review this class of problems in this thesis, as our research mainly focusses on the first three effects. However, for various aspects of models with resource dependent processing times, we refer to the recent reviews by Shabtay and Steiner (2007), Błażewicz et al. (2010), Hartmann and Briskorn (2010), Leyvand, Shabtay and Steiner (2010), Węglarz et al. (2011) and Różycki and Węglarz (2012).

The rest of this chapter is divided into two sections, in which we describe previously studied models with changing processing times. In Section 3.2, we formally introduce positional, time-dependent and cumulative effects, as they have been known in the past. We review some of the popular polynomially solvable results that exist for these

models. In Section 3.3, we discuss the effects of rate-modifying activities and review some of the prior studies that consider integrated models with rate-modifying activities and changing processing times. In all models, it is assumed that the jobs of set $N = \{1, 2, \ldots, n\}$ have to be processed either on a single machine or on parallel machines. The jobs are available for processing at time zero and are independent, i.e., there are no precedence constraints and any processing sequence is feasible. At time zero, each machine is assumed to be in perfect processing state, and its processing conditions change according to one of the three effects described above.

3.2 Models without Rate-Modifying Activities

In this section, we formally describe the three most popular models which are known for changing processing times. Problems with changing processing times can be denoted in the three-field notation, by using the middle field β to describe the particular model being applied.

3.2.1 Positional Effects

In the simplest case, if a job $j \in N$, is scheduled in position r of a schedule, then its actual processing time is given by

$$p_j(r) = p_j g(r), \ 1 \leq r \leq n, \tag{3.1}$$

where p_j is the normal processing time of a job j and the values of $g(1), g(2), \ldots, g(n)$ are *positional factors.* The function g is given in the form of an ordered array of numbers such that in the case of deterioration, we have

$$1 = g(1) \leq g(2) \leq \cdots \leq g(n), \tag{3.2}$$

and in the case of learning, we have

$$1 = g(1) \geq g(2) \geq \cdots \geq g(n). \tag{3.3}$$

Each value $g(1), g(2), \ldots, g(n)$ is assumed to be computable in constant time. Notice that in the prior studies, apart from some recent papers, see, e.g., Gordon and Strusevich (2009), the positional factors have not been considered as given by a general function g; instead only specific functions have been analysed. Often in scheduling

literature, the factors $g(r)$ are defined as the value of a known function, e.g., polynomial in r (see, e.g., Biskup (1999), Mosheiov (2001a, 2005)) or exponential in r (see, e.g., Gordon et al. (2008)), i.e., $g(r) = r^a$ and $g(r) = \gamma^r$, respectively.

In another form of positional effect, if a job $j \in N$, is scheduled in position r of a schedule, then its actual processing time is given by

$$p_j(r) = p_j g_j(r),\ 1 \leq r \leq n, \tag{3.4}$$

where the values $g_j(1), g_j(2), \ldots, g_j(n),\ j \in N$, are *job-dependent positional factors.* Such a model represents a scenario in which each job changes the machine conditions in a different way, hence each job $j \in N$ is associated with a unique set of positional factors, $g_j(r)$, $1 \leq r \leq n$. Similar to *job-independent* positional factors of the form $g(r)$, for each job $j \in N$, the job-dependent factors are also given in the form of a collection of ordered arrays of numbers, such that in the case of deterioration, we have

$$1 = g_j(1) \leq g_j(2) \leq \cdots \leq g_j(n), \tag{3.5}$$

and in the case of learning, we have

$$1 = g_j(1) \geq g_j(2) \geq \cdots \geq g_j(n). \tag{3.6}$$

Again, in scheduling literature, the factors $g_j(r)$ are often defined as the value of a known function, e.g., Mosheiov and Sidney (2003) consider a polynomial function given by $g_j(r) = r^{a_j}$. On the other hand, many authors also consider a general job-dependent positional effect, in which the actual processing time of a job scheduled in position r of a schedule is simply given by $p_j(r)$, without any mention of the associated positional factors, see, e.g., Mosheiov (2008).

Some authors study scheduling problems in which the processing time of job a $j \in N$ scheduled in position r is given as a *linear function* of r. Bachman and Janiak (2004) consider one such model in which the processing time of job is given by

$$p_j(r) = A_j + b_j r,\ 1 \leq r \leq n, \tag{3.7}$$

where A_j is the normal processing time of a job $j \in N$ and b_j is a job-dependent rate, which is strictly positive for the deterioration environment and strictly negative for the learning environment.

Problems with positional effects are typically represented in the three-field notation

by providing the formula of the actual processing time $p_j(r)$, in the middle field β of the three-field descriptor. For example, the problem of minimising the makespan on a single machine that is subject to positional deterioration of the form (3.7), can be denoted by $1|A_j + b_j r, b_j > 0|C_{\max}$.

We now discuss the algorithmic aspects of solving different problems that have been studied previously.

Minimising Makespan

Recall that the problem of minimising the makespan on parallel machines, is an NP-hard problem even without the presence of positional effects. Thus, we only discuss single machine problems in this section. Let us first discuss problems in which a job-independent positional effect is considered, so that the actual processing time of a job is defined by (3.1).

Assume that the jobs are processed on a single machine in accordance with some permutation $\pi = (\pi(1), \ldots, \pi(n))$, and a job $j = \pi(r)$ sequenced in position r is associated with a job-independent positional factor $g(r)$. Then the makespan is written as

$$C_{\max}(\pi) = \sum_{r=1}^{n} g(r) p_{\pi(r)}. \tag{3.8}$$

Notice that the problem of minimising the makespan can be seen as a linear assignment problem with a product matrix (2.3), with arrays $(g(1), g(2), \ldots, g(n))$ and $\left(p_{\pi(1)}, p_{\pi(2)}, \ldots, p_{\pi(n)}\right)$.

Mosheiov (2005) solves problem $1|p_j r^a,\ a > 0|C_{\max}$, in which a *polynomial deterioration effect* is considered, so that the positional factors $g(r)$ are defined as

$$g(r) = r^a,\ a > 0,\ 1 \leq r \leq n. \tag{3.9}$$

For this problem an optimal solution is obtained if the jobs are scheduled by the LPT rule (2.7), i.e., the job with the largest normal processing time is scheduled in the earliest available position.

Gordon et al. (2008) solves problem $1|p_j \gamma^{r-1},\ \gamma > 1|C_{\max}$, in which an *exponential deterioration effect* is considered, so that the positional factors $g(r)$ are defined as

$$g(r) = \gamma^{r-1},\ \gamma > 1,\ 1 \leq r \leq n. \tag{3.10}$$

Again, an optimal solution is obtained if the jobs are scheduled by the LPT rule (2.7).

Problems with positional learning have also received attention. Mosheiov (2001a) solves problem $1\left|p_j r^a,\ a<0\right|C_{\max}$, in which a *polynomial learning effect* is considered, so that the positional factors $g(r)$ are defined as

$$g(r)=r^a,\ a<0,\ 1\leq r\leq n. \tag{3.11}$$

For this problem an optimal solution is obtained if the jobs are scheduled by the SPT rule (2.13), i.e., the job with the smallest normal processing time is scheduled in the earliest available position.

Gordon et al. (2008) solves problem $1\left|p_j\gamma^{r-1},\ 0<\gamma<1\right|C_{\max}$, in which an *exponential learning effect* is considered, so that the positional factors $g(r)$ are defined as

$$g(r)=\gamma^{r-1},\ 0<\gamma<1,\ 1\leq r\leq n. \tag{3.12}$$

Again, an optimal solution is obtained if the jobs are scheduled by the SPT rule (2.13).

Bachman and Janiak (2004) solve problem $1\left|A_j+b_j r, b_j<0\right|C_{\max}$, in which the positional effect is given as a linear function of the form (3.7). They consider a learning effect, i.e., $b_j<0$, $j\in N$, and prove that an optimal schedule is found in $O(n\log n)$ time by sequencing the jobs in a non-increasing order of the values b_j.

Notice that, each of the problems considered above are solved by using a simple priority rule. The running time required to solve these problems is $O(n\log n)$, as this is the time needed to sort the jobs in the desired order. The optimality of these solutions follows from Lemma 2.1, as all of the problems reduce to an assignment problem with a product matrix (2.3). However, the authors often fail to notice this connection and prove the optimality of their results using a pairwise job interchange argument. Although a powerful technique in its own right, the pairwise job interchange argument does not permit a solution if the positional factors $g(1), g(2), \ldots, g(n)$ are non-monotonically sorted.

Next, let us discuss problems in which a job-dependent positional effect is considered, so that the actual processing time of a job is defined by (3.4).

Mosheiov and Sidney (2003) solve problem $1\left|p_j r^{a_j},\ a_j<0\right|C_{\max}$, in which a poly-

nomial learning effect is considered and the positional factors are given by

$$g_j(r) = r^{a_j},\ a_j < 0,\ j \in N,\ 1 \leq r \leq n. \tag{3.13}$$

They reduce the problem to an LAP of the form (2.2) with the cost function $c_{ji} = p_j i^{a_j}$, and obtain an optimal solution by the Hungarian method in $O(n^3)$ time. The same approach can be extended to solve a problem with a general job-dependent positional effect of the form $p_j(r)$, in which case the cost function becomes $c_{ji} = p_j(i)$; see Bachman and Janiak (2004).

Notice that in the case of a job-dependent positional effect, it is not possible to obtain an optimal schedule by means of a simple priority rule. The use of a full form LAP is essential, as the resulting cost function $c_{ji} = p_j g_j(r)$, cannot be seen as a product of the elements of two independent arrays. However, under certain special conditions, see, e.g., Koulamas (2010), the problem can still be solved in $O(n \log n)$ time by simple priority rules.

Minimising Total Flow Time

Unlike the makespan objective, the problem of minimising the total flow time, can be solved optimally for both single and parallel machine environments.

Single Machine Scheduling: Assume that the jobs are processed on a single machine in accordance with some permutation $\pi = (\pi(1), \ldots, \pi(n))$, and a job $j = \pi(r)$ sequenced in position r is associated with a job-independent positional factor $g(r)$. Then the total flow time is written as

$$\sum C_j(\pi) = \sum_{r=1}^{n} g(r)(n - r + 1)p_{\pi(r)}. \tag{3.14}$$

Notice that the problem of minimising the total flow time can be seen as a linear assignment problem with a product matrix (2.3), with arrays $(ng(1), (n-1)g(2), \ldots, g(n))$ and $\left(p_{\pi(1)}, p_{\pi(2)}, \ldots, p_{\pi(n)}\right)$. However, similar to the problem of minimising the makespan, many authors fail to notice this connection.

For problems in which a positional learning effect is considered, an optimal solution is achieved by scheduling the jobs in the SPT order. Optimality of the SPT rule for problem $1\,|p_j r^a,\ a < 0|\sum C_j$ with a polynomial learning effect (3.11) has been proved by Biskup (1999), whereas the optimality of the SPT rule for problem $1\,|p_j \gamma^{r-1},\ 0 < \gamma < 1|\sum C_j$ with an exponential learning effect (3.12) has been proved

by Gordon et al. (2008).

Surprisingly, the status of the problem with a positional deterioration effect, i.e., with the factors that satisfy (3.2), has not been properly resolved prior to this study. Notice that the sequence $g(r)(n-r+1)$, $1 \leq r \leq n$, is non-monotone as the sequence $g(r)$, $1 \leq r \leq n$, is non-decreasing, where as the sequence $(n-r+1)$, $1 \leq r \leq n$, is decreasing. Thus, a simple priority rule obtained by a pairwise job interchange argument cannot deliver an optimal solution.

Indeed, Mosheiov (2005) demonstrates that for a problem with a polynomial deterioration effect (3.9), an optimal permutation cannot be obtained by a simple priority rule. Gordon et al. (2008) show that for the problem with an exponential deterioration effect (3.10) the LPT rule is optimal for $\gamma \geq 2$, but an optimal permutation cannot be obtained by a simple priority rule for $1 < \gamma < 2$. In Chapter 6 of this thesis, we show that an approach based on Algorithm Match can solve this problem in $O(n \log n)$ time. In fact, the approach works even if the positional factors $g(r)$, $1 \leq r \leq n$, are non-monotone.

Let us now consider single machine problems with job-dependent positional effects. Mosheiov and Sidney (2003) solve problem $1 \,|p_j r^{a_j},\ a_j < 0| \sum C_j$, in which a polynomial learning effect (3.13) is considered. They reduce the problem to an LAP of the form (2.2), with the cost function $c_{ji} = p_j i^{a_j} (n-i+1)$, and obtain an optimal solution in $O(n^3)$ time. The same approach can be extended to solve a problem with a general job-dependent positional effect of the form $p_j(r)$, in which case the cost function becomes $c_{ji} = p_j(i)(n-i+1)$; see Bachman and Janiak (2004). The latter formulation can also be used to solve the problem of minimising the total flow time for the model in which the positional effect is given as a linear function of the form (3.7).

Parallel Machine Scheduling: Suppose that in a feasible schedule $S(m)$ with m identical machines, there are $h^{[i]}$ jobs assigned to machine M_i, so that $\sum_{i=1}^{m} h^{[i]} = n$, and these jobs are processed in accordance with a permutation $\pi^{[i]} = \left(\pi^{[i]}(1), \pi^{[i]}(2) \ldots, \pi^{[i]}(h^{[i]})\right)$, where $1 \leq i \leq m$. A job $j = \pi^{[i]}(r)$ sequenced in position r of machine M_i is associated with a job-independent positional factor $g(r)$. Then it follows from (2.14) that the sum of completion times of all jobs can be written as

$$\sum_{j=1}^{n} C_j\left(S(m)\right) = \sum_{i=1}^{m} \sum_{r=1}^{h^{[i]}} \sum_{k=1}^{r} g(r)\, p_{\pi^{[i]}(k)} = \sum_{i=1}^{m} \sum_{r=1}^{h^{[i]}} g(r)\left(h^{[i]} - r + 1\right) p_{\pi^{[i]}(r)}.$$

Notice that for known values of $h^{[i]}, 1 \leq i \leq m$, the problem of minimising the total

flow time can be seen as a linear assignment problem with a product matrix (2.3), with arrays $\left(G^{[1]}, G^{[2]}, \ldots, G^{[m]}\right)$ and $\left(p_{\pi(1)}, p_{\pi(2)}, \ldots, p_{\pi(n)}\right)$, where $G^{[i]}$ represents a sub-array $\left(h^{[i]}g\left(1\right), \left(h^{[i]}-1\right)g\left(2\right), \ldots, g\left(h^{[i]}\right)\right)$, for each, i, $1 \leq i \leq m$. Similar to earlier instances, many authors fail to notice this connection.

Mosheiov (2001b) solves the problem $Pm\left|p_j r^a, a<0\right|\sum C_j$, in which a polynomial learning effect of the form (3.11) is considered on m identical machines. For given values of $h^{[i]}$, $1 \leq i \leq m$, he reduces this problem to a full form LAP of the form (2.2), with the cost function $c_{jr} = p_j r^a\left(h^{[i]}-r+1\right)$, $1 \leq r \leq h^{[i]}$, $1 \leq i \leq m$. All possible options for the values $h^{[i]}$, $1 \leq i \leq m$, are generated by computing the compositions of an integer n into exactly m summands. For each instance the resulting $n \times n$ LAP is solved by the Hungarian algorithm and the total running time for solving problem $Pm\left|p_j r^a, a<0\right|\sum C_j$ is estimated to be $O\left(n^{m+3}\right)$. Mosheiov and Sidney (2003) extend the same approach for solving a more general problem $Qm\left|p_j r^{a_j}, a<0\right|\sum C_j$, in which a job-dependent polynomial learning effect of the form (3.13) is considered on m uniform machines. For known values of $h^{[i]}$, $1 \leq i \leq m$, the cost function of the LAP is given as $c_{jr} = p_j r^{a_j}\left(h^{[i]}-r+1\right)/s_i$, $1 \leq r \leq h^{[i]}$, $1 \leq i \leq m$, and the overall running time for solving problem $Qm\left|p_j r^{a_j}, a<0\right|\sum C_j$ is estimated as $O\left(n^{m+3}\right)$. Notice the same approach can be further extended to solve an even more general problem with m unrelated parallel machines.

We feel that for problem $Pm\left|p_j r^a, a<0\right|\sum C_j$ the running time proposed by Mosheiov (2001b), is overestimated by at least two orders of n. This is due to the fact that (i) the number of generated instances of the problem is miscalculated as $O(n^m)$, instead of $O(n^{m-1})$ (see Section 2.2.4), and (ii) each generated instance is solved in $O(n^3)$ time by a full form LAP, instead of using a special form of the LAP which permits an optimal solution in $O\left(n \log n\right)$ time by Lemma 2.1. See Rustogi and Strusevich (2012b) for details.

Other Objective Functions

In this section, we review some more problems with positional effects which have been solved for somewhat exotic objective functions.

Minimising deviation from a common unrestricted due date: Biskup (1999) considers a single machine problem with a polynomial learning effect given by (3.11) in which the jobs are given a common unrestricted due date d. The due date is called unrestricted because in principle it is possible to schedule all jobs by that due date. If a job j completes before d, then its earliness $E_j = d - C_j$ is penalised; otherwise its

tardiness $T_j = C_j - d$ is penalised. The objective is to sequence the jobs in order to minimise the function $\sum_{j\in N}(wC_j + w'E_j + w''T_j)$, where w, w' and w'' are given positive constants. This is an extension of the model by Panwalkar, Smith and Seidmann (1982), who assume constant processing times. The authors obtain an optimal solution to this problem in $O(n^3)$ time by reducing it to a full form LAP of the form (2.2) with the cost function given by

$$c_{ji} = p_j i^a \min\{(i-1)w' + (n-i+1)w, (n-i+1)(w''+w)\}, 1 \leq i \leq n,\ j \in N.$$

Common due date assignment: Mosheiov (2001a) considers a single machine problem in which the jobs have to be assigned a common due date d, so that the function $\sum_{j\in N}(wd + w'E_j + w''T_j)$ is minimised. The jobs are subject to a polynomial learning effect. This is an extension of the model by Panwalkar, Smith and Seidmann (1982), who assume constant processing times. Again, the authors obtain an optimal solution to this problem in $O(n^3)$ time by reducing it to a full form LAP of the form (2.2) with the cost function given by

$$c_{ji} = \begin{cases} p_j i^a (nw + (i-1)w'), & 1 \leq i \leq u \\ p_j i^a (n+1-i)w'', & u+1 \leq i \leq n, \end{cases} \quad , j \in N,$$

where

$$u = \left\lceil \frac{w'' - w}{w' + w''} \right\rceil.$$

Minimising the weighted sum of total completion time and variation of completion times: Mosheiov (2001a) considers a single machine problem with a polynomial learning effect to minimise the sum of the functions $w\sum C_j$ and $(1-w)\sum_{i=1}^{n}\sum_{j=1}^{n}|C_i - C_j|$ for a given w, $0 \leq w \leq 1$. This is an extension of the model by Bagchi (1989), who assumes constant processing times. Again, the authors obtain an optimal solution to this problem in $O(n^3)$ time by reducing it to a full form LAP of the form (2.2) with the cost function given by

$$c_{ji} = p_j i^a \left[(2w-1)(n+1) + i(2 - 3w + n(1-w)) - i^2(1-w)\right],\ 1 \leq i \leq n,\ j \in N.$$

Notice that the published running times for each of the problems considered above have been overestimated by the authors. The cost functions c_{ji} for each of the problems can be seen as a product of the elements of two independent arrays. In such cases, the

resulting LAP is said to have a product matrix so that an optimal solution is found in $O(n \log n)$ time by Lemma 2.1.

If the positional factors are job-dependent, i.e., of the form $g_j(r)$, the corresponding problems are indeed, solvable in $O(n^3)$ time by reduction to a full form LAP; see, e.g., Mosheiov and Sidney (2003) and Mosheiov (2008) for details on the job-dependent analogue of the above mentioned problems.

Notice that for all the problems discussed so far, the found optimal solution has either been a simple priority rule (LPT/SPT), or has been computed using a full form LAP. Actually, if the positional factors are job-dependent, the latter approach remains the only available. However, in the case of job-independent positional factors, we show in Chapter 6 of this thesis that the search for an optimal strategy should not be limited to simple priority rules, and also that the use of a full form LAP is not required. In fact, our method based on Lemma 2.1, not only solves these problems for simple monotone effects like deterioration or learning, but works for arbitrary, possibly non-monotone, positional effects and finds an optimal solution in $O(n \log n)$ time.

3.2.2 Time-Dependent Effects

Typical time-dependent models are often of the form $p_j(\tau) = p_j + f(\tau)$ or $p_j(\tau) = p_j f(\tau)$, where $p_j(\tau)$ is the actual processing time of a job $j \in N$ scheduled at time $\tau \geq 0$, and p_j is its normal processing time. The function $f(\tau)$ is common for all jobs and takes only non-negative values. For a more general time-dependent effect, each job j can be associated with an individual function $f_j(\tau)$, so that the effect becomes job-dependent. Below we list out some of the most popular time-dependent models, in which the functions $f(\tau)$ and $f_j(\tau)$ are linear:

- $p_j(\tau) = p_j + a\tau$; a linear function of the start-time, where a is a job-independent constant, which is strictly positive for the deterioration environment and strictly negative for the learning environment.
- $p_j(\tau) = p_j + a_j\tau$; a linear function of the start-time, where a_j is a job-dependent constant, which is strictly positive for the deterioration environment and strictly negative for the learning environment.
- $p_j(\tau) = a_j\tau$; one of the simplest start-time dependent models, introduced by Mosheiov (1994) for a deterioration effect, where a_j is the deterioration rate of a job j.

- $p_j(\tau) = p_j\,(b + a\tau)$; a generalisation of the above model, where a is a job-independent constant, which is strictly positive for the deterioration environment and strictly negative for the learning environment.

Apart from the models listed above, several other models have been introduced in the recent past, in which a time-dependent model is combined with a positional effect, so that the actual processing time of a job j sequenced in position r and starting at time $\tau \geq 0$ is given by

$$p_j(\tau, r) = p_j\,(\tau)\, g(r),$$

where $p_j\,(\tau)$ is defined in one of the ways listed above and $g(r)$ is a general positional factor as defined in Section 3.2.1. One of the most general models that combines a time-dependent effect and a positional effect is studied by Yin and Xu (2011). The authors define the actual processing time of a job j sequenced in position r and starting at time $\tau \geq 0$ as $p_j(\tau, r) = p_j f(\tau) g(r)$, where $g(r)$ is a positional learning factor, while a start-time deterioration is given by $f(S_j)$, the function f being non-decreasing with a non-decreasing first derivative.

In this thesis, we only study start-time dependent models of the form

$$p_j(\tau) = p_j + a\tau, \tag{3.15}$$

and its combined version given by

$$p_j(\tau, r) = (p_j + a\tau)\, g(r). \tag{3.16}$$

We refer to models (3.15) and (3.16) as *job-independent* models for linear time-dependent effects. Recall that the job-independent constant a is strictly positive for a deterioration effect and strictly negative for a learning effect. For details and results related to other time-dependent models, job-dependent or non-linear, we refer the reader to the review by Cheng, Ding and Lin (2004), and a recent monograph by Gawiejnowicz (2008), which provides a state-of-the-art exposition of time-dependent scheduling models.

The combined model of the form (3.16) was first introduced by Wang (2006), for a time-dependent deterioration effect and a polynomial learning effect (3.11), so that $g(r) = r^b$, $b < 0$, and $a > 0$. An example of a situation in which this model may appear relevant is as follows: a human operator processes jobs on a certain equipment and during the process the equipment might be subject to wear and tear, i.e., it might deteriorate with time, however, the operator will gain additional skills by learning from

experience. Yang and Kuo (2009) also study this model with $g(r) = r^b$, $b < 0$, and $a > 0$, and solve the problems of minimising the makespan and the total flow time. Another variation of the model (3.16) is considered by Yang (2010), who study the case of time-dependent learning effect and a polynomial deterioration effect (3.9), so that $g(r) = r^b$, $b > 0$, and $a < 0$.

Problems with time-dependent effects can be represented in the three-field notation by providing the formula of the actual processing time $p_j(\tau)$, in the middle field β of the three-field descriptor. For example, the problem of minimising the makespan on a single machine that is subject to time-dependent deterioration of the form (3.15), can be denoted by $1\,|p_j + a\tau, a > 0|\,C_{\max}$.

We now discuss the algorithmic aspects of solving different problems related to models (3.15) and (3.16), that have been studied previously.

Minimising Makespan

We only consider single machine problems, as the problem of minimising the makespan in a parallel machine environment is known to be NP-hard.

Let the jobs be processed on a single machine in accordance with some permutation $\pi = (\pi(1), \ldots, \pi(n))$. If the machine is subject to a time-dependent effect of the form (3.15), then following the argument given by Browne and Yechiali (1990), the makespan can be written as

$$C_{\max}(\pi) = \sum_{r=1}^{n} p_{\pi(r)} (1+a)^{n-r}.$$

Notice that the problem of minimising the makespan can be seen as a linear assignment problem with a product matrix (2.3), with arrays $(W(1), W(2), \ldots, W(n))$ and $\left(p_{\pi(1)}, p_{\pi(2)}, \ldots, p_{\pi(n)}\right)$, where $W(r) = (1+a)^{n-r}, 1 \leq r \leq n$. Similar to earlier instances, many authors fail to notice this connection.

Using a pairwise job-interchange argument, Browne and Yechiali (1990) prove that irrespective of the sign of a, an optimal solution is obtained in $O(n \log n)$ time by scheduling the jobs in non-decreasing order of the values p_j/a. This implies that problem $1\,|p_j + a\tau, a > 0|\,C_{\max}$ can be solved by scheduling the jobs in an SPT order, while problem $1\,|p_j + a\tau, a < 0|\,C_{\max}$ can be solved by scheduling the jobs in an LPT order.

Let us now consider the combined model given by (3.16). If the jobs are processed on a single machine in accordance with some permutation $\pi = (\pi(1), \ldots, \pi(n))$, then following the argument given by Wang (2006), Yang and Kuo (2009) and Yang (2010),

the makespan can be written as

$$C_{\max}(\pi) = \sum_{r=1}^{n} p_{\pi(r)} \left(g(r) \prod_{i=r+1}^{n} (1 + ag(i)) \right), \ 1 \leq r \leq n.$$

Notice that the problem of minimising the makespan can be seen as a linear assignment problem with a product matrix (2.3), with arrays $(W(1), W(2), \ldots, W(n))$ and $\left(p_{\pi(1)}, p_{\pi(2)}, \ldots, p_{\pi(n)}\right)$, where $W(r) = g(r) \prod_{i=r+1}^{n} (1 + ag(i)), 1 \leq r \leq n$.

Indeed, Wang (2006) and Yang and Kuo (2009) notice this connection and use Lemma 2.1 to solve problem $1 \left| (p_j + a\tau) r^b, a > 0, b < 0 \right| C_{\max}$, with a time-dependent deterioration effect and a polynomial learning effect (3.11). They prove that an optimal solution is obtained if the jobs are scheduled in an SPT order. Yang (2010) also use the same technique to solve problem $1 \left| (p_j + a\tau) r^b, a < 0, b > 0 \right| C_{\max}$, with a time-dependent learning effect and a polynomial deterioration effect (3.9). They prove that an optimal solution is obtained if the jobs are scheduled in an LPT order.

Minimising Total Flow Time

Unlike the makespan objective, the problem of minimising the total flow time, can be solved optimally for both single and parallel machine environments.

Single Machine Scheduling: Assume that the jobs are processed on a single machine in accordance with some permutation $\pi = (\pi(1), \ldots, \pi(n))$. If the machine is subject to a time-dependent effect of the form (3.15), then following the argument given by Ng et al. (2002), the total flow time can be written as

$$\sum C_j(\pi) = \sum_{r=1}^{n} p_{\pi(r)} \left(\sum_{k=0}^{n-r} (1 + a)^k \right).$$

Notice that the problem of minimising the total flow time can be seen as a linear assignment problem with a product matrix (2.3), with arrays $(W(1), W(2), \ldots, W(n))$ and $\left(p_{\pi(1)}, p_{\pi(2)}, \ldots, p_{\pi(n)}\right)$, where $W(r) = \sum_{k=0}^{n-r} (1 + a)^k, 1 \leq r \leq n$. Similar to earlier instances, many authors fail to notice this connection.

Using a pairwise job-interchange argument, Ng et al. (2002) prove that an optimal solution to problem $1 \left| p_j + a\tau, a < 0 \right| \sum C_j$ can be obtained by scheduling the jobs in an SPT order. It can be easily verified (see, e.g., Kuo and Yang (2008b)) that the same ordering of jobs is also optimal for problem $1 \left| p_j + a\tau, a > 0 \right| \sum C_j$.

Let us now consider the combined model given by (3.16). If the jobs are processed on a single machine in accordance with some permutation $\pi = (\pi(1), \ldots, \pi(n))$, then following the argument given by Wang (2006), Yang and Kuo (2009) and Yang (2010), the total flow time can be written as

$$\sum C_j(\pi) = \sum_{r=1}^{n} p_{\pi(r)} \left(g(r) \left[\sum_{u=r}^{n} \prod_{i=r+1}^{u} (1 + ag(i)) \right] \right), \ 1 \leq r \leq n.$$

Notice that the problem of minimising the makespan can be seen as a linear assignment problem with a product matrix (2.3), with arrays $(W(1), W(2), \ldots, W(n))$ and $\left(p_{\pi(1)}, p_{\pi(2)}, \ldots, p_{\pi(n)}\right)$, where $W(r) = g(r) \left[\sum_{u=r}^{n} \prod_{i=r+1}^{u} (1 + ag(i)) \right], 1 \leq r \leq n.$

Indeed, Wang (2006) and Yang and Kuo (2009) notice this connection and use Lemma 2.1 to prove that an optimal solution to problem $1 \left| (p_j + a\tau) r^b, a > 0, b < 0 \right| \sum C_j$ can be obtained by the SPT rule. Yang (2010) also use the same technique to solve problem $1 \left| (p_j + a\tau) r^b, a < 0, b > 0 \right| \sum C_j$, and prove that an optimal solution is obtained by the SPT rule.

For the same special case of (3.16), i.e., with $g(r) = r^b$, problems involving other objective functions, including various generalisations of the functions considered in Section 3.2.1, are handled by Qian and Steiner (2012) (for $a > 0$ and $b < 0$) and by Yang (2010) (for $a < 0$ and $b > 0$). The authors reduce these enhanced problems to solving a linear assignment problem with a product matrix (2.3) an obtain an optimal solution in $O(n \log n)$ time.

Parallel Machine Scheduling: Kuo and Yang (2008b) solve the problem $Pm \left| (p_j + a\tau) \right| \sum C_j$, in which a time-dependent effect of the form (3.15) is considered on m identical machines. They prove using Lemma 2.1, that irrespective of the sign of a, an optimal schedule can be found in $O(n \log n)$ time by applying the SPT list scheduling algorithm.

3.2.3 Cumulative Effects

This is a fairly recent model for changing processing times, first introduced by Kuo and Yang (2006a, 2006b) for a learning environment. Assume that the jobs are processed on a single machine in accordance with some permutation $\pi = (\pi(1), \ldots, \pi(n))$. In the model introduced by Kuo and Yang (2006a, 2006b), the actual processing time of a job

$j = \pi(r)$ that is sequenced in position r of a permutation π is given by

$$p_j(r) = p_j \left(1 + \sum_{k=1}^{r-1} p_{\pi(k)}\right)^Z, \tag{3.17}$$

where Z is a given constant which is strictly negative for a learning environment and strictly positive for a deterioration environment. The deterioration case with $Z > 0$ is initiated by Gordon et al. (2008).

Notice that a cumulative effect can be seen as version of time-dependent effects. The only difference being that the processing time of a job j is not dependent on the actual time elapsed, but is dependent on the sum of the normal processing times of the jobs scheduled earlier. For this reason, several authors including Kuo and Yang (2006a, 2006b), often refer to this effect simply as a time-based effect.

Kuo and Yang (2006a, 2006b) prove that the problem of minimising the makespan and the problem of minimising the total flow time on a single machine which is subject to a cumulative effect of the form (3.17), with $Z \leq 0$, can be solved optimally by scheduling the jobs in an SPT (2.13) order. They prove the optimality of this result by establishing the following preliminary statements.

Lemma 3.1. *The following relation holds*

$$1 - (1+t)^Z + Zt(1+t)^{Z-1} \geq 0, \text{ if } Z \leq 0 \text{ and } t \geq 0.$$

Lemma 3.2. *The following relation holds*

$$\lambda\left(1 - (1+t)^Z\right) - (1 - (1+\lambda t))^Z \geq 0 \text{ if } \lambda \geq 1,\ t \geq 0 \text{ and } Z \leq 0.$$

Further, a pairwise job interchange argument is provided which proves the optimality of the SPT rule.

In the recent years, several variations of this model have appeared, see, e.g., Wu, Yin and Cheng (2011), who list about a dozen of models with cumulative learning, in which $p_j(r)$ is expressed in different ways in terms of $\sum_{k=1}^{r-1} p_{\pi(k)}$. In general, all models with cumulative effects can be written in the form

$$p_j(r) = p_j f\left(\sum_{k=1}^{r-1} p_{\pi(k)}\right),$$

where f is non-increasing function for a learning environment and a non-decreasing function for a deterioration environment. For all the models described, the SPT rule

has been proved to be the optimal policy for the problem of minimising the makespan and for the problem of minimising the total flow time. Typically, authors use a version of Lemmas 3.1 and 3.2, along with a pairwise job interchange argument to prove the optimality of SPT for different models.

Among the most general models, is the following in which a cumulative deterioration effect is combined with a general positional learning effect by

$$p_j(r) = p_j f\left(\sum_{i=1}^{r-1} p_{\pi(i)}\right) g(r),$$

as used by Yin et al. (2009). Again, the authors prove that the SPT rule can be used to solve the problem of minimising the makespan or the total flow time.

Models with precedence constraints and cumulative deterioration effects as given by (3.17), for $Z = 1$ and $Z = 2$, are studied by Gordon et al. (2008).

3.3 Models with Rate Modifying Activities

In this section, we review some of the prior studies that integrate changing processing times and rate-modifying activities. This line of research has only started in the last 4-5 years, with Kuo and Yang (2008a) publishing one of the first papers of its kind. Almost all of the papers that consider such models deal study a deterioration effect and use rate-modifying (maintenance) activities to negate these effects. In all existing models, it is assumed that each of the RMPs are identical and they are able to fully restore the machine conditions. Typically, a decision-maker would need to known an optimal permutation of jobs and the optimal number of RMPs to include in the schedule, so that a given objective function can be minimised. The latter is an important question, because the RMPs are of a finite duration. Including many of them in a schedule, may reduce the processing time of the jobs on one hand, but on the other hand, may increase the overall duration of the schedule. The durations of the RMPs are either known to be constant or are given as a linear function of its start time, i.e.,

$$D_{RMP} = \alpha\tau + \beta, \tag{3.18}$$

where τ is the start-time of the RMP, measured from the time the previous RMP was completed, and α and β are positive parameters that define the RMP. The formula (3.18) implies that the duration of the RMP becomes larger if it is performed later in a schedule. Such a model for RMP durations was introduced by Kubzin and Strusevich

(2005, 2006).

In the next two sub-sections, we review prior models in which rate-modifying activities are studied with positional and time-dependent effects. Note that there is no prior history of problems with cumulative effects and rate-modifying activities.

In this review, to denote the presence of an RMP in the schedule, we include the term "RMP" in the middle field of the standard three-field notation. By default, it is assumed that the durations of the RMPs are start-time dependent (3.18). If they are known to be of constant duration, we denote them by "$RMP\,[0]$". Additionally, if the number of RMPs is known in advance and is not a decision variable, we use the term "$RMP\,(K)$" instead, to denote that K RMPs are included in a schedule. Notice that in a schedule with K RMPs, the jobs are divided into a total of $K+1$ groups, one placed before the first RMP and one after each of the K RMPs.

Unlike the earlier parts of this review, we do not give a full mathematical formulation of the problems presented here, as most of them will be revisited in the main body of this thesis. We only talk about the algorithmic principles behind solving these problems and their resulting time complexity.

3.3.1 Positional Effects

Minimising Makespan

Kuo and Yang (2008a) study problem $1\,|p_j r^a, a>0, RMP\,[0]|\,C_{\max}$, with a polynomial deterioration effect (3.9) and identical RMPs, whose duration is assumed to be constant. In order to solve this problem, Kuo and Yang (2008a) turn to solving a series of sub-problems $1\,|p_j r^a, a>0, RMP\,[0]\,(k-1)|\,C_{\max}$, with $k-1$ number of RMPs. To solve each problem $1\,|p_j r^a, a>0, RMP\,[0]\,(k-1)|\,C_{\max}$ they prove the so-called *group balance principle*, according to which, in an optimal schedule with k groups, the difference between the number of jobs in any two groups is at most one. They further prove using Lemma 2.1 that in each group the jobs are sequenced in the LPT order. As a result, an algorithm for solving problem $1\,|p_j r^a, a>0, RMP\,[0]\,(k-1)|\,C_{\max}$ scans the jobs in the LPT order and assigns them one by one to the smallest available position across all k groups. Such an algorithm requires $O(n)$ time to output an optimal schedule and the optimal value of the makespan, provided that the LPT sequence of jobs is known. Trying all possible values of k, $1 \leq k \leq n$, they obtain a solution to the original problem $1\,|p_j r^a, a>0, RMP\,[0]|\,C_{\max}$ as the best of all found schedules. The resulting running time is $O\left(n^2\right)$. It should be noted that Kuo and Yang (2008a) make

a mistake when they claim that their algorithm requires $O(n \log n)$ time: they do not take into account the linear time that is needed to compute the value of the makespan for each k, $1 \leq k \leq n$. In Section 6.7, we prove that the running time of problem $1\,|p_j r^a, a > 0, RMP|\, C_{\max}$ can be reduced to $O(n \log n)$.

Zhao and Tang (2010) study a problem similar to Kuo and Yang (2008a), but with a job-dependent polynomial effect. The resulting problem is denoted as $1\,|p_j r^{a_j}, a_j > 0, RMP\,[0]|\, C_{\max}$. In order to solve this problem, Zhao and Tang (2010) also turn to solving a series of sub-problems $1\,|p_j r^{a_j}, a_j > 0, RMP\,[0]\,(k-1)|\, C_{\max}$, and prove the group balance principle. This principle allows them to guess the number of jobs to schedule in each of the k groups. With knowledge of the n candidate positions across all groups, they reduce problem $1\,|p_j r^{a_j}, a_j > 0, RMP\,[0]\,(k-1)|\, C_{\max}$ to a linear assignment problem. The cost of scheduling a job j at position i in some group is given by $c_{ji} = p_j i^{a_j}$. As a result, problem $1\,|p_j r^{a_j}, a_j > 0, RMP\,[0]\,(k-1)|\, C_{\max}$ is solved in $O\left(n^3\right)$ time. Trying all possible values of k, $1 \leq k \leq n$, they obtain a solution to the original problem $1\,|p_j r^{a_j}, a_j > 0, RMP\,[0]|\, C_{\max}$ as the best of all found schedules. The resulting running time is $O\left(n^4\right)$.

Yang and Yang (2010a) study a problem similar to Zhao and Tang (2010), but the durations of the RMPs are given as a linear function of their start time (3.18). The resulting problem can be denoted as $1\,|p_j r^{a_j}, a_j > 0, RMP|\, C_{\max}$. In order to solve this problem, Yang and Yang (2010a) also turn to solving a series of sub-problems $1\,|p_j r^{a_j}, a_j > 0, RMP\,(k-1)|\, C_{\max}$. For problem $1\,|p_j r^{a_j}, a_j > 0, RMP\,(k-1)|\, C_{\max}$, they prove the group-balance principle for the first $k-1$ groups, which allows them to guess the number of jobs in those groups. However, for the last group, they are not able to guess the number of jobs, so they resort to enumerating all possible options for the number of jobs in that group. As a result, the running time needed to solve this problem is n times greater than that required by Zhao and Tang (2010), thereby making problem $1\,|p_j r^{a_j}, a_j > 0, RMP|\, C_{\max}$ solvable in $O\left(n^5\right)$ time. In Section 7.4, we prove that a full enumeration of the number of jobs in the last group is not required, and problem $1\,|p_j r^{a_j}, a_j > 0, RMP|\, C_{\max}$ can be solved in $O\left(n^4\right)$ time.

Minimising Total Flow Time

Yang and Yang (2010b) study problem $1\,|p_j r^a, a > 0, RMP\,(K)|\, \sum C_j$, with a polynomial deterioration effect (3.9) and a known number, K, of RMPs. They prove that if the number of jobs in each group is known, problem $1\,|p_j r^a, a > 0, RMP\,(K)|\, \sum C_j$ reduces to a linear assignment problem with a product matrix (2.3), so that an optimal permutation of jobs can be found in $O\left(n \log n\right)$ time. To find the optimal number of

jobs in each group, they enumerate all possible options in which n jobs can be divided into $K+1$ groups. Recall from Section 2.2.4, that this can be done in $n^K/K!$ ways. Thus, problem $1\left|p_j r^a, a>0, RMP\left(K\right)\right|\sum C_j$ can be solved in $O\left(n^{K+1}\log n\right)$ time.

Ji and Cheng (2010) study problem $1\left|p_j r^{a_j}, a_j<0, RMP\left[0\right]\left(K\right)\right|\sum C_j$, with a job-dependent polynomial learning effect (3.13) and a known number, K, of RMPs. This is the first paper of its kind, which combines a learning effect with a rate-modifying activity. The RMPs are aimed at further enhancing the learning rate of the operator. This is achieved by multiplying the positional factors r^{a_j} with a constant λ, $0\leq\lambda\leq 1$. They prove that if the number of jobs in each group is known, problem $1\left|p_j r^{a_j}, a_j<0, RMP\left[0\right]\left(K\right)\right|\sum C_j$ reduces to a full form linear assignment problem (2.2), so that an optimal permutation of jobs can be found in $O\left(n^3\right)$ time. To find the optimal number of jobs in each group, they enumerate all possible options in which n jobs can be divided into $K+1$ groups. As a result, problem $1\left|p_j r^{a_j}, a_j<0, RMP\left[0\right]\left(K\right)\right|\sum C_j$ can be solved in $O\left(n^{K+3}\right)$ time. They further extend this model to a parallel machine environment, and show that an optimal solution can be found in $O\left(n^{m+K+2}\right)$ time.

Notice that both problems reviewed in this section require a full enumeration for the number of jobs in each group, thereby resulting in polynomial running times of a greater order. Although we are not able to improve upon this running time for the problem of minimising the total flow time, we show in Chapter 9 of this thesis, that the same (or marginally higher, in some cases) running time is achieved for a much more general problem, one with combined positional and time-dependent effects, simultaneous learning and deterioration effects, and distinct rate-modifying activities.

3.3.2 Time-Dependent Effects

For time-dependent models, very few results exist which study the effect of rate-modifying activities.

Lodree and Geiger (2010) study problem $1\left|a_j\tau, RMP\left[0\right]\left(1\right)\right|C_{\max}$, with a time-dependent effect of the form $p_j\left(\tau\right)=a_j\tau, a_j\geq 1$, and a single RMP in the schedule. They provide an optimal policy to determine the number of jobs to be included in each of the two created groups. According to this policy, if n is even, both groups should contain $\left(\frac{n}{2}+1\right)$ jobs, whereas if n is odd, one group should contain $\left(\frac{n+1}{2}\right)$ jobs and the other should contain $\left(\frac{n+3}{2}\right)$ jobs.

Yang and Yang (2010b) study problem $1\left|p_j+a\tau, a>0, RMP\left(K\right)\right|\sum C_j$, with a linear time-dependent deterioration effect of the form (3.15) and a known number,

K, of RMPs. Similar to the solution of problem $1\left|p_j r^a, a>0, RMP\left(K\right)\right|\sum C_j$, they prove that if the number of jobs in each group is known, the problem reduces to a linear assignment problem with a product matrix (2.3), so that an optimal permutation of jobs can be found in $O\left(n\log n\right)$ time. Further, they enumerate all possible options in which n jobs can be divided into $K+1$ groups, and provide an optimal solution to problem $1\left|p_j+a\tau, a>0, RMP\left(K\right)\right|\sum C_j$ in $O\left(n^{K+1}\log n\right)$ time.

The same method is extended by Yang (2010) and Yang (2012), to study different versions of problem $1\left|\left(p_j+a\tau\right)r^b, RMP\left(K\right)\right|F$, $F\in\{C_{\max},\sum C_j\}$, with a combined effect of the form (3.16) and a known number, K, of RMPs. Yang (2010) considers the case in which a single RMP ($K=1$) is to be scheduled on a machine which is subject to time-dependent learning ($a<0$) and polynomial deterioration ($b>0$) effects. Yang (2012) considers the case in which multiple RMPs are to be scheduled on a machine which is subject to time-dependent deterioration ($a>0$) and polynomial learning ($b<0$) effects. For their respective models, both the papers claim to solve the problem of minimising the makespan and the problem of minimising the total flow time in $O\left(n^{K+1}\log n\right)$ time each. We notice however, that both papers underestimate the running time needed to solve the problem of minimising the total flow time. This is because, the authors ignore the running time needed to compute the values of the positional weights (costs) that must be input in the reduced LAP. It can be easily verified that this running time is $O\left(n^2\right)$, for the problem of minimising the total flow time. Thus, for both cases, problem $1\left|\left(p_j+a\tau\right)r^b, RMP\left(K\right)\right|\sum C_j$ cannot be solved in less than $O\left(n^{K+2}\right)$ time.

Notice that apart from problem $1\left|a_j\tau, RMP\left[0\right]\left(1\right)\right|C_{\max}$ studied by Lodree and Geiger (2010), all other problems reviewed in this section require a full enumeration for the number of jobs in each group. To the best of our knowledge, there are no other existing results which solve the problems related to time-dependent effects and rate-modifying activities in a different way. In Chapter 8 of this thesis, we consider a model with a pure time-dependent effect of the form (3.15) and show that even in the presence of distinct rate-modifying activities, the problem of minimising the makespan can be solved by an efficient polynomial algorithm, which does not depend on full enumeration.

3.4 Conclusion

In this chapter, we review several prior studies related to scheduling with changing processing times. Below we list out some of the common features that we notice in the

existing results:

- In models with positional effects, the positional factors are always given as the values of a specific function, e.g., polynomial, exponential, etc. However, none of the authors actually use any of the mathematical properties associated with these functions. Thus, by using a specific function to model positional effects, the authors unnecessarily limit the scope of their models.

- In problems without rate-modifying activities, many authors solve the problem at hand by using a simple priority rule, and they prove its optimality by a pairwise job interchange argument. Such rules may work in some simple cases, but are not applicable in general. We notice that in most cases these priority rules are in fact special cases of the matching algorithm, so that they work because the matching algorithm works, and if they do not work, the matching algorithm still will. Both these approaches, permit a running time of $O(n \log n)$. By restricting themselves to simple priority rules, the authors severely restrict the range of problems they can solve.

- We notice on multiple occasions that the authors have wrongly used a full form of the linear assignment problem as a subroutine of their solution procedure (requires $O(n^3)$ time for a problem with n jobs). In such cases, the authors fail to notice that LAP is in fact, of a special structure which permits the use of faster matching algorithm. The latter approach reduces the running time of the resulting algorithms by up to two orders, and carries over to a wider range of models, with more general positional effects. See our survey Rustogi and Strusevich (2012b), in which we systematically review a wide variety of problems which can be solved faster by use of Algorithm Match.

- In all problems with rate-modifying activities, it is noticed that an important decision to make is to determine the optimal number of jobs to schedule in each group. Once this is found, the problem easily reduces to a linear assignment problem, either in its full form or in the reduced form. In all the papers reviewed, establishing a group-balance principle is a preferred technique to guess the number of jobs in a group. In cases when a group-balance principle does not work, the authors resort to full enumeration of the possible ways in which n jobs can be divided in to a given number of groups. We feel that the group balance principle is not a very robust tool, when it comes to determining the optimal number of jobs in a group. Similar to the priority rules, such rules may work in some simple cases, but are not applicable in general. For example, in

the problem considered by Yang and Yang (2010a), the group balance principle is only applicable to some of the group in the schedule. Solely relying on this approach often restricts the authors in the range of models they can handle.

Taking into account all of these points, in the next chapter, we present a generic framework which allows us to systematically study scheduling problems with non-constant processing times and the effect of rate-modifying activities, for a very wide range of models.

Part II

Methodological Aspects

CHAPTER 4

General Framework for Solving Problems with Rate-Modifying Activities

The main focus of this thesis is to study the effect of incorporating rate-modifying activities in scheduling models with changing processing times. We consider three models for changing processing times, namely, positional, time-dependent, and cumulative. In Chapter 3 we introduce each of these three models and provide a brief overview of prior research that is related to them. It can be noticed that only a handful of papers exist which address the issue of combining rate-modifying activities with these models. Our work in this area has enabled us to further generalise and improve known results so that they can be applied to many more practical situations. In this chapter, we provide a general framework which is used to systematically study the effect of rate-modifying activities throughout this thesis.

4.1 Notation and Problem Description

In all the problems considered in this thesis, the jobs of set $N = \{1, 2, \ldots, n\}$ have to be processed on some machines. The jobs are available for processing at time zero and are independent, i.e., there are no precedence constraints and any processing sequence is feasible. At time zero, the machines are assumed to be in perfect processing state, and as more jobs are processed their processing conditions change according to one (or more) of the three models listed above.

All problems studied in this thesis can be denoted in the three-field notation by $\alpha\,|\varphi, \Psi|\,F$, where the first field points out the machine environment $\alpha \in \{1, Pm, Qm, Rm\}$, and the third field points out our objective function. The first item in the middle field is used to describe how the processing times change, and the

second term in the middle field is used to describe the type of rate-modifying period (RMP) included in the schedule.

If $\Psi = \varnothing$, we consider a problem with no RMPs at all, so that the jobs scheduled on a machine are organised as one group.

If $\Psi = RMP$, we consider a general situation, in which the decision-maker is presented with a total of $K \geq 0$ possible rate-modifying activities, which can be either distinct or alike. Each RMP can have a different effect on the machine conditions, so that they do not necessarily restore the machine to its default "as good as new" state. If the selected RMPs are different in nature, e.g., one RMP replaces the cutting tool of the machine, whereas the other refills gas in the system, then occurrence of such a situation is natural. However, such a situation can also arise if the RMPs are identical, but their efficiency in performing the desired task keeps changing depending on their position in the schedule. For instance, consider a scenario in which the RMP is aimed at improving the machine conditions by running maintenance. In real life it is often observed that even after a maintenance activity, some wear and tear might still remain in the machine and if the same RMPs are performed every time, this deviation might get accumulated.

If, out of the available K RMPs, $k-1$ of them are selected and included in a schedule on some machine, then the jobs on that machine will be divided into k, $1 \leq k \leq K+1$ groups, one to be scheduled before the first RMP and one after each of the $k-1$ RMPs. Since the RMPs are known to effect the machine conditions differently, it follows that each of the k groups might treat the jobs scheduled in them in a different way. As a result, the actual processing time of a job will be dependent on the group it is scheduled in. We refer to such an effect as a *group-dependent* effect. This is the first study, in which such effects are studied. Recall from Section 3.3, that in the earlier papers which studied changing processing times with rate-modifying activities, it is assumed that all RMPs are identical and they restore the machine to the same state. As a result, the created groups are indistinguishable and it does not matter which group a job is scheduled in. We refer to such an effect as a *group-independent* effect.

Below we give an illustration of a possible application of group-dependent effects.

Example 4.1. A human operator uses a tool to process n jobs. During the processing of the jobs, the tool undergoes deterioration whereas the operator undergoes both deterioration and learning effects. It is known that two RMPs will be included in the schedule. The first RMP is a maintenance period which restores the machine to its original condition. However, the deterioration rate of the machine becomes greater after the maintenance period, since original spare parts are not used. This RMP also

provides the operator with sufficient rest, so that after the first RMP the operator is as fresh as he/she was at the beginning of the schedule. Additionally, the operator gets a technical briefing from his supervisor so his learning curve changes. The second RMP does not repair the machine at all, instead, a new operator is brought in. Below, we give details about how these effects are modelled mathematically, and compute the resulting positional factors. We distinguish between the positional factors associated with the machine and the operator by using the subscript "m" for the machine and "w" for the operator (worker), respectively.

In a feasible schedule the jobs will be split into $k = 3$ groups. The machine suffers a positional exponential deterioration effect (3.10) with a rate $A_1 > 0$ before the first RMP and $A_2 > A_1$, after the second RMP. As a result, the positional factors associated with the machine for the three groups are given as $g_m^{[1]}(r) = (A_1)^{r-1}$, $g_m^{[2]}(r) = (A_2)^{r-1}$ and $g_m^{[3]}(r) = (A_2)^{n^{[2]}+r-1}$ respectively, where $n^{[2]}$ represents the number of jobs scheduled in the second group. The two operators are subject to a positional polynomial deterioration effect (3.9), with rates $B_1 > 0$ and $B_2 > 0$, respectively. They are also subject to positional polynomial learning effects (3.11). The rate with which Operator 1 learns before the first RMP is $C_1 < 0$, while the rate with which he learns after the RMP is $C_2 < C_1 < 0$. The learning rate of Operator 2 is given by $C_3 < 0$. As a result, the positional factors associated with the operators for the three groups are given as $g_w^{[1]}(r) = r^{B_1+C_1}$, $g_w^{[2]}(r) = r^{B_1}\left(n^{[1]}+r\right)^{C_2}$ and $g_w^{[3]}(r) = r^{B_2+C_3}$, respectively, where $n^{[1]}$ represents the number of jobs scheduled in the first group.

Thus, the positional factors for the entire processing system can be given as

$$\begin{aligned} g^{[1]}(r) &= g_m^{[1]}(r)\, g_w^{[1]}(r) = (A_1)^{r-1} r^{B_1+C_1}; \\ g^{[2]}(r) &= g_m^{[2]}(r)\, g_w^{[2]}(r) = (A_2)^{r-1} r^{B_1}\left(n^{[1]}+r\right)^{C_2}; \\ g^{[3]}(r) &= g_m^{[3]}(r)\, g_w^{[3]}(r) = (A_2)^{n^{[2]}+r-1} r^{B_2+C_3}, \end{aligned}$$

where the super-script $[x]$, $1 \leq x \leq 3$, is used to distinguish between the positional factors for different groups. Notice that this model allows us to assume (unlike, e.g., Yang (2010)), that during an RMP, if the operator is not replaced, he/she does not lose his/her skills which were improved due to learning in the earlier groups of the schedule. Similarly, if during an RMP a machine is not fully repaired, our model is capable of handling the resulting situation in which the deterioration effect from the group before the RMP must be carried forward to the next group. These instances are captured by adjusting the relative position of a job in the relevant group. For example, the learning factor involved in the computation of $g^{[2]}(r)$ is given by $\left(n^{[1]}+r\right)^{C_2}$ (implying that Operator 1 has completed $n^{[1]}$ jobs before starting group 2), and the deterioration

factor involved in the computation of $g^{[3]}(r)$ is given by $(A_2)^{n^{[2]}+r-1}$ (implying that since its last maintenance, the machine has completed $n^{[2]}$ jobs before starting group 3).

Notice that during each RMP no job processing takes place. In a very general setting, each of the K available RMPs, are associated with its own set of duration parameters, $\alpha^{[y]}$ and $\beta^{[y]}$, $1 \leq y \leq K$. This allows us to further generalise the linear model for start-time dependent durations (3.18), so that the duration D_{RMP} of an RMP can be given as

$$D_{RMP} = \alpha^{[y]}\tau + \beta^{[y]}, \tag{4.1}$$

where τ is the start-time of the RMP, measured from the time the previous RMP was completed, and $\alpha^{[y]}$ and $\beta^{[y]}$ are parameters that define the RMP. The value of $\beta^{[y]}$ is always positive; this parameter represents a fixed amount of time that must be spent if an RMP with an index y, $1 \leq y \leq K$, is performed. It can be seen as the duration of some standardised procedures that must be completed during an RMP. The value of $\alpha^{[y]}$ can be negative or positive, depending on the nature of the RMP. If $\alpha^{[y]}$ is positive, the RMP can be considered to be a maintenance period (MP) and the formula (4.1) implies that the duration of the MP becomes larger if it is performed later in a schedule. Such an extended formulation as (4.1) was first introduced in our paper Rustogi and Strusevich (2012a). If $\alpha^{[y]}$ is negative, the formula (4.1) implies that the duration of the RMP becomes smaller if it is performed later in a schedule. The value of $\alpha^{[y]}$ should be chosen such that the overall duration of the RMP always remains positive. A special case of (4.1) is the case in which the duration of the RMP is given by a constant, so that $\alpha^{[y]} = 0$, $1 \leq y \leq K$; see, e.g., Kuo and Yang (2008a) and Zhao and Tang (2010), who consider such a case with $\beta^{[y]} = \beta$, $1 \leq y \leq K$.

Notice that if it is known that the RMP is, in fact an MP, we will use $\Psi = MP$ instead of $\Psi = RMP$ in the three-field notation. In such a case, it is implied that the value of $\alpha^{[y]}$ is non-negative. Moreover, if it is known that duration of an RMP is constant, we will use $\Psi = RMP\,[0]$, signifying that the value $\alpha^{[y]} = 0$, $1 \leq y \leq K$. For a general situation with rate-modifying activities, we stick to the notation $\Psi = RMP$.

If a single machine environment is considered, the problems can be denoted by $1\,|\varphi, \Psi|\,F$.

To obtain an optimal solution to problem $1\,|\varphi|\,F$, with no RMPs, the only decision a decision-maker must make is regarding the optimal permutation of jobs in a single group. Prior problems of such a kind are reviewed in Section 3.2. However, to obtain an optimal solution to problem $1\,|\varphi, RMP|\,F$ a decision-maker must make the following decisions:

Decision 1. *The number of RMPs:* If $k-1$ RMPs are included in the schedule, the jobs are divided into k groups. Determine the optimal value of k, $1 \leq k \leq K+1$.

Decision 2. *The choice of RMPs:* From a given list of K RMPs, choose $k-1$, $1 \leq k \leq K+1$, RMPs that are included in the schedule.

Decision 3. *The sequence of RMPs:* Determine the optimal order in which the selected $k-1$ RMPs are scheduled on a machine.

Decision 4. *An optimal permutation of jobs:* For each job $j \in N$, determine the group and the position within that group, that it must be scheduled in.

In this thesis we consider many versions of problem $1\,|\varphi, \Psi|\,F$, which can be differentiated by changing the values of φ, Ψ and F in the problem formulation.

- φ is assigned based on whether the problem under consideration follows a positional, time-dependent or cumulative model. Problems are further classified depending on whether the machine is undergoing a deterioration, learning or an arbitrary effect. An arbitrary effect could arise in problems that combine a deterioration and learning effect.

- Ψ is assigned based on what kind of rate-modifying activities, if any, are included in the schedule. Different problems can be classified as follows:

  (i). $\Psi = \varnothing$, or otherwise;

  (ii). The RMPs are distinct or identical;

  (iii). The RMPs result in a group-dependent or group-independent effect;

  (iv.) The duration of the RMPs are constant or start-time dependent of the form (4.1).

- F is assigned based on which objective function is being optimised. We mainly concentrate on two objectives, minimising the makespan, i.e., $F = C_{\max}$, and minimising the total flow time, i.e., $F = \sum C_j$.

If a parallel machine environment is considered, the problems can be denoted in the three-field notation by $\alpha\,|\varphi, \Psi|\,F$, where $\alpha \in \{Pm, Qm, Rm\}$. The problem of minimising the makespan on parallel machines is NP-hard, thus, we concentrate on the problems of minimising the total flow time. We consider several versions of problem $\alpha\,|\varphi, \Psi|\sum C_j$, differentiating between them based on the values α, φ and Ψ. Rest of the notation and problem formulation is similar to the single machine case and will be discussed in more detail in Chapter 12.

4.2 General Methodology

In this section, we outline some general principles that we often use in the thesis for solving different versions of problem $1\,|\varphi, \Psi|\,F$. Let us begin with problems in which $\Psi = \varnothing$, i.e., no rate-modifying activities are included in the schedule.

4.2.1 Models without Rate-Modifying Activities

Assume that in all versions of problem $1\,|\varphi|\,F$, jobs from the set N are processed on a single machine in accordance with some permutation $\pi = (\pi(1), \ldots, \pi(n))$.

Recall from Section 3.2, that many versions of problem $1\,|\varphi|\,F$ can be solved by reducing them to either a full form LAP (2.2), or to a special form of the LAP with a product matrix (2.3). Typically, such a reduction is possible if the value of an objective function can written in the form

$$F = \sum_{r=1}^{n} W_j(r) p_{\pi(r)} + \Gamma, \tag{4.2}$$

where Γ is a constant and $W_j(r)$ is a *positional weight* associated with a job $j = \pi(r)$ sequenced in a position r, $1 \leq r \leq n$. A positional weight can be defined as a quantity which when multiplied by the normal processing time of a job j, gives the overall contribution of job j to the objective function F.

The function F of the form (4.2) can be minimised by reducing the problem to an LAP of the form (2.2), with the cost function $c_{ji} = W_j(i)p_j$, $1 \leq i \leq n$, $1 \leq j \leq n$. It follows from Section 2.2.3 that an optimal sequence of jobs can be found in $O(n^3)$ time by running the Hungarian Algorithm.

If however, it is observed that the value of an objective function can written in the form

$$F = \sum_{r=1}^{n} W(r) p_{\pi(r)} + \Gamma, \tag{4.3}$$

so that the resulting cost function of the LAP is given by $c_{ji} = W(i)p_j$, $1 \leq i \leq n$, $1 \leq j \leq n$, then such an LAP is said to have a product matrix and can be solved by Algorithm Match; see Section 2.2.3. Below we provide a formal description of an algorithm based on Algorithm Match, that minimises the objective function F of the form (4.3).

Algorithm Match1

INPUT: An instance of problem $1\,|\varphi|\,F$ whose objective function can written in the form (4.3).

OUTPUT: An optimal sequence of jobs π^*

Step 1. If required, renumber the jobs in LPT order, i.e., in non-increasing order of the values p_j.

Step 2. Compute the constant term Γ and the positional weights $W(r)$, $1 \leq r \leq n$, for the given problem.

Step 3. Sort the computed positional weights in non-decreasing order of their values, and store them in a list $L := (\gamma_1, \gamma_2, \ldots, \gamma_n)$, where γ_j, $1 \leq j \leq n$, denotes the j-th largest element in the list L.

Step 4. Match job j to the j-th element in list L, so that the optimal value of the function F is given by $\sum_{j=1}^{n} \gamma_j p_j + \Gamma$.

As follows from Lemma 2.1, Algorithm Match1 will minimise the function F of the form (4.3) by assigning jobs with small processing times to positions with large positional weights. Since the computed positional weights can be non-monotone, we require $O(n \log n)$ time to create the sorted list L. The following statement holds.

Lemma 4.1. *If the LPT order of the jobs is known in advance, Algorithm Match1 minimises an objective function F of the form (4.3), in $O(T(W) + O(n \log n))$ time, where $T(W)$ is the time taken to compute the positional weights $W(r)$, $1 \leq r \leq n$.*

4.2.2 Models with Rate-Modifying Activities

To solve all versions of problem $1\,|\varphi, RMP|\,F$, we first assume that Decisions 1-3 are taken in advance, so that we know that a total of $k-1$ MPs have been included in the schedule. As a result the jobs are split into k, $1 \leq k \leq K+1$, groups. Denote the resulting problem as $1\,|\varphi, RMP\,(k-1)|\,F$.

To solve problem $1\,|\varphi, RMP\,(k-1)|\,F$ consider a schedule $S(k)$ with a permutation of jobs $\pi = \left(\pi^{[1]}, \pi^{[2]}, \ldots, \pi^{[k]}\right)$. Assume that each group contains a total of $n^{[x]}$ jobs, so that $\pi^{[x]} = \left(\pi^{[x]}(1), \pi^{[x]}(2), \ldots, \pi^{[x]}\left(n^{[x]}\right)\right)$, $1 \leq x \leq k$, where $\sum_{x=1}^{k} n^{[x]} = n$. Recall from Section 3.3, that many versions of problem $1\,|\varphi, RMP\,(k-1)|\,F$ with a known number of groups, can be solved by reduction to the linear assignment problem,

provided that the number of jobs in each group is known in advance. Typically, such a reduction is possible if the value of an objective function can written in the form

$$F(k) = \sum_{x=1}^{k} \sum_{r=1}^{n^{[x]}} W_j^{[x]}(r) p_{\pi^{[x]}(r)} + \Gamma(k), \tag{4.4}$$

where $\Gamma(k)$ is a sequence-independent constant that depends on k and $W_j^{[x]}(r)$ is a *group-dependent positional weight* associated with a job $j = \pi^{[x]}(r)$, sequenced in a position r, $1 \leq r \leq n^{[x]}$, of a group x, $1 \leq x \leq k$.

The function F of the form (4.4) can be minimised by reducing the problem to an $n \times n$ LAP of the form (2.2). Each of the n rows of the associated cost matrix, corresponds to a job $j \in N$, and each of the n columns corresponds to a position in the schedule. For our purposes, it is convenient to number the columns by a string of the form (x, r), where x refers to a group, $1 \leq x \leq k$, and r, $1 \leq r \leq n^{[x]}$, indicates a position within the group. Thus, the first $n^{[1]}$ columns $(1,1), (1,2), \cdots, (1, n^{[1]})$ of the matrix are associated with the positions in group 1, the next $n^{[2]}$ columns $(2,1), (2,2), \cdots, (2, n^{[2]})$ are associated with the positions in group 2, and so on. It follows that the cost function c_{ji} of the LAP can be written as $c_{j,(x,r)}$, with its value defined by

$$c_{j,(x,r)} = p_j W_j^{[x]}(r), \ 1 \leq r \leq n^{[x]}, \ 1 \leq x \leq k. \tag{4.5}$$

If the number of jobs in each group $n^{[x]}$, $1 \leq x \leq k$, is known so that $\sum_{x=1}^{k} n^{[x]} = n$, an optimal solution to the resulting LAP can be obtained in $O(n^3)$ by the famous Hungarian algorithm. An optimal solution to the LAP assigns each job $j \in N$ to exactly one position from the available positions (x, r), $1 \leq r \leq n^{[x]}$, $1 \leq x \leq k$, so that the function (4.4) is minimised.

If however, it is observed that the value of an objective function can written in the form

$$F(k) = \sum_{x=1}^{k} \sum_{r=1}^{n^{[x]}} W^{[x]}(r) p_{\pi^{[x]}(r)} + \Gamma(k), \tag{4.6}$$

so that the resulting cost function of the LAP is given by $c_{j,(x,r)} = p_j W^{[x]}(r)$, $1 \leq r \leq n^{[x]}$, $1 \leq x \leq k$, then such an LAP is said to have a product matrix and can be solved by Algorithm Match; see Section 2.2.3. Below we provide a formal description of an algorithm based on Algorithm Match, that minimises the objective function $F(k)$ of the form (4.6), if the number of jobs in each group $n^{[x]}$, $1 \leq x \leq k$, is known in advance.

Algorithm Match2

INPUT: An instance of problem $1\,|\varphi, RMP\,(k-1)|\,F$ whose objective function can written in the form (4.6)

OUTPUT: An optimal schedule $S^*\,(k)$

Step 1. If required, renumber the jobs in LPT order, i.e., in non-increasing order of the values p_j.

Step 2. Compute the constant term $\Gamma\,(k)$ and the positional weights $W^{[x]}(r)$, $1 \leq r \leq n^{[x]}$, $1 \leq x \leq k$, for the given problem.

Step 3. Sort the computed positional weights $W^{[x]}(r)$, $1 \leq r \leq n^{[x]}$, $1 \leq x \leq k$, in non-decreasing order of their values, and store them in a list $L := (\gamma_1, \gamma_2, \ldots, \gamma_n)$, where γ_j, $1 \leq j \leq n$, denotes the j-th largest element in the list L.

Step 4. Match job j to the j-th element in list L, so that the optimal value of the function $F(k)$ is given by $\sum_{j=1}^{n} \gamma_j p_j + \Gamma(k)$.

As follows from Lemma 2.1, Algorithm Match2 will solve the problem of minimising the function $F(k)$ of the form (4.6) by assigning jobs with small processing times to positions with large positional weights. If the LPT order of the jobs is known in advance, the running time of Algorithm Match2 is equal to the time $T\,(W)$ it takes to compute the positional weights $W^{[x]}(r)$, $1 \leq r \leq n^{[x]}$, $1 \leq x \leq k$, and the time required to create the sorted list L.

Notice that both the solution approaches presented in this section, assume that the number of jobs, $n^{[x]}$, in each group x, $1 \leq x \leq k$, is known in advance. Thus, to provide a complete solution to problem $1\,|\varphi, RMP\,(k-1)|\,F$, we must find the optimal number of jobs in each group. Additionally, to obtain a solution to the original problem $1\,|\varphi, RMP|\,F$, we need to find the optimal choices for Decisions 1-3. An obvious way to find these unknown quantities is to enumerate all possible options associated with them and choose the best instance. Alternatively, we can design specific algorithms that find these values in a more efficient way.

Notice however, that there are several versions of problem $1\,|\varphi, RMP|\,F$, for which no efficient algorithms can be developed to determine the optimal number of jobs in each group. For such cases, the only available tool is to perform a full enumeration of all possible options. Some of the cases for which this restriction holds, are:

- Problem of minimising the total flow time, see, e.g., problems reviewed in Section 3.3.

- Models in which the processing time of the current job depends on the number of jobs processed in the previous groups, e.g., problem considered in Example 4.1.
- Models in which a non-monotone effect is considered.

We consider all of these problems in Chapter 9.

The problems, which do allow us to design an efficient algorithm for determining the optimal number of jobs in each group, are studied in Chapters 6, 7, and 8.

Moreover, there are some versions of problem $1\,|\varphi, RMP\,(k-1)|\,F$, which do not reduce to minimising a function of the form (4.4) or (4.6) at all, see, e.g., problems with cumulative deterioration effects and rate-modifying activities in Chapter 10. In such cases, we develop a solution approach from scratch.

CHAPTER 5

Convex Sequences with Sums of Ceiling Functions

In this chapter, we establish certain properties that are used on several occasions in this thesis. The main results that we present here primarily revolve around the convex and V-shaped finite sequences and the inequalities that govern them. We prove an inequality that involves an arbitrary non-decreasing function that depends on ceiling functions, thereby establishing the convexity and V-shapeness of the corresponding sequence. This sequence often appears in scheduling problems, especially when a given set of jobs are to be divided into a known number of groups. The V-shapeness of this sequence enables us to speed up the running times of several problems that we consider in this study, see, e.g., Chapters 6, 8 and 11.

The most important results of this chapter are presented in our recent paper Rustogi and Strusevich (2011), published in the Journal of Integer Sequences. In addition to the main results, the paper also provides several examples and applications of the established properties. Other results presented in this chapter can be found in the papers Rustogi and Strusevich (2012a); Rustogi and Strusevich (2013a).

5.1 Brief Overview of Convex and V-shaped Sequences

A sequence $A(k), 1 \leq k \leq n$, is called *convex* if

$$A(k) \leq \frac{1}{2}\left(A(k-1) + A(k+1)\right), \ 2 \leq k \leq n-1, \tag{5.1}$$

i.e., any element is no larger than the arithmetic mean of its neighbours. For a convex sequence, the rate $\nabla(k) = A(k+1) - A(k)$ does not decrease as k grows. A concept of a convex sequence is closely related to the notion of a log-convex sequence, for which

$$A(k) \leq \sqrt{A(k-1)A(k+1)},\ 2 \leq k \leq n-1,$$

i.e., any element is no larger than the geometric mean of its neighbours. Convex sequences play an important role in the derivation of various inequalities. Wu and Debnath (2007) give a necessary and sufficient condition for a sequence to be convex in terms of majorisation, and this gives access to a powerful tool-kit that is systematically exposed by Marshall and Olkin (1979). Various applications of convex sequences to the problems of combinatorics, algebra and calculus are studied by Mercer (2005), Toader (1996) and Wu and Debnath (2007). Any log-convex sequence is also convex due to the inequality between the arithmetic and geometric means. An additional important link between convex and log-convex sequences is pointed out by Došlić (2009). In Operational Research, an application of convex sequences to a special form of the assignment problem and their relations to the famous Monge property is discussed in Chapter 5.2 of the recent monograph by Burkard, Dell'Amico and Martello (2009).

A sequence $C(k)$ is called *V-shaped* if there exists a k_0, $1 \leq k_0 \leq n$, such that

$$C(1) \geq \cdots \geq C(k_0 - 1) \geq C(k_0) \leq C(k_0 + 1) \leq \cdots \leq C(n).$$

If a finite sequence that contains n terms is V-shaped then its minimum value and the position k_0 of that minimum in the sequence can be found by binary search in at most $\lceil \log_2 n \rceil$ comparisons.

The V-shaped sequences often arise in optimisation over a set of permutations, in particular in machine scheduling. For a number of scheduling problems it is possible to establish that an optimal sequence of jobs possesses the V-shaped property, e.g., the elements of the input sequence of the processing times can be interchanged so that the resulting sequence is V-shaped. This property has been explored in numerous papers. Here, we refer to only five, mostly with the words "V-shaped" in the title; see Alidaee and Rosa (1995), Alturki, Mittenthal and Raghavachari (1996), Federgruen and Mosheiov (1997), Mittenhal, Raghavachari and Rana (1995) and Mosheiov (1991).

There is also a somewhat dual concept of the Λ-shaped sequence, also known as unimodal sequence or pyramidal sequence, in which the elements are placed in non-decreasing order and then in non-increasing order. These sequences are the main object of study in identifying efficiently solvable combinatorial optimisation problems,

in particular the traveling salesman problem. Here we only refer to two most recent surveys by Burkard et al. (1998) and Kabadi (2007).

Despite the fact that the two types of sequences, convex and V-shaped, are well-known in the corresponding area of study, to the best of our knowledge, so far there has been no attempt to link these two concepts. In the following section we give an elementary proof that a convex sequence is in fact V-shaped.

5.2 A Convex Sequence is V-Shaped

The statement below links the convex and V-shaped sequences.

Lemma 5.1. *A convex sequence $C(k), 1 \leq k \leq n$, is V-shaped.*

Proof: Suppose that a convex sequence $C(k)$, $1 \leq k \leq n$, is not V-shaped, i.e., there exists a k_1, $1 < k_1 < n$, such that

$$C(k_1 - 1) \leq C(k_1) > C(k_1 + 1).$$

Combining the inequalities

$$\begin{aligned} C(k_1 - 1) &\leq C(k_1), \\ C(k_1) &> C(k_1 + 1), \end{aligned}$$

we obtain

$$C(k_1) > \frac{1}{2}\left(C(k_1 - 1) + C(k_1 + 1)\right).$$

The latter inequality contradicts (5.1). □

The statement opposite to Lemma 5.1 is not true: indeed, any monotone sequence is V-shaped, but need not be convex.

It can be immediately verified that the sum of two convex sequences is convex, and therefore is V-shaped. On the other hand, the sum of two V-shaped sequences is not necessarily V-shaped, which, e.g., is demonstrated by the following counterexample. For an integer n, $0 \leq n \leq 9$, both sequences $\sqrt{|n-1|}$ and $\sqrt{|n-9|}$ are V-shaped, but their sum is not, since the sum has two equal maxima at $n = 0$ and $n = 5$ as well as two equal minima at $n = 1$ and $n = 9$.

5.3 Convexity of a Sequence Involving Sums of Functions of Ceilings

In this chapter, we study a specially structured sequence of the form

$$P(k) = \sum_{j=1}^{n} p_j g\left(\left\lceil \frac{j}{k} \right\rceil\right),\ 1 \le k \le n, \tag{5.2}$$

where p_j is the normal processing time of a job $j \in \{1, 2, \ldots, n\}$ and g is an arbitrary non-negative non-decreasing function. Recall that $\lceil x \rceil$ is the ceiling function and is equal to the smallest integer that is not less than x.

The sequence (5.2), or some version of it, is commonly found in the scheduling literature, especially in problems in which a given set of jobs needs to be divided into a known number of groups. For example, recall from Section 2.2.5, that the optimal value of the objective function for problem $Pm \,||\, \sum C_j$, is given by a closed form formula (2.15), which is equivalent to (5.2) for $g \equiv 1$. In Chapters 6 and 8, we encounter such sequences while studying the problem of scheduling jobs with rate-modifying activities.

In this section, we prove that the sequence (5.2) is convex, provided that the values p_j, $j \in \{1, 2, \ldots, n\}$ are numbered in the LPT order (2.7). We start our consideration with a simpler sequence

$$G(k) = \sum_{j=1}^{n} g\left(\left\lceil \frac{j}{k} \right\rceil\right),\ 1 \le k \le n. \tag{5.3}$$

Below we give an elementary proof of the convexity of the sequence (5.3). It should be noticed that although the ceiling function and its counterpart, the floor function $\lfloor x \rfloor = \max\{n \in \mathbb{Z} | n \le x\}$, arise and find applications in many areas, publications that study the relations that involve these functions are quite scarce; we mention only Chapter 3 of Graham, Knuth and Patashnik (1989) and Nyblom (2002).

Theorem 5.1. *The sequence $G(k)$, $1 \le k \le n$, of the form (5.3) is convex.*

Proof: In accordance with (5.1), we need to prove that

$$2G(k) - G(k-1) - G(k+1) \le 0,\ 2 \le k \le n-1, \tag{5.4}$$

Given a value of k, $2 \le k \le n-1$, for a j, $1 \le j \le n$, we can express j as $ak + b$,

where a is in integer in $\left\{0, 1, \ldots, \left\lfloor \frac{j}{k} \right\rfloor\right\}$ and $b \leq k$. We can write

$$\begin{aligned} G(k) &= \sum_{a=0}^{\left\lfloor \frac{n}{k} \right\rfloor - 1} \sum_{b=1}^{k} g\left(\left\lceil \frac{ak+b}{k} \right\rceil\right) + \sum_{b=1}^{n-k\left\lfloor \frac{n}{k} \right\rfloor} g\left(\left\lceil \frac{\left\lfloor \frac{n}{k} \right\rfloor k + b}{k} \right\rceil\right) \\ &= \sum_{a=0}^{\left\lfloor \frac{n}{k} \right\rfloor - 1} \sum_{b=1}^{k} g\left(a + \left\lceil \frac{b}{k} \right\rceil\right) + \sum_{b=1}^{n-k\left\lfloor \frac{n}{k} \right\rfloor} g\left(\left\lfloor \frac{n}{k} \right\rfloor + \left\lceil \frac{b}{k} \right\rceil\right). \end{aligned}$$

Since $b \leq k$, it follows that $\left\lceil \frac{b}{k} \right\rceil = 1$, so that

$$\begin{aligned} G(k) &= \sum_{a=0}^{\left\lfloor \frac{n}{k} \right\rfloor - 1} \sum_{b=1}^{k} g\left(a+1\right) + \sum_{b=1}^{n-k\left\lfloor \frac{n}{k} \right\rfloor} g\left(\left\lfloor \frac{n}{k} \right\rfloor + 1\right) \\ &= k \sum_{r=1}^{\left\lfloor \frac{n}{k} \right\rfloor} g\left(r\right) + \left(n - k\left\lfloor \frac{n}{k} \right\rfloor\right) g\left(\left\lfloor \frac{n}{k} \right\rfloor + 1\right), \end{aligned}$$

where $r = a + 1$ is a new summation index. Similarly, we deduce

$$\begin{aligned} G(k+1) &= (k+1) \sum_{r=1}^{\left\lfloor \frac{n}{k+1} \right\rfloor} g\left(r\right) + \left(n - (k+1)\left\lfloor \frac{n}{k+1} \right\rfloor\right) g\left(\left\lfloor \frac{n}{k+1} \right\rfloor + 1\right); \\ G(k-1) &= (k-1) \sum_{r=1}^{\left\lfloor \frac{n}{k-1} \right\rfloor} g\left(r\right) + \left(n - (k-1)\left\lfloor \frac{n}{k-1} \right\rfloor\right) g\left(\left\lfloor \frac{n}{k-1} \right\rfloor + 1\right). \end{aligned}$$

To prove (5.4), we rewrite the difference $2G(k) - G(k+1) - G(k-1)$ as

$$\begin{aligned} &\left(k \sum_{r=1}^{\left\lfloor \frac{n}{k} \right\rfloor} g\left(r\right) - (k+1) \sum_{r=1}^{\left\lfloor \frac{n}{k+1} \right\rfloor} g\left(r\right)\right) + \left(k \sum_{r=1}^{\left\lfloor \frac{n}{k} \right\rfloor} g\left(r\right) - (k-1) \sum_{r=1}^{\left\lfloor \frac{n}{k-1} \right\rfloor} g\left(r\right)\right) \\ &+2\left(n - k\left\lfloor \frac{n}{k} \right\rfloor\right) g\left(\left\lfloor \frac{n}{k} \right\rfloor + 1\right) - \left(n - (k+1)\left\lfloor \frac{n}{k+1} \right\rfloor\right) g\left(\left\lfloor \frac{n}{k+1} \right\rfloor + 1\right) \\ &- \left(n - (k-1)\left\lfloor \frac{n}{k-1} \right\rfloor\right) g\left(\left\lfloor \frac{n}{k-1} \right\rfloor + 1\right) \end{aligned}$$

and show that it is non-positive.

Notice that the expression

$$k \sum_{r=1}^{\left\lfloor \frac{n}{k} \right\rfloor} g\left(r\right) - (k+1) \sum_{r=1}^{\left\lfloor \frac{n}{k+1} \right\rfloor} g\left(r\right)$$

can be simplified by cancelling the values of the function g computed for the same values of r. Since $\left\lfloor \frac{n}{k} \right\rfloor \geq \left\lfloor \frac{n}{k+1} \right\rfloor$, we obtain

$$k \sum_{r=1}^{\left\lfloor \frac{n}{k} \right\rfloor} g(r) - (k+1) \sum_{r=1}^{\left\lfloor \frac{n}{k+1} \right\rfloor} g(r) = k \sum_{r=\left\lfloor \frac{n}{k+1} \right\rfloor + 1}^{\left\lfloor \frac{n}{k} \right\rfloor} g(r) - \sum_{r=1}^{\left\lfloor \frac{n}{k+1} \right\rfloor} g(r).$$

Similarly, since $\left\lfloor \frac{n}{k-1} \right\rfloor \geq \left\lfloor \frac{n}{k} \right\rfloor$, we obtain

$$k \sum_{r=1}^{\left\lfloor \frac{n}{k} \right\rfloor} g(r) - (k-1) \sum_{r=1}^{\left\lfloor \frac{n}{k-1} \right\rfloor} g(r) = \sum_{r=1}^{\left\lfloor \frac{n}{k-1} \right\rfloor} g(r) - k \sum_{r=\left\lfloor \frac{n}{k} \right\rfloor + 1}^{\left\lfloor \frac{n}{k-1} \right\rfloor} g(r).$$

Combining the two above equalities, we obtain

$$\begin{aligned} & \left(k \sum_{r=1}^{\left\lfloor \frac{n}{k} \right\rfloor} g(r) - (k+1) \sum_{r=1}^{\left\lfloor \frac{n}{k+1} \right\rfloor} g(r) \right) + \left(k \sum_{r=1}^{\left\lfloor \frac{n}{k} \right\rfloor} g(r) - (k-1) \sum_{r=1}^{\left\lfloor \frac{n}{k-1} \right\rfloor} g(r) \right) \\ = \; & k \left(\sum_{r=\left\lfloor \frac{n}{k+1} \right\rfloor + 1}^{\left\lfloor \frac{n}{k} \right\rfloor} g(r) - \sum_{r=\left\lfloor \frac{n}{k} \right\rfloor + 1}^{\left\lfloor \frac{n}{k-1} \right\rfloor} g(r) \right) + \sum_{r=\left\lfloor \frac{n}{k+1} \right\rfloor + 1}^{\left\lfloor \frac{n}{k-1} \right\rfloor} g(r), \end{aligned}$$

which reduces to

$$\begin{aligned} & \left(k \sum_{r=1}^{\left\lfloor \frac{n}{k} \right\rfloor} g(r) - (k+1) \sum_{r=1}^{\left\lfloor \frac{n}{k+1} \right\rfloor} g(r) \right) + \left(k \sum_{r=1}^{\left\lfloor \frac{n}{k} \right\rfloor} g(r) - (k-1) \sum_{r=1}^{\left\lfloor \frac{n}{k-1} \right\rfloor} g(r) \right) \\ = \; & (k+1) \sum_{r=\left\lfloor \frac{n}{k+1} \right\rfloor + 1}^{\left\lfloor \frac{n}{k} \right\rfloor} g(r) - (k-1) \sum_{r=\left\lfloor \frac{n}{k} \right\rfloor + 1}^{\left\lfloor \frac{n}{k-1} \right\rfloor} g(r). \end{aligned}$$

Coming back to the initial difference in (5.4), we have that

$$\begin{aligned} 2G(k) - G(k+1) - G(k-1) \; = \; & (k+1) \sum_{r=\left\lfloor \frac{n}{k+1} \right\rfloor + 1}^{\left\lfloor \frac{n}{k} \right\rfloor} g(r) - (k-1) \sum_{r=\left\lfloor \frac{n}{k} \right\rfloor + 1}^{\left\lfloor \frac{n}{k-1} \right\rfloor} g(r) \\ & + 2 \left(n - k \left\lfloor \frac{n}{k} \right\rfloor \right) g\left(\left\lfloor \frac{n}{k} \right\rfloor + 1 \right) \\ & - \left(n - (k+1) \left\lfloor \frac{n}{k+1} \right\rfloor \right) g\left(\left\lfloor \frac{n}{k+1} \right\rfloor + 1 \right) \end{aligned}$$

Recombine

$$(k+1)\sum_{r=\left\lfloor \frac{n}{k+1}\right\rfloor+2}^{\left\lfloor \frac{n}{k}\right\rfloor} g(r)+\left((k+1)-\left(n-(k+1)\left\lfloor\frac{n}{k+1}\right\rfloor\right)\right)g\left(\left\lfloor\frac{n}{k+1}\right\rfloor+1\right)$$
$$-(k-1)\sum_{r=\left\lfloor \frac{n}{k}\right\rfloor+2}^{\left\lfloor \frac{n}{k-1}\right\rfloor} g(r)+\left(2n-2k\left\lfloor\frac{n}{k}\right\rfloor-(k-1)\right)g\left(\left\lfloor\frac{n}{k}\right\rfloor+1\right)$$
$$-\left(n-(k-1)\left\lfloor\frac{n}{k-1}\right\rfloor\right)g\left(\left\lfloor\frac{n}{k-1}\right\rfloor+1\right).$$

To prove (5.4), we collect together the terms that make a positive contribution X and a negative contribution Y so that $2G(k)-G(k+1)-G(k-1)=X-Y$, and show that $X\le Y$. It follows that

$$\begin{aligned} X &= \left((k+1)\left(\left\lfloor\frac{n}{k+1}\right\rfloor+1\right)-n\right)g\left(\left\lfloor\frac{n}{k+1}\right\rfloor+1\right)+(k+1)\sum_{r=\left\lfloor \frac{n}{k+1}\right\rfloor+2}^{\left\lfloor \frac{n}{k}\right\rfloor} g(r)+ \\ &\quad +(2n+1)\,g\left(\left\lfloor\frac{n}{k}\right\rfloor+1\right); \\ Y &= \left(2k\left\lfloor\frac{n}{k}\right\rfloor+k\right)g\left(\left\lfloor\frac{n}{k}\right\rfloor+1\right)+(k-1)\sum_{r=\left\lfloor \frac{n}{k}\right\rfloor+2}^{\left\lfloor \frac{n}{k-1}\right\rfloor} g(r)+ \\ &\quad +\left(n-(k-1)\left\lfloor\frac{n}{k-1}\right\rfloor\right)g\left(\left\lfloor\frac{n}{k-1}\right\rfloor+1\right). \end{aligned}$$

In the expression for X, the arguments of the function g are from $\left\lfloor\frac{n}{k+1}\right\rfloor+1$ to $\left\lfloor\frac{n}{k}\right\rfloor+1$, while in the expression for Y, the arguments of the function g are from $\left\lfloor\frac{n}{k}\right\rfloor+1$ to $\left\lfloor\frac{n}{k-1}\right\rfloor+1$, so that we can write

$$X=\sum_{r=\left\lfloor \frac{n}{k+1}\right\rfloor+1}^{\left\lfloor \frac{n}{k}\right\rfloor+1} x_r g(r),\ Y=\sum_{r=\left\lfloor \frac{n}{k}\right\rfloor+1}^{\left\lfloor \frac{n}{k-1}\right\rfloor+1} y_r g(r),$$

where all coefficients x_r and y_r are non-negative.

Computing

$$\begin{aligned}
\sum_{r=\left\lfloor \frac{n}{k+1} \right\rfloor +1}^{\left\lfloor \frac{n}{k} \right\rfloor +1} x_r &= (k+1)\left(\left\lfloor \frac{n}{k+1} \right\rfloor +1 \right) - n + (k+1)\left(\left\lfloor \frac{n}{k} \right\rfloor - \left\lfloor \frac{n}{k+1} \right\rfloor - 1 \right) + (2n+1), \\
\sum_{r=\left\lfloor \frac{n}{k} \right\rfloor +1}^{\left\lfloor \frac{n}{k-1} \right\rfloor +1} y_r &= \left(2k\left\lfloor \frac{n}{k} \right\rfloor + k \right) + (k-1)\left(\left\lfloor \frac{n}{k-1} \right\rfloor - \left\lfloor \frac{n}{k} \right\rfloor - 1 \right) + \left(n - (k-1)\left\lfloor \frac{n}{k-1} \right\rfloor \right),
\end{aligned}$$

we deduce that both sums above are equal to $(k+1)\left\lfloor \frac{n}{k} \right\rfloor + n + 1$. Thus, each X and Y can be seen as the sum of $(k+1)\left\lfloor \frac{n}{k} \right\rfloor + n + 1$ values of the function g; however, for any i, $1 \leq i \leq (k+1)\left\lfloor \frac{n}{k} \right\rfloor + n + 1$, the i-th smallest value of $g(r)$ involved in the expression for X is no larger than the i-th smallest value of $g(r)$ involved in the expression for Y. This proves that $X \leq Y$, i.e., $2G(k) - G(k+1) - G(k-1) \leq 0$, so that the sequence $G(k)$, $1 \leq k \leq n$, is convex. □

Theorem 5.2. *The sequence $P(k)$, $1 \leq k \leq n$, of the form (5.2) is convex, if $p_1 \geq p_2 \geq \cdots \geq p_n$.*

Proof: For a given $j \in \{1, 2, ..., n\}$, define

$$A_j(k) = \left[2g\left(\left\lceil \frac{j}{k} \right\rceil \right) - g\left(\left\lceil \frac{j}{k+1} \right\rceil \right) - g\left(\left\lceil \frac{j}{k-1} \right\rceil \right) \right], \ 2 \leq k \leq n-1.$$

By Theorem 5.1, due to the convexity of the sequence $B(k)$, $1 \leq k \leq n$, we deduce that

$$\sum_{i=1}^{q} A_i(k) \leq 0 \tag{5.5}$$

for each k, $2 \leq k \leq n-1$, and all q, $1 \leq q \leq n$.

In order to prove the theorem, we need to demonstrate that the inequality

$$\sum_{j=1}^{n} p_j A_j(k) \leq 0 \tag{5.6}$$

holds for each k, $2 \leq k \leq n-1$.

Fix a k, $2 \le k \le n-1$, and transform

$$\begin{aligned}
\sum_{j=1}^{n} p_j A_j(k) &= p_n \left(\sum_{i=1}^{n} A_i(k) - \sum_{i=1}^{n-1} A_i(k) \right) + p_{n-1} \left(\sum_{i=1}^{n-1} A_i(k) - \sum_{i=1}^{n-2} A_i(k) \right) + \cdots \\
&\quad + p_1 A_1(k) \\
&= p_n \sum_{i=1}^{n} A_i(k) + (p_{n-1} - p_n) \sum_{i=1}^{n-1} A_i(k) + (p_{n-2} - p_{n-1}) \sum_{i=1}^{n-2} A_i(k) + \cdots \\
&\quad + p_1 A_1(k) \\
&= \sum_{j=2}^{n} \left[(p_{j-1} - p_j) \sum_{i=1}^{j-1} A_i(k) \right] + p_n \sum_{i=1}^{n} A_i(k).
\end{aligned}$$

The last right-hand expression is non-positive due to (5.5) and the LPT numbering of p_j (2.7), so that the desired inequality (5.6) holds and the sequence $P(k)$, $1 \le k \le n$, is convex. □

The convexity of the sequence $P(k)$, $1 \le k \le n$, as established in Theorem 5.2, allows us to use an efficient binary search algorithm to solve several problems considered in this thesis, see, e.g., Sections 6.7, 8.5 and 11.5. For other applications of the results of this chapter, refer to our paper Rustogi and Strusevich (2011).

Part III

Single Machine Scheduling

CHAPTER 6

Job-Independent Positional Effects and Rate-Modifying Activities

In this chapter, we discuss single machine scheduling problems with job-independent positional effects. We start with making some general improvements of the existing results in scheduling with positional effects without any rate-modifying activities. Then, we study models in which positional effects are combined with rate modifying activities. Our main focus in this chapter is to explore general models with job-independent positional deterioration and maintenance activities. We provide a variety of solution approaches that efficiently solve different versions of such problems.

The most important results of this chapter are published in our recent papers Rustogi and Strusevich (2012a,b). In these papers, we only provide a simplified version of the problem, so that the main ideology behind the developed algorithms can be clearly understood. In this chapter, however, we provide a full account of the entire range of problems that can be solved using the developed solution approaches.

6.1 Brief Overview of Positional Effects

As described in Section 3.2.1, under a job-independent positional effect the actual processing time of job j scheduled in position r of a schedule is given by

$$p_j(r) = p_j g(r),\ 1 \leq r \leq n,$$

where $g(r)$ is a job-independent positional factor. If the values $g(r)$, $1 \leq r \leq n$, form a non-decreasing sequence (3.2), we deal with a positional deterioration effect; if the sequence is non-increasing (3.3), a learning effect is observed. A positional effect can also be job-dependent, in which the actual processing time of job j scheduled in

position r of a schedule is given by

$$p_j(r) = p_j g_j(r),\ 1 \leq r \leq n.$$

We shall discuss problems of the latter category in Chapter 7.

To demonstrate that a deterioration effect can be positional, consider the following example. Imagine that in a manufacturing shop there are several parts that need a hole of the same diameter to be punched through by a pneumatic punching unit. Ideally, the time that is required for such an operation depends on the thickness of the metal to be punched through; and this will determine the normal processing times for all parts. In reality however, there occurs an unavoidable gas leakage after each punch, due to which the punching unit loses pressure, so that the later a part is subject to punching the longer it takes to perform it, as compared to the duration under perfect conditions. Clearly, a positional deterioration effect is observed.

A learning effect can occur when what we call machines are in fact human operators that gain experience and improve their performance rate with each processed job. Being part of the academia, it is easy to notice that positional learning takes place when a teacher marks a number of coursework scripts based on the same question paper. It takes a reasonably long time to mark the first two or three scripts, then the teacher realises the key factors to be checked, typical strong or weak points to be looked for, and the marking process goes faster and faster with each marked script.

In the literature on scheduling with positional effects, learning and deterioration are often studied separately, although similar methods can be employed in either case and some algorithmic ideas are either directly transferable from one effect to the other or at least can be adapted. In this study, we argue that in most cases there is no need to separate the studies on learning from those on deterioration. In fact, we demonstrate that we may look at an arbitrary positional effect, given by a non-monotone sequence of the positional factors.

These arbitrary non-monotone positional effects can be found in practice as well. Extending the coursework marking example above, after marking a certain number of scripts, the teacher might get tired or bored, her attention becomes less focused and each new script may even take longer to mark than the one before. In this case we can say that a deterioration and learning effect is taking place simultaneously. If the deterioration in the conditions of the teacher is modelled by the positional factors given by an array $g_d(1) \leq g_d(2) \leq \cdots \leq g_d(n)$ and the positional learning effect is given by an array $g_l(1) \geq g_l(2) \geq \cdots \geq g_l(n)$, then the time it takes for the teacher to mark the

r-th coursework can be given by $p_j(r) = p_j g(r)$, where $g(r) := g_d(r)\, g_l(r)$, $1 \leq r \leq n$. Clearly, the positional factors $g(r)$, $1 \leq r \leq n$, are non-monotone. This is the first study in which such effects have been considered.

In earlier papers, the focus has been on particular functions that define the positional factors $g(r)$, e.g., polynomial of the form (3.9)-(3.11) or exponential (3.10)-(3.12). In this study however, we show that in order to obtain a polynomial-time algorithm for a problem with job-independent positional effects, there is no need to look at a particular function. Instead, a general function $g(r)$ can be used to model the effects and all results can be derived with as much ease.

Further in this chapter, we consider situations in which a positional effect is combined with rate modifying activities. If a machine undergoes a deterioration effect, performing a maintenance activity in the schedule might stop the deterioration process and prevent the processing times of jobs to grow to unacceptable values. For instance, in the manufacturing shop example given above, after a considerable drop of pressure, the punching unit can be subjected to maintenance, so that the cylinder is refilled and the unit is as good as new, or close to that state. On the other hand, in the coursework example, if the teacher feels very tired, she can take a coffee break and refresh herself. This can be seen as an RMP and will certainly improve the productivity of the teacher. Another option she has is to give all the remaining courseworks to her Ph.D. student. This event can also be seen as rate-modifying activity in the entire sequence of marking n courseworks. Notice that while the Ph.D. student is fresh, he does not have the experience of marking the earlier courseworks, and thus, to mark the first few courseworks, he might take even longer than the tired professor. Thus, unlike the previous case, this RMP does not necessarily bring the machine/operator to a better state. This is the first study in which different types of RMPs are simultaneously considered in the schedule. The models that combine a general possibly non-monotone positional effect and rate modifying activities of different kinds are among the most general models addressed in this study.

The rest of this chapter is organised as follows. In Section 6.2 we focus on problems with positional effects and no rate-modifying activities. As a result all jobs are scheduled in a single group and we show that even if an arbitrary, possibly non-monotone positional effect is considered, several objective functions can be solved in $O(n \log n)$ time. Previously many of such problems were known to be solvable in $O(n^3)$ time, even if a specific monotone positional effect was considered. In Section 6.3 we introduce rate-modifying activities and show how we can mathematically model the effect they have on machine conditions. In Section 6.4 we provide preliminary calculations

required to solve the problem of minimising the makespan for a model with positional effects and rate modifying activities. In Sections 6.5, 6.6 and 6.7 we provide different solution approaches for solving different versions of the problem of minimising the makespan. In the last section, we give some concluding remarks and provide a table with all the problems considered and the running times needed to solve them.

6.2 Models without Rate Modifying Activities

In this section, we focus on the simplest type of scheduling problems, in which the jobs are processed on a single machine and are organised as a single group. Unlike most of the previously published papers relevant to the material of this section, here we assume that an arbitrary positional effect is involved, so that the positional factors are given by a general sequence of numbers $g(r)$, $1 \leq r \leq n$, and are not necessarily monotone. In the case of either positional deterioration or learning the corresponding factors may satisfy either (3.2) or (3.3), respectively.

Formally, all the problems considered in the section can be denoted in conventional scheduling nomenclature by $1|p_j g(r)|F$, where the second field explicitly points out that the actual processing time of job j sequenced in the r-th position is equal to $p_j g(r)$. Let us assume that in all versions of problem $1|p_j g(r)|F$, jobs from the set N are processed on a single machine in accordance with some permutation $\pi = (\pi(1), \ldots, \pi(n))$ and a job $j = \pi(r)$ sequenced in position r is associated with a job-independent positional factor $g(r)$.

Below we revisit each of the problems considered in Section 3.2.1 and prove that they can be expressed in the form (4.3), and can be solved optimally by running Algorithm Match1, which was presented in Chapter 4.

Minimising the Makespan

It is easy to verify that for problem $1|p_j g(r)|C_{\max}$, the makespan can be written as (3.8), even if the positional factors $g(r)$, $1 \leq r \leq n$, are not monotone. Notice that (3.8) satisfies (4.3) with $W(r) = g(r),\ 1 \leq r \leq n$, and $\Gamma = 0$, so that an optimal solution can found by Algorithm Match1. Since the positional weights $W(r)$, $1 \leq r \leq n$, are already known, an optimal schedule can be found in $O(n \log n)$ time, due to Lemma 4.1.

Next, consider the problem $1|A_j + b_j r|C_{\max}$ with a linear positional effect (3.7). For this problem, even in a more general situation, e.g., when the actual time of job $j = \pi(r)$ scheduled in position r is defined by $A_{\pi(r)} + b_{\pi(r)} g(r)$, and arbitrary values of

b_j, the makespan can be written as

$$C_{\max}(\pi) = \sum_{r=1}^{n} A_{\pi(r)} + \sum_{r=1}^{n} b_{\pi(r)} g(r),$$

which satisfies (4.3) with $W(r) = g(r)$, $1 \leq r \leq n$, $p_j = b_j$, $j \in N$ and $\Gamma = \sum_{j=1}^{n} A_j$, so that an optimal solution can found by Algorithm Match1 in $O(n \log n)$ time.

Minimising the Total Flow time

It is easy to verify that for problem $1 \,|p_j g(r)| \sum C_j$, the total flow time can be written as (3.14), even if the positional factors $g(r)$, $1 \leq r \leq n$, are not monotone. Notice that (3.14) satisfies (4.3) with $W(r) = (n - r + 1)\, g(r)$, $1 \leq r \leq n$, and $\Gamma = 0$, so that an optimal schedule can be found by Algorithm Match1. It requires $T(W) = O(n)$ time to compute all the positional weights $W(r)$, $1 \leq r \leq n$, thus, an optimal schedule can be found in $O(n \log n)$ time, due to Lemma 4.1.

Recall from Section 3.2.1, that prior to this study no positive results have been known for the deterioration version of problem $1 \,|p_j g(r)| \sum C_j$. Our approach shows that for a deterioration effect the search for an optimal strategy should not be limited to simple priority rules. In fact, Algorithm Match1 not only solves problem $1 \,|p_j g(r)| \sum C_j$ in the case of deterioration, but works for arbitrary, possibly non-monotone positional factors.

If we consider the problem of minimising the total flow time with the effect $p_j(r) = A_j + b_j r$, introduced by Bachman and Janiak (2004), it can be easily verified that the problem does not reduce to minimising (4.3) and it is essential to use a full form LAP; see Yang and Yang (2010b).

Other Objective Functions

In line with Section 3.2.1, in which we review other somewhat exotic objective functions, we now revisit those problems for a general positional effect. Notice that all problems considered in Section 3.2.1 share the following common features:

- the problems have been previously studied for a particular model of positional learning, normally with a polynomial learning effect;
- the problems have been previously formulated as LAPs;

- the authors fail to notice that those assignment problems are of a special structure;
- in the original papers and in the reviews by Bachman and Janiak (2004) and Biskup (2008) each of these problems is mentioned as solvable in $O(n^3)$ time.

Below we show that each of the problems considered in Section 3.2.1 can in fact be reduced to the form (4.3), even if the positional factors are given by a general non-monotone function $g(r)$. As a result, using appropriate values of $W(r)$ and Γ, each of the problems can be solved by Algorithm Match1.

Minimising deviation from a common unrestricted due date: In the case of an arbitrary positional effect, we can denote the problem by $1|p_j g(r), d_j = d|\sum(wC_j + w'E_j + w''T_j)$. Extending the argument by Panwalkar, Smith and Seidmann (1982) and Biskup (1999), we can reduce the problem to minimising the function (4.3) with $\Gamma = 0$ and

$$W(r) = g(r)\min\{(r-1)w' + (n-r+1)w, (n-r+1)(w''+w)\}, 1 \le r \le n,$$

so that an optimal solution can be found in $O(n \log n)$ time. Notice that this is possible because the cost function

$$c_{ij} = p_j i^a \min\{(i-1)w' + (n-i+1)w, (n-i+1)(w''+w)\}, 1 \le i \le n,\ j \in N,$$

derived by the authors for use in the full form LAP is of the form $c_{ij} = \alpha_i \beta_j$, so that the input of the problem is determined by two arrays $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ and $\beta = (\beta_1, \beta_2, \ldots, \beta_n)$. The same logic holds for the other two objective functions.

Common due date assignment: In the case of an arbitrary positional effect, we can denote the problem by $1|p_j g(r), d_j = d|\sum(wd + w'E_j + w''T_j)$. Extending the argument by Panwalkar, Smith and Seidmann (1982) and Mosheiov (2001a), we can reduce the problem to minimising the function (4.3) with $\Gamma = 0$ and

$$W(r) = \begin{cases} g(r)\left(nw + (r-1)w'\right), & 1 \le r \le u \\ g(r)\left(n+1-r\right)w'', & u+1 \le r \le n, \end{cases}$$

where

$$u = \left\lceil \frac{w'' - w}{w' + w''} \right\rceil.$$

Thus, an optimal permutation can be found in $O(n \log n)$ time. Notice that a similar observation is also made by Kuo and Yang (2007) for a polynomial learning

effect.

Minimising the weighted sum of total completion time and variation of completion times: In the case of an arbitrary positional effect, we can denote the problem by $1\,|p_j g(r), d_j = d|\, w\sum C_j + (1-w)\sum\sum |C_i - C_j|$. Extending the argument by Bagchi (1989) and Mosheiov (2001a), we can reduce the problem to minimising the function (4.3) with $\Gamma = 0$ and

$$W(r) = g(r)\left[(2w-1)(n+1) + r(2-3w+n(1-w)) - r^2(1-w)\right], 1 \le r \le n,$$

so that an optimal solution can be found in $O(n \log n)$ time.

The results of this section have been published in our recent paper, see Rustogi and Strusevich (2012b). In the next chapter we discuss how the general positional effects described in this section, can be combined with rate-modifying activities to give rise to enhanced scheduling models.

6.3 Rate Modifying Activities

In this section, we combine rate-modifying activities with positional effects. As outlined in Chapter 4, consider a general situation, in which the decision-maker is presented with a total of $K \ge 0$ possible rate-modifying activities, which can be either distinct or alike. For each RMP, it is exactly known how it affects the processing conditions of the machine, should the decision-maker decide to include it into a schedule.

If $k-1$ RMPs are chosen from the available K options, then the jobs are divided into k, $1 \le k \le K+1$ groups. Depending on which RMPs are chosen and the order in which they are performed, the actual processing time of a job $j \in N$, scheduled in position r of the x-th group can be given by

$$p_j^{[x]}(r) = p_j g^{[x]}(r), \ 1 \le r \le n, \ 1 \le x \le k, \tag{6.1}$$

where $g^{[x]}(r)$ is a *group-dependent positional factor*. The presence of group-dependent positional factors implies that the actual processing time of a job is dependent on the position of the job in a group and also on the group that job is scheduled in.

Recall from Section 3.3.1, that in the earlier problems which studied positional effects with rate-modifying activities, it is assumed that all RMPs are identical and they restore the machine to the same state. As a result, the resulting groups are indistinguishable and each group can be associated with the same set of positional

factors, which are of the form $g(r),\ 1 \leq r \leq n$. We refer to such positional factors as *group-independent* positional factors.

The problem of minimising a certain objective function F, under the general settings defined by (6.1) and (4.1) can be denoted by $1\left|p_j g^{[x]}(r), RMP\right|F$, where the first item in the middle field indicates how the actual processing time of a job j scheduled in position r of the x-th group of a schedule is calculated; later on we will use appropriate simplified notation for various special cases of the model.

If a pure deterioration model is considered, then the positional factors within a group x are in non-decreasing order

$$1 \leq g^{[x]}(1) \leq g^{[x]}(2) \leq \cdots \leq g^{[x]}(n),\ 1 \leq x \leq k, \tag{6.2}$$

whereas if a pure learning model is considered, then the positional factors within a group x are in non-increasing order

$$1 \geq g^{[x]}(1) \geq g^{[x]}(2) \geq \cdots \geq g^{[x]}(n),\ 1 \leq x \leq k. \tag{6.3}$$

In the former case, the RMPs are essentially maintenance periods (MPs), which must be included in the schedule in order to negate the deteriorating machine conditions; see, e.g., Kuo and Yang (2008a) and Yang and Yang (2010b). In the latter case, the RMPs can either be associated with replacing a machine/operator or be associated with an activity which further enhances the learning rate of the machine; see, e.g., Ji and Cheng (2010). If a pure learning model is considered and the RMPs are related to replacing the machine, then the problem is trivial as it is optimal to simply schedule all jobs in one group. On the other hand, if a pure learning model is considered and the RMPs are related to further enhancing the machine conditions, then it is essential to pass on the learning effects of previous groups to the current group. This can be done if the positional factors of the current group are dependent on the number of jobs scheduled in previous groups. This situation is similar to Example 4.1. For such cases, it is not possible to determine the optimal number of jobs in a group without full enumeration. Thus, in this chapter, we skip both scenarios related to learning with rate-modifying activities. We revisit these problems in Chapter 9, where we consider a much more general scenario in which an arbitrary, possibly non-monotone positional effect is considered.

In this chapter, we only concentrate on problems in which the objective function is to minimise the makespan and the positional factors are non-decreasing within a group (6.2), so that a pure deterioration effect is considered. In some places, we may

refer to such positional factors as *deterioration factors*. Recall that in the case of a pure deterioration effect, the RMPs are essentially maintenance periods (MPs). The resulting class of problems can be denoted as $1\left|p_j g^{[x]}(r)\text{-det}, MP\right|C_{\max}$. An optimal solution to problem $1\left|p_j g^{[x]}(r)\text{-det}, MP\right|C_{\max}$ is found by determining the outcomes of Decisions 1-4, as defined in Chapter 4.

6.4 Computing Positional Weights

To solve problem $1\left|p_j g^{[x]}(r)\text{-det}, MP\right|C_{\max}$, we first assume that Decisions 1-3 are taken in advance, so that we know that a total of $k-1$ MPs have been included in the schedule. As a result the jobs are split into k, $1 \leq k \leq K+1$, groups. Renumber the indices of duration parameters of the MPs, in order of their occurrence in the schedule, so that the duration of the MP scheduled after the x-th group is given by

$$T_x = \alpha^{[x]} F_x + \beta^{[x]},\ 1 \leq x \leq k-1, \tag{6.4}$$

where F_x is the total processing time of all jobs scheduled in the x-th group. Denote the resulting problem as $1\left|p_j g^{[x]}(r)\text{-det}, MP(k-1)\right|C_{\max}$.

To solve problem $1\left|p_j g^{[x]}(r)\text{-det}, MP(k-1)\right|C_{\max}$ consider a schedule $S(k)$ with a permutation of jobs $\pi = \left(\pi^{[1]}, \pi^{[2]}, \ldots, \pi^{[k]}\right)$. Assume that each group contains a total of $n^{[x]}$ jobs, so that $\pi^{[x]} = \left(\pi^{[x]}(1), \pi^{[x]}(2), \ldots, \pi^{[x]}\left(n^{[x]}\right)\right)$, $1 \leq x \leq k$, where $\sum_{x=1}^{k} n^{[x]} = n$. The actual processing time of a job $j = \pi^{[x]}(r)$, scheduled in position r, $1 \leq r \leq n^{[x]}$, of the x-th group, $1 \leq x \leq k$, is given by (6.1). It follows that the total processing time of the jobs assigned to group x can be given by

$$F_x = \sum_{r=1}^{n^{[x]}} g^{[x]}(r)\, p_{\pi^{[x]}(r)},\ 1 \leq x \leq k. \tag{6.5}$$

Thus, the completion time $C_{\pi^{[x]}(r)}$ of a job $j = \pi^{[x]}(r)$, scheduled in position r, $1 \leq r \leq n^{[x]}$, of the x-th group, $1 \leq x \leq k$, can be written as

$$C_{\pi^{[x]}(r)} = F_1 + T_1 + F_2 + T_2 + \cdots + F_{x-1} + T_{x-1} + \sum_{u=1}^{r} g^{[x]}(u)\, p_{\pi^{[x]}(u)},$$

where $T_1, T_2, \ldots, T_{x-1}$ are the durations of the first $x-1$ MPs and are given by (6.4).

It follows that

$$C_{\pi^{[x]}(r)} = \sum_{v=1}^{x-1}(1+\alpha^{[v]})F_v + \sum_{u=1}^{r} g^{[x]}(u)\, p_{\pi^{[x]}(u)} + \sum_{v=1}^{x-1}\beta^{[v]}. \tag{6.6}$$

The makespan of a schedule $S(k)$ is defined as the completion time of the last job in the sequence and is denoted by $C_{\max}(S(k))$. It follows from (6.6) that

$$C_{\max}(S(k)) = \sum_{x=1}^{k-1}(1+\alpha^{[x]})F_x + F_k + \sum_{x=1}^{k-1}\beta^{[x]}.$$

Substituting the value of F_x from (6.5) in the above equation we get

$$C_{\max}(S(k)) = \sum_{x=1}^{k-1}\sum_{r=1}^{n^{[x]}}(1+\alpha^{[x]})g^{[x]}(r)\, p_{\pi^{[x]}(r)} + \sum_{r=1}^{n^{[k]}} g^{[k]}(r)\, p_{\pi^{[k]}(r)} + \sum_{x=1}^{k-1}\beta^{[x]}. \tag{6.7}$$

The above objective function can be written as generic function given by (4.6) with the positional weights

$$W^{[x]}(r) = \begin{cases} (1+\alpha^{[x]})g^{[x]}(r), & 1 \le r \le n^{[x]},\ 1 \le x \le k-1, \\ g^{[x]}(r) & 1 \le r \le n^{[x]},\ x = k, \end{cases} \tag{6.8}$$

and the constant term

$$\Gamma(k) = \sum_{x=1}^{k-1}\beta^{[x]}. \tag{6.9}$$

As shown in Section 4.2.2, if the number of jobs in each group, $n^{[x]}$, $1 \le x \le k$, is known in advance, the problem of minimising the generic objective function (4.6), can be solved by running Algorithm Match2. To find an optimal solution to problem $1\left|p_j g^{[x]}(r)\text{-det}, MP(k-1)\right| C_{\max}$, however, we must find a way of determining the optimal number of jobs $n^{[x]}$, that are scheduled in each group x, $1 \le x \le k$, before we can apply Algorithm Match2. To obtain a solution to the original problem $1\left|p_j g^{[x]}(r)\text{-det}, MP\right| C_{\max}$, we solve all possible instances of problem $1\left|p_j g^{[x]}(r)\text{-det}, MP(k-1)\right| C_{\max}$ by modifying the outcomes of Decisions 1-3 and choose the instance with the smallest value of the objective function as our optimal solution.

For different versions of problem $1\left|p_j g^{[x]}(r)\text{-det}, MP\right| C_{\max}$ the resulting subproblem $1\left|p_j g^{[x]}(r)\text{-det}, MP(k-1)\right| C_{\max}$ might generate positional weights that are different from those found in (6.8). Recall that in the calculation of (6.8) we consider

a very general scheduling model, in which the actual processing times of jobs are given by (6.1) and the MP durations are given by (4.1). In total, we consider $2^3 = 8$ versions of problem $1\left|p_j g^{[x]}(r)\text{-det}, MP\right| C_{\max}$, differentiating them based on three criteria: (i) deterioration factors are group-dependent or group-independent, (ii) MPs are identical or distinct, and (iii) duration of the MPs are constant or start-time dependent. For each version, the computed positional weights are found by making an appropriate substitution in (6.8).

Depending on the found positional weights, it might be possible to solve problem $1\left|p_j g^{[x]}(r)\text{-det}, MP(k-1)\right| C_{\max}$ by running a revised version of Algorithm Match2, in which the optimal values of $n^{[x]}$, $1 \leq x \leq k$, can be found on the fly. Moreover, for some versions of problem $1\left|p_j g^{[x]}(r)\text{-det}, MP\right| C_{\max}$, it is even possible to make the optimal choices for Decisions 1-3 on the fly, while for others, all possible options for these decisions must be enumerated. Based on such differences in different versions of problem $1\left|p_j g^{[x]}(r)\text{-det}, MP\right| C_{\max}$, we classify them into three separate sections.

In each of the following three sections, we describe which versions of problem $1\left|p_j g^{[x]}(r)\text{-det}, MP\right| C_{\max}$ are being considered and provide a solution approach to obtain optimal values for Decisions 1-4 for those problems. The eight versions of problem $1\left|p_j g^{[x]}(r)\text{-det}, MP\right| C_{\max}$ are listed out in Table 6.1 along with a reference to the section that deals with it. Each of the three solution approaches have different running times, but they all use Algorithm Match2 as a sub-routine, either as it is or as a revised version.

	Constant Duration MPs		Start-time dependent MPs	
	Identical	Distinct	Identical	Distinct
Group-indep	Section 6.7	Section 6.7	Section 6.6	Section 6.5
Group-dep	Section 6.6	Section 6.5	Section 6.5	Section 6.5

Table 6.1: Different versions of problem $1\left|p_j g^{[x]}(r)\text{-det}, MP\right| C_{\max}$

6.5 Solution Approach PosiJIGD

In this section, we describe a solution approach, which can solve all versions of problem $1\left|p_j g^{[x]}(r)\text{-det}, MP\right| C_{\max}$ without prior knowledge of the number of jobs $n^{[x]}$ in each group. The only pre-requisite condition is that the computed positional weights $W^{[x]}(r)$, $1 \leq r \leq n^{[x]}$, should be non-decreasing in every group x, $1 \leq x \leq k$.

Let us begin our consideration with the most general version of prob-

lem $1\left|p_j g^{[x]}(r)\text{-det}, MP\right|C_{\max}$, such as the one discussed in Section 6.4, with group-dependent deterioration factors and distinct MPs with start-time dependent durations. In order to solve problem $1\left|p_j g^{[x]}(r)\text{-det}, MP\right|C_{\max}$, assume that Decisions 1-3 are taken in advance and denote the resulting problem as $1\left|p_j g^{[x]}(r)\text{-det}, MP(k-1)\right|C_{\max}$. This problem can be solved by minimising the generic objective function (4.6) with positional weights given by (6.8) and the constant term given by (6.9). It is easy to notice that the computed positional weights $W^{[x]}(r)$, $1 \leq r \leq n^{[x]}$, are non-decreasing within each group x, $1 \leq x \leq k$, due to the fact that the positional factors $g^{[x]}(r)$, $1 \leq r \leq n^{[x]}$, are also non-decreasing within a group. Below we outline a solution approach that solves an instance of problem $1\left|p_j g^{[x]}(r)\text{-det}, MP(k-1)\right|C_{\max}$. Let us begin with the following theorem.

Theorem 6.1. *For a given k, $1 \leq k \leq K+1$, if it is possible to compute a set of all possible values of $W^{[x]}(r)$, without prior knowledge of the number of jobs in each group, then from such a set choose the n smallest elements. If the positions associated with the chosen positional weights are consecutive in each group, then assigning the largest jobs to the positions corresponding to the smallest positional weights will ensure that the objective function (4.6) is minimised.*

The proof of Theorem 6.1 is straightforward. Notice that a particular group can have no more than n used positions. If we consider a schedule with k groups, we therefore have a choice of at most nk positions in which n jobs are to be scheduled. Each of these positions have a certain positional weight associated with them. Recall that the contribution of a job $j = \pi^{[x]}(r)$ to the objective function $F(k)$ is given by $W^{[x]}(r)\, p_j$. Thus, in order to ensure the smallest value of the objective function we must choose n positions that generate the smallest positional weights. Having access to a set of all possible values of $W^{[x]}(r)$ enables us identify these positions. If the found positions are consecutively ordered within a group, they may be used to create a feasible schedule. The number of such positions in a group enables us to determine the values $n^{[x]}$, $1 \leq x \leq k$. Finally, an optimal sequence of jobs in these positions can be found by running Algorithm Match2. We now apply Theorem 6.1 to find an optimal solution to problem $1\left|p_j g^{[x]}(r)\text{-det}, MP(k-1)\right|C_{\max}$.

First, set the value $n^{[x]} = n$, $1 \leq x \leq k$, and compute all positional weights $W^{[x]}(r)$, $1 \leq r \leq n$, $1 \leq x \leq k$, by (6.8). Notice that these positional weights represent a set of all possible values of $W^{[x]}(r)$ and can be computed in $O(nk)$ time. As mentioned in Theorem 6.1, an optimal schedule can be found by choosing the n smallest of these values and assigning the largest jobs to the positions corresponding to the smallest positional weights. The following algorithm, which is essentially a version

of Algorithm Match2, formally describes the solution approach.

Algorithm NSmall

INPUT: An instance of problem $1\left|p_j g^{[x]}(r)\text{-det}, MP(k-1)\right|C_{\max}$ with positional weights $W^{[x]}(r)$, $1 \leq r \leq n^{[x]}$, non-decreasing within each group x, $1 \leq x \leq k$

OUTPUT: An optimal schedule $S^*(k)$ defined by the processing sequences $\pi^{[x]}$, $1 \leq x \leq k$

Step 1. If required, renumber the jobs in the LPT order. For each group x, $1 \leq x \leq k$, define an empty processing sequence $\pi^{[i]} := (\varnothing)$ and the weight $W^{[x]} := W^{[x]}(1)$ computed as in (6.8).

Step 2. For each job j from 1 to n do

(a) Find the smallest index v, $1 \leq v \leq k$, with $W^{[v]} = \min\left\{W^{[i]} | 1 \leq i \leq k\right\}$.

(b) Assign job j to group v and place it after the current permutation $\pi^{[v]}$, i.e., define $\pi^{[v]} := (\pi^{[v]}, j)$. Use (6.8) to define $W^{[v]} := W^{[v]}(r)$, where r is the next available position in group v.

Step 3. With the found permutation $\pi^* = \left(\pi^{[1]}, \pi^{[2]}, \ldots, \pi^{[k]}\right)$, compute the optimal value of the objective function $C_{\max}(S^*(k))$ by substituting appropriate values in (4.6).

Step 2a of Algorithm NSmall requires $O(k)$ comparisons, while Step 2b is completed in constant time. Thus, in all Step 2, requires $O(nk)$ time and Step 3 requires $O(n)$ time. The following statement holds.

Theorem 6.2. *Algorithm NSmall solves an instance of problem* $1\left|p_j g^{[x]}(r)\text{-}det, MP(k-1)\right|C_{\max}$ *in* $O(nk)$ *time, provided that the LPT order of the jobs is known.*

In principle, the same solution approach remains valid even for a problem with non-monotone positional factors, since setting $n^{[x]} = n$, $1 \leq x \leq k$, does indeed generate a set of all possible positional weights $W^{[x]}(r)$. However, if the n smallest of these values are chosen in order to obtain an optimal solution, it cannot be guaranteed that consecutive positions (starting from the first position) are filled in each group, thereby resulting in an infeasible solution. To ensure feasibility of the obtained solution it is essential that the obtained positional weights be monotonically ordered within a group.

Let us consider another example for which Algorithm NSmall fails: problem $1\left|p_j g^{[x]}(r)\text{-det}, MP\right|\sum C_j$ with a group-independent polynomial deterioration effect, i.e., $g^{[x]}(r) = r^a$, $a > 0$, and K identical maintenance periods that have start-time dependent durations, i.e., $\alpha^{[x]} = \alpha$ and $\beta^{[x]} = \beta$, $1 \leq x \leq K$. Yang and Yang (2010b) study this problem, and for a known k, calculate the positional weights as

$$W^{[x]}(r) = \begin{cases} \left[\left(n - \sum_{v=1}^{x} n^{[v]}\right)(1+\alpha) + \left(n^{[x]} - r + 1\right)\right] r^a, & 1 \leq r \leq n^{[x]},\ 1 \leq x \leq k-1, \\ \left(n^{[x]} - r + 1\right) r^a & 1 \leq r \leq n^{[x]},\ x = k, \end{cases}.$$

Notice that for the positional weights defined above, it is not possible to generate a set of all possible values of $W^{[x]}(r)$ without prior knowledge of the number of jobs, $n^{[x]}$, in each group. Thus, Theorem 6.1 does not hold and as a result, Algorithm NSmall cannot be used to obtain an optimal solution to problem $1\left|p_j g^{[x]}(r)\text{-det}, MP(k-1)\right|\sum C_j$. This problem is solvable in $O\left(n^k \log n\right)$ time by full enumeration of all possible values of $n^{[x]}$, $1 \leq x \leq k$. We revisit both of these problems in Chapter 9 and solve them for a very general model.

In an optimal solution for an instance of problem $1\left|p_j g^{[x]}(r)\text{-det}, MP(k-1)\right| C_{\max}$ it is possible that out of the k groups, certain groups are not assigned any jobs at all, i.e., $n^{[x]} = 0$. Such a situation can occur if an MP is not efficient in restoring the machine to a better state, and as a result the group that follows generates positional weights with big values. Such an instance can never result in an optimal schedule for the general problem $1\left|p_j g^{[x]}(r)\text{-det}, MP\right| C_{\max}$, as we are unnecessarily spending time to perform an inefficient MP. This instance, if any, will be automatically eliminated from consideration if we try different combinations of Decision 1-3 to define problem $1\left|p_j g^{[x]}(r)\text{-det}, MP(k-1)\right| C_{\max}$.

To determine the optimal solution for the general problem $1\left|p_j g^{[x]}(r)\text{-det}, MP\right| C_{\max}$, all options associated with Decisions 1-3 must be enumerated and the solutions of the resulting sub-problems $1\left|p_j g^{[x]}(r)\text{-det}, MP(k-1)\right| C_{\max}$ be compared. The best of these solutions is chosen as the optimal solution for problem $1\left|p_j g_j^{[x]}(r)\text{-det}, MP\right| C_{\max}$. For a known k, $1 \leq k \leq K+1$, the number of ways to select $k-1$ MPs from K available MPs (Decision 2) is equal to $\binom{K}{k-1}$. Notice that the positional weights given by (6.8) that are associated with the first $k-1$ groups, do not depend on the order of the MPs. However, the last group $x = k$, is differently structured from the others, so it matters which MP is to be scheduled last, i.e., at the $(k-1)$-th position. Thus, the number of choices for Decision 3 is equal to $k-1$. Trying all possible values of (Decision 1) k, $1 \leq k \leq K+1$, the total number of options can be approximated by $\sum_{k=1}^{K+1}\binom{K}{k-1}(k-1)$. Since Algorithm NSmall requires $O(nk)$

time to run for a given k, $1 \leq k \leq K+1$, the total running time required to solve the most general version of problem $1\left|p_j g^{[x]}(r)\text{-det}, MP\right| C_{\max}$ can be estimated as $O\left(\sum_{k=1}^{K+1} nk\binom{K}{k-1}(k-1)\right) = O(nK^2 2^K)$, which is linear in n for a constant K.

Now let us consider other less general versions of problem $1\left|p_j g^{[x]}(r)\text{-det}, MP\right| C_{\max}$. In all we consider a total of four different cases as outlined in Table 6.2. For each case compute the positional weights $W^{[x]}(r)$, $1 \leq r \leq n^{[x]}$, $1 \leq x \leq k$, by making appropriate substitutions to the general formula given by (6.8). It is easy to verify that as long as the positional factors are non-decreasing within a group, the generated positional weights also remain non-decreasing within each group, even for less general models. It follows that Algorithm NSmall must be used to solve the resulting sub-problems of the form $1\left|p_j g^{[x]}(r)\text{-det}, MP(k-1)\right| C_{\max}$. Table 6.2 states the number of times Algorithm NSmall must be run in order to solve different versions of problem $1\left|p_j g^{[x]}(r)\text{-det}, MP\right| C_{\max}$.

	Constant Duration MPs		Start-time dependent MPs	
	Identical	Distinct	Identical	Distinct
Group-indep	Section 6.7	Section 6.7	Section 6.6	$\sum_{k=1}^{K+1}\binom{K}{k-1}$
Group-dep	Section 6.6	$\sum_{k=1}^{K+1}\binom{K}{k-1}$	$\sum_{k=1}^{K+1} 1$	$\sum_{k=1}^{K+1}\binom{K}{k-1}(k-1)$

Table 6.2: Number of times to run Algorithm NSmall to solve different versions of problem $1\left|p_j g^{[x]}(r)\text{-det}, MP\right| C_{\max}$.

Notice that although Algorithm NSmall is able to solve all eight versions of problem $1\left|p_j g^{[x]}(r)\text{-det}, MP\right| C_{\max}$, for some cases it is possible to make Decisions 1-3 on the fly by using another solution approach, which allows the optimal solution to be found in faster time. For such cases, a reference to the relevant section has been made in Table 6.2.

First, consider a version of problem $1\left|p_j g^{[x]}(r)\text{-det}, MP\right| C_{\max}$, with group-independent deterioration factors, i.e., $g^{[x]}(r) = g(r)$, $1 \leq r \leq n^{[x]}$, $1 \leq x \leq k$, and distinct MPs with start-time dependent durations. This problem corresponds to a scenario in which distinct MPs are performed in the schedule, but they all restore the machine to the same state. Making relevant substitutions in (6.8) the computed positional weights can be written as

$$W^{[x]}(r) = \begin{cases} (1+\alpha^{[x]})g(r), & 1 \leq r \leq n^{[x]},\ 1 \leq x \leq k-1, \\ g(r) & 1 \leq r \leq n^{[x]},\ x = k, \end{cases}$$

whereas the constant term $\Gamma(k)$ remains as before. Since each MP restores the machine to the same state, Decision 2 is made only on the basis of the durations of the MPs.

There are two parameters that define the MP durations, therefore, there is no easy way by which the best $k-1$ MPs can selected out of the available K MPs. Thus, all possible selections need to be tried and this can be done in $\binom{K}{k-1}$ ways. Moreover, the found positional weights appear to be independent of the order of the MPs. Thus, any choice of Decision 3 can be considered optimal. Trying all possible values of (Decision 1) k, $1 \leq k \leq K+1$, the total number of options can be estimated by $\sum_{k=1}^{K+1} \binom{K}{k-1}$. Thus, the total running time required to solve this version of problem $1 \left| p_j g^{[x]}(r)\text{-det}, MP \right| C_{\max}$ can be estimated as $O\left(\sum_{k=1}^{K+1} nk\binom{K}{k-1}\right) = O\left(nK2^K\right)$, which is linear in n for a constant K.

For the problem with group-dependent deterioration factors and distinct MPs with constant durations, i.e., $\alpha^{[x]} = 0$, $1 \leq x \leq k-1$, the computed positional weights can be written as

$$W^{[x]}(r) = g^{[x]}(r),\ 1 \leq r \leq n^{[x]},\ 1 \leq x \leq k.$$

Clearly, these positional weights are dependent on the type of the MP, but not on their order. Thus, as before the total number of options can be given by $\sum_{k=1}^{K+1} \binom{K}{k-1}$ and total running time required to solve this version of problem $1 \left| p_j g^{[x]}(r)\text{-det}, MP \right| C_{\max}$ can be estimated as $O\left(nK2^K\right)$.

Lastly, consider the problem with group-dependent deterioration factors and identical MPs with start-time dependent durations, i.e., $\alpha^{[x]} = \alpha$, $\beta^{[x]} = \beta$, $1 \leq x \leq k-1$. This problem corresponds to a scenario in which identical start time dependent MPs are performed in the schedule, but they all restore the machine to a different state. As described earlier, such a situation is possible when the MPs fail to completely repair the machine. Since the MPs are identical, clearly Decision 2 and 3 do not have an effect on the optimal solution. Thus, only Decision 1 needs to be taken. Trying all possible values of k, $1 \leq k \leq K+1$, an optimal solution to this version of problem $1 \left| p_j g^{[x]}(r)\text{-det}, MP \right| C_{\max}$ can be found in $O\left(\sum_{k=1}^{K+1} nk\right) = O\left(nK^2\right)$ time.

In our paper Rustogi and Strusevich (2012a), we use Algorithm NSmall to solve problem $1 \left| p_j g^{[x]}(r)\text{-det}, MP \right| C_{\max}$ with group-dependent deterioration factors and distinct MPs with start-time dependent durations. However, no emphasis has been made on Decisions 2 and 3. The decision-maker only needs to decide how many MPs from a given list of $n-1$ available MPs to include into the schedule. The published running time to solve this problem is $O\left(n^3\right)$, which is consistent with the results of this section for $K = n-1$.

6.6 Solution Approach PosiJIKdomi

In this section, we consider versions of problem $1\left|p_j g^{[x]}(r)\text{-det}, MP\right| C_{\max}$ in which Decisions 1-4 can be made on the fly, without having to enumerate all possible options. We list out two versions for which this is possible:

Problem 1: Problem $1\left|p_j g^{[x]}(r)\text{-det}, MP\right| C_{\max}$ with group-dependent deterioration factors and distinct MPs with constant durations, subject to the condition that if a certain MP with duration $\beta^{[y]}$ is included in the schedule, then the group that follows will contain the positional factors $g^{[y+1]}(r)$, $1 \leq r \leq n$, such that they can be ordered in a way that

$$g^{[1]}(r) \leq g^{[2]}(r) \leq \cdots \leq g^{[K+1]}(r),\ 1 \leq r \leq n, \tag{6.10}$$

and

$$\beta^{[1]} \leq \beta^{[2]} \leq \cdots \leq \beta^{[K]}, \tag{6.11}$$

hold simultaneously, where the positional factors $g^{[1]}(r)$, $1 \leq r \leq n$, are associated with the group that is created before the first MP. This version is a generalisation of one of the cases found in Table 6.2, in which group-dependent deterioration factors are considered along with identical MPs of constant duration, i.e., $\alpha^{[x]} = 0$, $\beta^{[x]} = \beta$, $1 \leq x \leq K$. For the latter problem, it is safe to assume that (6.10) will hold based on the following argument. If identical MPs of equal duration are performed on the machine then after an MP the condition of the machine can be no better than its condition after the previous MP. In such a case every position (including the first position) will have a worse deterioration factor than its counterpart in an earlier group.

Problem 2: Problem $1\left|p_j g^{[x]}(r)\text{-det}, MP\right| C_{\max}$ with group-independent deterioration factors and distinct MPs with start-time dependent durations, subject to the condition that the duration parameters of the MPs can be ordered such that

$$\alpha^{[1]} \leq \alpha^{[2]} \leq \cdots \leq \alpha^{[K]}, \tag{6.12}$$

and (6.11) hold simultaneously. This version is a generalisation of one of the cases found in Table 6.2, in which group-independent positional factors are considered along with identical MPs of start-time dependent duration. The latter problem corresponds to a scenario in which identical MPs are performed in the schedule and they all restore the machine to the same state.

In order to solve both versions of problem $1\left|p_j g^{[x]}(r)\text{-det}, MP\right| C_{\max}$ described above, the optimal choice for Decisions 2 and 3 can be made easily. If Decision 1 is assumed to be taken, so that $k-1$, $1 \leq k \leq K+1$, MPs are included in the schedule, then for both problems, the MPs with the indices $1, 2, \ldots, k-1$ are chosen and scheduled in the same order. This is the optimal choice for Problem 1 as the MPs with the indices $1, 2, \ldots, k-1$ have the smallest durations and create groups that contain the smallest deterioration factors, owing to (6.10) and (6.11) holding simultaneously. This is the optimal choice for Problem 2 as all MPs create identical groups and the ones with a smaller index have smaller values of the duration parameters, owing to (6.12) and (6.11) holding simultaneously. Notice that the order of the MPs is inconsequential in both cases.

With Decisions 1-3 having been made (with an assumed value of k), denote the resulting problem as $1\left|p_j g^{[x]}(r)\text{-det}, MP(k-1)\right| C_{\max}$. This problem can be solved by minimising the generic objective function of the form (4.6). For Problem 1, obtain the required positional weights $W^{[x]}(r)$ by substituting $\alpha^{[x]} = 0$, $1 \leq x \leq k$, in (6.8) so that we have

$$W^{[x]}(r) = g^{[x]}(r),\ 1 \leq r \leq n^{[x]},\ 1 \leq x \leq k, \tag{6.13}$$

and for Problem 2, substitute $g^{[x]}(r) = g(r)$, $1 \leq r \leq n^{[x]}$, $1 \leq x \leq k$, so that we have

$$W^{[x]}(r) = \begin{cases} (1+\alpha^{[x]})g(r), & 1 \leq r \leq n^{[x]},\ 1 \leq x \leq k-1, \\ g(r) & 1 \leq r \leq n^{[x]},\ x = k. \end{cases} \tag{6.14}$$

Set the value $n^{[x]} = n$, $1 \leq x \leq k$, and $k = K+1$, and compute all positional weights $W^{[x]}(r)$, $1 \leq r \leq n$, $1 \leq x \leq K+1$, for both problems by using the formulae above. Notice that the computed positional weights represent a set of all possible values of $W^{[x]}(r)$ across all possible groups. Further, notice that because of (6.10), the positional weights associated with Problem 1 are ordered such that for each k, $1 \leq k \leq K+1$, we have

$$W^{[1]}(r) \leq W^{[2]}(r) \leq \cdots \leq W^{[k]}(r),\ 1 \leq r \leq n,$$

and because of (6.12), the positional weights for Problem 2 are ordered such that for each k, $1 \leq k \leq K+1$, we have

$$W^{[k]}(r) \leq W^{[1]}(r) \leq W^{[2]}(r) \leq \cdots \leq W^{[k-1]}(r),\ 1 \leq r \leq n.$$

Definition 6.1. *If for each position the positional weight in group x is smaller than the positional weight in the same position in another group y, we say that group x dominates group y. If all available groups can be linearly ordered with the respect to*

the introduced dominance relation, we refer to such a condition as 'K-domi'.

Notice that both Problems 1 and 2 satisfy the *K-domi* condition. For instances of problem $1\left|p_j g^{[x]}(r)\text{-det}, MP\right| C_{\max}$ that satisfy (6.11) and the *K-domi* condition, it is possible to compute the optimal values of $n^{[x]}$, $1 \leq x \leq k$, and Decisions 1-4 on the fly. Recall that for a fixed value of Decision 1, the optimal values for Decisions 2 and 3 are already known. Thus, to solve problem $1\left|p_j g^{[x]}(r)\text{-det}, MP\right| C_{\max}$ we only need to worry about finding the optimal values for Decisions 1 and 4. We base our methodology on Theorem 6.1.

Let $G(k)$ denote a list of n smallest positional weights that are available across all positions from k, $1 \leq k \leq K+1$, groups. The list is sorted in non-decreasing order. Let $\gamma_i(k)$ denote the i-th element in the list $G(k)$, so that $G(k) = (\gamma_1(k), \gamma_2(k), \ldots, \gamma_n(k))$. This implies that

$$C_{\max}(S^*(k)) = P(S^*(k)) + \Gamma(k) = \sum_{j=1}^{n} p_j \gamma_j(k) + \sum_{x=1}^{k-1} \beta^{[x]}. \tag{6.15}$$

where $P(S^*(k))$ denotes the sum of actual durations of the jobs in an optimal schedule with k groups, and $\Gamma(k)$ is a constant term as defined in (6.9).

Create another list $H(v)$, $1 \leq v \leq K+1$, which is defined differently for Problems 1 and 2. For Problem 1, $H(v)$ contains the positional weights $W^{[v]}(r)$, $1 \leq r \leq n-v+1$, for $n^{[x]} = n$, so that by (6.13) we have $H(v) := \left(g^{[v]}(1), g^{[v]}(2), \ldots, g^{[v]}(n-v+1)\right)$, $1 \leq v \leq K+1$. For Problem 2, notice that the values of the positional weights given by (6.14) change dynamically as the value of k is changed. Thus, we define $H(v)$ so that this effect is incorporated; define $H(1) := (g(1), g(2), \ldots, g(n))$ and $H(v) := \left(\left(1+\alpha^{[v-1]}\right) g(1), \left(1+\alpha^{[v-1]}\right) g(2), \ldots, \left(1+\alpha^{[v-1]}\right) g(n-v+1)\right)$, $2 \leq v \leq K+1$. Notice that for both problems, list $H(v)$ has at most $n-v+1$, $1 \leq v \leq K+1$, elements sorted in a non-decreasing order. It suffices to consider only $n-v+1$ positions in a list $H(v)$, as due to condition '*K-domi*' it can be ensured that each of the $v-1$ earlier groups will have at least 1 job scheduled in them.

The following algorithm solves an instance of problem $1\left|p_j g^{[x]}(r)\text{-det}, MP\right| C_{\max}$ and returns the optimal number of MPs, k^*-1, to be included in the schedule (Decision 1) along with the optimal schedule $S^*(k^*)$ with k^* groups (Decision 4).

Algorithm NSmall2

INPUT: An instance of problem $1\left|p_j g^{[x]}(r)\text{-det}, MP\right| C_{\max}$ that satisfies '*K-domi*' and (6.11)

OUTPUT: An optimal schedule $S^*(k^*)$ defined by the processing sequences $\pi^{[x]}$, $1 \leq x \leq k^*$

Step 1. If required, renumber the jobs in the LPT order. For $k = 1$, define a sorted list

$$G(1) := H(1).$$

Compute $C_{\max}(S(1))$ by formula (6.15). Define $k' := K + 1$.

Step 2. For k from 2 to k' do

(a) Create the list $G(k) = (\gamma_1(k), \gamma_2(k), \ldots, \gamma_n(k))$ that contains n smallest elements in the merger of the lists $G(k-1)$ and $H(k)$.

(b) Compute $C_{\max}(S^*(k))$ by formula (6.15). If $P(S^*(k)) = P(S^*(k-1))$ then define $k' := k - 1$ and break the loop by moving to Step 3; otherwise, continue the loop with the next value of k.

Step 3. Find the value k^*, $1 \leq k^* \leq k'$, such that

$$C_{\max}(S^*(k^*)) = \min\{C_{\max}(S^*(k)) \,|\, 1 \leq k \leq k'\}.$$

Step 4. Run Algorithm NSmall for the found value of k^* to obtain the optimal processing sequence $\pi^* = \left(\pi^{[1]}, \pi^{[2]}, \ldots, \pi^{[k^*]}\right)$.

Notice that similar to Algorithm NSmall, Step 2 of the above algorithm also follows Theorem 6.1 and searches for the n smallest positional weights for a given k, $1 \leq k \leq K + 1$, and assigns the largest jobs to the positions corresponding to the smallest positional weights. The main difference between the two algorithms lies in the way the list of n smallest positional weights is found. For each k, $1 \leq k \leq K + 1$, Algorithm NSmall searches for the n smallest positional weights by comparing the elements of the set of nk positional weights across all groups. Algorithm NSmall2 on the other hand, searches for the n smallest positional weights only by comparing the values of the two lists $G(k-1)$ and $H(k)$. Recall that list $G(k-1)$ contains all the positional weights corresponding to the positions used in schedule $S^*(k-1)$, while the list $H(k)$ contains the positional weights that will be introduced if the k-th group is opened. This method is justified, because list $G(k-1)$ already contains the n smallest

positional weights coming from the first $k-1$ groups. Thus, to search for the n smallest weights needed for schedule $S(k)$, there is no need to scan the first $k-1$ groups again. In other words, we utilise the fact, that if a certain position in the first $k-1$ groups is not used in schedule $S^*(k-1)$, it will not be used in schedule $S^*(k)$ as well.

Since both lists $G(k-1)$ and $H(k)$, have at most n elements sorted in a non-decreasing order, Step 2a can be completed in $O(n)$ time. Step 2b also requires $O(n)$ time. Steps 2a and 2b are repeated for several values of k, $1 \leq k \leq K+1$. The iteration on k is stopped if in Step 2b the condition $P(S^*(k)) = P(S^*(k-1))$ is obtained. The condition $P(S^*(k)) = P(S^*(k-1))$ implies that the addition of the k-th group does not provide any positional weights smaller than those in the list $G(k-1)$. If this happens for the k-th group, all groups that will be opened after this will provide even worse positional weights because the list $H(k+1)$ is dominated by the list $H(k)$, $1 \leq k \leq K$. Thus, the makespan cannot be reduced by running more MPs after the k'-th group is opened. This implies that no further values of k should be examined and the best schedule should be found from the set $\{S^*(k) \,|1 \leq k \leq k'\}$. If the loop is not broken throughout the run of Algorithm NSmall, the default value of k' is set to $K+1$. Thus, at most Steps 2a and 2b are repeated $K+1$ times and the following statement holds.

Theorem 6.3. *Algorithm NSmall2 solves an instance of problem* $1\left|p_j g^{[x]}(r)\text{-}det, MP\right| C_{\max}$ *defined by Problems 1 and 2 in* $O(nK)$ *time, provided that the LPT order of the jobs is known.*

Algorithm NSmall2 can also be applied to solve problem $1\left|p_j g^{[x]}(r)\text{-det}, MP\right| C_{\max}$ with group-independent deterioration factors and constant duration MPs (both identical and distinct). This is possible since both conditions '*K-domi*' and (6.11) can be satisfied simultaneously. The required running time is again $O(nK)$. We do not discuss the solution of this problem here, as it is possible to solve it faster using another solution approach, discussed in Section 6.7.

Now consider a version of problem $1\left|p_j g^{[x]}(r)\text{-det}, MP\right| C_{\max}$ in which the conditions '*K-domi*' and (6.11) do not hold simultaneously. In principle, Algorithm NSmall can still be used to obtain a solution for Decisions 1 and 4, but to make Decisions 2 and 3, a full enumeration of options might be required. As a result, the overall running time to solve problem $1\left|p_j g^{[x]}(r)\text{-det}, MP\right| C_{\max}$ turns out to be no smaller than that obtained by using Solution Approach PosiJIJD presented in Section 6.5. In our paper Rustogi and Strusevich (2012a), we have used Algorithm NSmall2 to solve a version of problem $1\left|p_j g^{[x]}(r)\text{-det}, MP\right| C_{\max}$, in which '*K-domi*' holds, but (6.11) does not. Additionally, we have assumed that for each k, $1 \leq k \leq K+1$, Decisions 2 and 3 are fixed. As a result, an optimal solution is obtained in $O(nK)$ time, where $K = n-1$.

Below we provide a numerical example that illustrates the working of Algorithm NSmall2.

Example 6.1. Consider a version of problem1 $\left|p_j g^{[x]}(r)\text{-det}, MP\right| C_{\max}$ as defined in Problem 2. Six jobs are to be scheduled which have the following normal processing times listed in an LPT order

$$p_1 = 10, p_2 = 9, p_3 = 6, p_4 = 3, p_5 = 3, p_6 = 2.$$

The decision-maker has a choice of $K = 5$ MPs, with the following parameters

$$\begin{aligned}
\alpha^{[1]} &= 1,\ \beta^{[1]} = 1;\\
\alpha^{[2]} &= 1,\ \beta^{[2]} = 2;\\
\alpha^{[3]} &= 2,\ \beta^{[3]} = 3;\\
\alpha^{[4]} &= 2,\ \beta^{[4]} = 4;\\
\alpha^{[5]} &= 3,\ \beta^{[5]} = 4.
\end{aligned}$$

Each of the MPs restore the machine to its original state. The positional factors associated with the machine are as follows

$$g(1) = 1,\ g(2) = 2,\ g(3) = 2,\ g(4) = 3,\ g(5) = 3,\ g(6) = 4.$$

j	p_j	$G(1)$	$p_j\gamma_j$	$H(2)$	$G(2)$	$p_j\gamma_j$	$H(3)$	$G(3)$	$p_j\gamma_j$	$H(4)$	$G(4)$	$p_j\gamma_j$
1	10	1	10	2	1	10	2	1	10	3	1	10
2	9	2	18	4	2	18	4	2	18	6	2	18
3	6	2	12	4	2	12	4	2	12	6	2	12
4	3	3	9	6	2	6	6	2	6		2	6
5	3	3	9	6	3	9		2	6		2	6
6	2	4	8		3	6		3	6		3	6
		$P(S^*(1))$	66		$P(S^*(2))$	61		$P(S^*(3))$	58		$P(S^*(4))$	58
		$\Gamma(1)$	0		$\Gamma(2)$	1		$\Gamma(3)$	3		$\Gamma(4)$	6
		$C_{\max}(S^*(1))$	66		$C_{\max}(S^*(2))$	62		$C_{\max}(S^*(3))$	61		$C_{\max}(S^*(4))$	64

Table 6.3: Run of Algorithm NSmall2 for Example 6.1

Table 6.3 shows the details of the run of Algorithm 2 for the above instance. Since $P(S^*(3)) = P(S^*(4))$, the algorithm stops after the iteration $k = 4$, so that $k' = 3$. The algorithm outputs the minimum value of the makespan from the set $\{C_{\max}(S^*(k))|1 \leq k \leq 3\}$, which is $C_{\max}(S^*(3))$. In an optimal schedule for $k^* = 3$, the sequence of

jobs $\pi^{[1]} = (1, 2, 3, 6)$ is processed in the first group, the sequence of jobs $\pi^{[2]} = (4)$ is processed in the second group and the sequence of jobs $\pi^{[3]} = (5)$ is processed in the third group. The makespan of the resulting schedule is 61.

We take the productive idea behind Algorithm NSmall2 further, to the models with job-dependent deterioration effects; see Chapter 7.

6.7 Solution Approach PosiJIGI

In this section, we deal with problems in which the computed positional weights are group-independent, i.e., of the form $W^{[x]}(r) = W(r)$, $1 \leq x \leq k$, and additionally, they are ordered in a way such that $W(1) \leq W(2) \leq \cdots \leq W(n)$. Such a situation arises for versions of problem $1 \left| p_j g^{[x]}(r)\text{-det}, MP \right| C_{\max}$, in which the deterioration factors are group-independent, i.e., $g^{[x]}(r) = g(r)$, $1 \leq r \leq n$, $1 \leq x \leq K+1$. The MPs are of constant duration and can be either distinct or identical. The problem with identical MPs is a special case of the problem with distinct MPs and does not result in better running times. Thus, we only consider the latter problem.

Formally, we denote the described problem as $1 \left| p_j g(r)\text{-det}, MP[0] \right| C_{\max}$. The first term in the middle field is used to notify that the deterioration factors are group-independent. The second term $MP[0]$ in the middle field is used to notify that the MPs are of constant duration, i.e., in (4.1) we have $\alpha^{[y]} = 0$.

Notice that for problem $1 \left| p_j g(r)\text{-det}, MP[0] \right| C_{\max}$, the optimal choice for Decisions 2 and 3 can be made easily. Assume that an optimal solution to problem $1 \left| p_j g(r)\text{-det}, MP[0] \right| C_{\max}$ includes $k-1$ MPs in the schedule, so that the jobs are divided into k, $1 \leq k \leq K+1$, groups. Since it is known that the MPs create identical groups, it follows that the order in which they are performed is not important. Further, it is obvious that in order to choose $k-1$ MPs out of the available K, the ones with smaller durations are given priority. To ensure that the smallest $k-1$ MPs are chosen in an optimal schedule, we renumber the K available MPs in a way that (6.11) holds and select the ones with indices $1, 2, \ldots, k-1$. Lastly, we fix their order as per their index numbers.

With Decisions 1-3 having been made (with an assumed value of k), the resulting problem $1 \left| p_j g(r)\text{-det}, MP[0](k-1) \right| C_{\max}$ can be solved by minimising the generic objective function (4.6). Obtain the required positional weights $W^{[x]}(r)$ by substituting $g^{[x]}(r) = g(r)$, $1 \leq r \leq n$, and $\alpha^{[x]} = 0$, $1 \leq x \leq k$, in (6.8) so that we have $W^{[x]}(r) = g(r)$, $1 \leq r \leq n^{[x]}$, $1 \leq x \leq k$. Below we outline a solution

approach which is again based on Theorem 6.1, and solves an instance of problem $1\left|p_{j}g\left(r\right)\text{-det},MP\left[0\right]\left(k-1\right)\right|C_{\max}$.

Notice that for a given position r, $1\leq r\leq n$, the positional weights are the same for every group and within each group they are sorted in a non-decreasing order. This implies that unlike the problems dealt with previously, the n smallest positional weights for this problem are already known. The smallest k positional weights are due to the first positions of each of the k groups. The next smallest k positional weights are due to the second positions of each of the k groups, and so on. As a result, the optimal number of jobs in each group can be given by

$$n^{[x]}=\begin{cases}\left\lceil \frac{n}{k}\right\rceil , & 1\leq x\leq \mod\left(n,k\right),\\ \left\lfloor \frac{n}{k}\right\rfloor , & \mod\left(n,k\right)+1\leq x\leq k.\end{cases}\tag{6.16}$$

where $\mod\left(n,k\right)$ is the remainder of the division of n by k.

With known values of $W^{[x]}(r)$, $1\leq r\leq n^{[x]}$, and $n^{[x]}$, $1\leq x\leq k$, an optimal schedule $S^{*}\left(k\right)$ for problem $1\left|p_{j}g\left(r\right)\text{-det},MP\left[0\right]\left(k-1\right)\right|C_{\max}$ can be found by running Algorithm Match2. For solving problem $1\left|p_{j}g\left(r\right)\text{-det},MP\left[0\right]\left(k-1\right)\right|C_{\max}$, Step 2 of Algorithm Match2 can be completed in constant time as the list of sorted positional weights is already known and is given by

$$L=\begin{pmatrix}k\text{ times }g\left(1\right),\\ k\text{ times }g\left(2\right),\\ \cdots\\ k\text{ times }g\left(\left\lfloor \frac{n}{k}\right\rfloor \right),\\ \mod\left(n,k\right)\text{ times }g\left(\left\lceil \frac{n}{k}\right\rceil \right)\end{pmatrix}.$$

Step 3 requires $O\left(n\right)$ time for computing the value of the objective function $C_{\max}\left(S^{*}\left(k\right)\right)$. Thus, the following statement holds.

Theorem 6.4. *Algorithm Match2 solves an instance of problem* $1\left|p_{j}g\left(r\right)\text{-}det,MP\left[0\right]\left(k-1\right)\right|C_{\max}$ *in* $O\left(n\right)$ *time, provided that the LPT order of the jobs is known.*

To determine the optimal solution for problem $1\left|p_{j}g\left(r\right)\text{-det},MP\left[0\right]\right|C_{\max}$, we only need to worry about the optimal value of the number of MPs, as Decisions 2 and 3 have already been chosen optimally. We can do this by solving problem $1\left|p_{j}g\left(r\right)\text{-det},MP\left[0\right]\left(k-1\right)\right|C_{\max}$ for all values of k, $1\leq k\leq K+1$, and choosing the best instance as an optimal solution. Thus, problem $1\left|p_{j}g\left(r\right)\text{-det},MP\left[0\right]\right|C_{\max}$ can be solved in $O\left(nK\right)$ time.

Recall from Section 3.3.1, that Kuo and Yang (2008a) study a special case of problem $1\left|p_j g\left(r\right)\text{-det}, MP\left[0\right]\right|C_{\max}$ with a polynomial deterioration function given by $g(r) = r^a, a > 0$, and identical MPs. In order to solve this problem, Kuo and Yang (2008a) solve a total of $K+1 = n$ sub-problems denoted by $1\left|p_j r^a\text{-det}, MP\left[0\right]\left(k-1\right)\right|C_{\max}$, and prove what they call the *group balance principle.* Notice that so far in this section, our solution approach is very similar to that provided by Kuo and Yang (2008a). Indeed, the number of jobs in each group given by (6.16) conforms with the group balance principle. Even our running time of $O\left(nK\right)$ needed to solve problem $1\left|p_j g\left(r\right)\text{-det}, MP\left[0\right]\right|C_{\max}$, is consistent with that of Kuo and Yang (2008a), who solve problem $1\left|p_j r^a\text{-det}, MP\left[0\right]\right|C_{\max}$ in $O\left(n^2\right)$ time, for $K = n-1$.

Further in their paper, Kuo and Yang (2008a) have made a conjecture that in the case of a polynomial deterioration function, the sequence of values $C_{\max}(S^*\left(k\right))$, $1 \leq k \leq K+1$, might be V-shaped with respect to k. Recall from Chapter 5 that a sequence $A(k)$ is called V-shaped if there exists a k_0, $1 \leq k_0 \leq K+1$, such that

$$A(1) \geq \cdots \geq A(k_0-1) \geq A(k_0) \leq A(k_0+1) \leq \cdots \leq A(K+1).$$

If this were true for $C_{\max}(S^*\left(k\right))$, $1 \leq k \leq K+1$, then instead of $K+1$, at most $\lceil \log_2\left(K+1\right)\rceil$ values of k should be tried to solve the original problem $1\left|p_j r^a\text{-det}, MP\left[0\right]\right|C_{\max}$. Below we show that even for a general deterioration function $g\left(r\right)$, the sequence $C_{\max}(S^*\left(k\right))$, $1 \leq k \leq K+1 \leq n$, is in fact V-shaped. Notice that in a schedule with n jobs, it is not possible to have more than n groups, i.e., $1 \leq k \leq n$. Thus, K is bounded by $n-1$. We start with the following statement.

Lemma 6.1. *For problem* $1\left|p_j g(r), MP\left[0\right]\left(k-1\right)\right|C_{\max}$*, if the jobs be numbered in the LPT order (2.7), then the makespan of the optimal schedule can be written as*

$$C_{\max}(S^*\left(k\right)) = P(S^*\left(k\right)) + \Gamma(k) = \sum_{j=1}^{n} p_j g\left(\left\lceil \frac{j}{k} \right\rceil\right) + \sum_{x=1}^{k-1} \beta^{[x]}. \tag{6.17}$$

Proof: The value $C_{\max}(S\left(k\right))$ can be seen as $P(S\left(k\right)) + \Gamma(k)$, where $P(S\left(k\right))$ denotes the sum of the actual durations of the jobs in a schedule $S\left(k\right)$, and $\Gamma(k)$ is the total duration of all $k-1$ MPs. If the jobs are numbered in the LPT order, then to minimise the value $P(S(k))$, we need to assign the jobs one by one to the smallest available position. This can be done by distributing the first k jobs to the first positions in each of the k groups, then the next k jobs going to the second positions in each of the k groups, and so on, until all jobs have been sequenced.

If $j = ak$ then the predecessors of j are placed into the first a positions of groups

$1, 2, \ldots, k-1$ and take $a-1$ positions of group k, so that job j gets position $a = \left\lceil \frac{j}{k} \right\rceil$. If $j = ak + b$ for $1 \leq b \leq k-1$, then the predecessors of j will take the first a positions in each group and additionally the $(a+1)$-th position in each of the groups $1, 2, \ldots, b-1$, so that job j gets position $a + 1 = \left\lceil \frac{j}{k} \right\rceil$ in group b.

It follows that the actual processing time of a job $j \in N$ in an optimal schedule $S^*(k)$ is equal to $p_j g\left(\left\lceil \frac{j}{k} \right\rceil\right)$, and the total processing time for all jobs is equal to

$$P(S^*(k)) = \sum_{j=1}^{n} p_j g\left(\left\lceil \frac{j}{k} \right\rceil\right). \tag{6.18}$$

□

For problem $1\,|p_j g(r)\text{-det}, MP[0]|\,C_{\max}$, we need to determine the optimal number of groups k^* to be opened such that the makespan $C_{\max}(S^*(k))$, $1 \leq k \leq K+1 \leq n$, is minimised. As k increases, $P(S^*(k))$ becomes smaller since new groups are added and a greater number of smaller positions become available. At the same time, $\Gamma(k)$ becomes larger, with more maintenance activities being performed. The sequence $C_{\max}(S^*(k))$ captures the trade-off between its two components, $P(S^*(k))$ and $\Gamma(k)$.

Theorem 6.5. *For problem $1\,|p_j g(r)\text{-det}, MP[0]|\,C_{\max}$, the sequence $C_{\max}(S^*(k)) = P(S^*(k)) + \Gamma(k)$, $1 \leq k \leq K+1 \leq n$, given by (6.17), is V-shaped.*

Proof: Recall Theorem 5.2, in which it is stated that a sequence

$$P(k) = \sum_{j=1}^{n} p_j g\left(\left\lceil \frac{j}{k} \right\rceil\right),\ 1 \leq k \leq n,$$

is convex, if $p_1 \geq p_2 \geq \cdots \geq p_n$. Thus, the sequence $P(S^*(k))$, $1 \leq k \leq n$, given by (6.18) is convex. The sequence $\Gamma(k) = \sum_{x=1}^{k-1} \beta^{[x]}$, $1 \leq k \leq n$, can also be proved to be convex. Recall from Chapter 5 that a sequence $\Gamma(k)$, $1 \leq k \leq n$, is called convex if

$$\Gamma(k) \leq \frac{1}{2}\left(\Gamma(k-1) + \Gamma(k+1)\right),\ 2 \leq k \leq n-1.$$

Substituting $\Gamma(k) = \sum_{x=1}^{k-1} \beta^{[x]}$ in the above inequality, we obtain the following

inequalities:

$$\begin{aligned}\sum_{x=1}^{k-1}\beta^{[x]} &\leq \frac{1}{2}\left(\sum_{x=1}^{k-2}\beta^{[x]}+\sum_{x=1}^{k}\beta^{[x]}\right),\ 2\leq k\leq n-1,\\ \beta^{[k-1]} &\leq \frac{1}{2}\beta^{[k-1]}+\frac{1}{2}\beta^{[k]},\ 2\leq k\leq n-1,\\ \beta^{[k-1]} &\leq \beta^{[k]},\ 2\leq k\leq n-1.\end{aligned}$$

The last inequality is true as (6.11) holds. The equality holds for the case in which the MPs have identical durations, see, e.g., the problem considered by Kuo and Yang (2008a). Thus, the sequence $\Gamma(k)$ is convex. Since the sum of two convex sequences is convex, the sequence $C_{\max}\left(S^*(k)\right)=P(S^*\left(k\right))+\Gamma(k)$, $1\leq k\leq K+1\leq n$, is also convex and by Lemma 5.1 is V-shaped. □

Theorem 6.5 allows us to find the optimal number of groups k^*, $1\leq k\leq K+1$, to be created by the following binary search algorithm.

Algorithm BinarySearch

INPUT: An instance of problem $1\left|p_jg(r)\text{-det},MP\left[0\right]\right|C_{\max}$

OUTPUT: The optimal number of MPs to include in the schedule

Step 0. If required renumber the jobs in an LPT order and renumber the MPs in a non-decreasing order of their durations so that $\beta^{[1]}\leq\beta^{[2]}\leq\cdots\leq\beta^{[K]}$.

Step 1. Define $\underline{k}:=1$, $\overline{k}:=K+1$ and $\widetilde{k}:=\lceil(K+1)/2\rceil$. Compute $C_{\max}(S^*\left(k\right))$ by formula (6.17) for $k\in\left\{\underline{k},\widetilde{k},\overline{k}\right\}$.

Step 2. If $C_{\max}\left(S^*(\underline{k})\right)\leq C_{\max}\left(S^*(\widetilde{k})\right)$ then go to Step 3; otherwise, go to Step 4

Step 3. Redefine $\overline{k}:=\widetilde{k}$, $C_{\max}\left(S^*(\overline{k})\right):=C_{\max}\left(S^*(\widetilde{k})\right)$ and go to Step 5.

Step 4. Redefine $\underline{k}:=\widetilde{k}$, $C_{\max}\left(S^*(\underline{k})\right):=C_{\max}\left(S^*(\widetilde{k})\right)$ and go to Step 5.

Step 5. Redefine $\widetilde{k}:=\left\lceil\left(\underline{k}+\overline{k}\right)/2\right\rceil$. If $\underline{k}=\widetilde{k}=\overline{k}$, then output $k^*=\underline{k}$, and stop; otherwise, compute $C_{\max}\left(S^*(\widetilde{k})\right)$ and go to Step 2.

It is clear that due to the V-shapeness of the sequence $C_{\max}(S^*\left(k\right))$, $1\leq k\leq n$, the inequality $C_{\max}\left(S^*(\underline{k})\right)\leq C_{\max}\left(S^*(\widetilde{k})\right)$ implies that the subsequence $C_{\max}\left(S^*(k)\right)$, $\widetilde{k}\leq k\leq n$, is monotone non-decreasing, so that the minimum should be sought between the values $\underline{k}$ and $\widetilde{k}$. Similarly, the inequality $C_{\max}\left(S^*(\underline{k})\right)>C_{\max}\left(S^*(\widetilde{k})\right)$

implies that the minimum should be sought between the values $\widetilde{k}$ and $\overline{k}$. All together, Algorithm BinarySearch explores at most $\lceil \log_2 (K+1) \rceil$ values of k and the following statement holds.

Theorem 6.6. *Algorithm BinarySearch solves an instance of problem* $1 \left| p_j g\left(r\right) \text{-det}, MP\left[0\right] \right| C_{\max}$ *in* $O\left(n \log K\right)$ *time, provided that the LPT order of the jobs is known.*

The results of this section are published in our recent paper Rustogi and Strusevich (2012a), where we consider the situation in which $K = n-1$, so that the running time obtained is $O(n \log n)$.

6.8 Conclusion

In this chapter, we solve several problems with positional deterioration and maintenance activities. Before this study, only a handful of results existed which combined a study of positional deterioration with maintenance activities. The only paper we are aware of, which address this problem with job-independent positional effects is due to Kuo and Yang (2008a), who studied a special case of problem $1 \left| p_j g(r) \text{-det}, MP\left[0\right] \right| C_{\max}$ and propose an $O\left(nK\right)$ time algorithm to solve it. Our study has further expanded and generalised this area. Our main contributions can be summarised as follows:

- Use of a general function $g\left(r\right)$ to model a positional effect.
- Use of arbitrary possibly non-monotone positional factors in scheduling problems with no maintenance activities.
- Simultaneous use of distinct maintenance activities in a schedule.
- Introduction of group-dependent positional effects.

Each of the problems considered, reduces to a linear assignment problem with a product matrix, so that a solution is possible by applying Lemma 2.1. We propose three novel solution approaches that solve different versions of problem $1 \left| p_j g^{[x]}\left(r\right) \text{-det}, MP \right| C_{\max}$.The developed algorithms rely on Theorem 6.1, which in turn is based on Lemma 2.1. Table 6.4 summarises all the problems considered in this chapter along with the running times needed to solve them.

	Constant Duration MPs		Start-time dependent MPs	
	Identical	Distinct	Identical	Distinct
Group-indep	$O(n \log K)$	$O(n \log K)$	$O(nK)$	$O\left(nK2^K\right)$
Group-dep	$O(nK)$	$O\left(nK2^K\right)$	$O(nK^2)$	$O\left(nK^2 2^K\right)$

Table 6.4: Computational complexities of different versions of problem $1\left|p_j g^{[x]}(r)\text{-det}, MP\right| C_{\max}$, assuming that the LPT order of the jobs is known

CHAPTER 7

Job-Dependent Positional Effects and Rate-Modifying Activities

In this chapter, we discuss single machine scheduling problems with job-dependent positional effects and rate-modifying activities. This chapter can be seen as an extension of the previous chapter as the underlying effect is still positional, only a more general model is being considered. Similar to the last chapter, our main focus will be to explore models with deterioration effects and maintenance activities. In the following sections, we give a brief overview of the problem under consideration and provide a variety of solution approaches that efficiently solve different versions of this problem.

The results of this chapter are published in our recent paper Rustogi and Strusevich (2012a). In this paper, we only provide a simplified version of the problem, so that the main ideology behind the developed algorithms can be clearly understood. In this chapter, however, we provide a full account of the entire range of problems that can be solved using the developed solution approaches.

7.1 Overview of the Problems

As described in Section 3.2.1, under a job-dependent positional effect the actual processing time of job j scheduled in position r of a schedule is given by

$$p_j(r) = p_j g_j(r),\ 1 \leq r \leq n,$$

where $g_j(r)$, $j \in N$, is a job-dependent positional factor. Recall from Section 3.2.1, that if the values $g_j(r)$, $1 \leq r \leq n$, form a non-decreasing sequence (3.5) for each $j \in N$, we deal with a positional deterioration effect. Such a model represents a scenario in which each job wears out the machine in a different way, hence each job $j \in N$ is

associated with a unique set of positional factors. On the other hand, if the sequence is non-increasing (3.6) for each $j \in N$, a learning effect is observed. Unlike the case with job-independent positional effects, it is common in the scheduling literature on job-dependent effects, to study models in which an arbitrary, possibly non-monotone positional effect is considered, see, e.g., Mosheiov (2008) and Bachman and Janiak (2004).

As outlined in Chapter 4, consider a general situation, in which the decision-maker is presented with a total of $K \geq 0$ possible rate-modifying activities, which can be either distinct or alike. For each MP, it is exactly known how it affects the processing conditions of the machine, should the decision-maker decide to include it into a schedule.

If $k-1$ MPs are chosen from the available K options, then the jobs are divided into k, $1 \leq k \leq K+1$ groups. Depending on which MPs are chosen and the order in which they are performed, the actual processing time of a job $j \in N$, scheduled in position r of the x-th group can be given by

$$p_j^{[x]}(r) = p_j g_j^{[x]}(r), \ 1 \leq r \leq n, \ 1 \leq x \leq k, \tag{7.1}$$

where $g_j^{[x]}(r)$ is a *job-dependent group-dependent* positional factor. This is the most general positional factor known, as it allows a three way dependency, namely on job, group and position. Recall from Chapter 6, that a group-dependent positional factor enables us to handle situations in which each MP can have a different effect on the machine conditions.

If a pure deterioration model is considered, then the positional factors within a group x are in non-decreasing order

$$1 \leq g_j^{[x]}(1) \leq g_j^{[x]}(2) \leq \cdots \leq g_j^{[x]}(n), \ 1 \leq x \leq k, \ j \in N, \tag{7.2}$$

whereas if a pure learning model is considered, then the positional factors within a group x are in non-increasing order

$$1 \geq g_j^{[x]}(1) \geq g_j^{[x]}(2) \geq \cdots \geq g_j^{[x]}(n), \ 1 \leq x \leq k, \ j \in N. \tag{7.3}$$

In the former case, the RMPs are essentially maintenance periods (MPs), which must be included in the schedule in order to negate the deteriorating machine conditions; see, e.g., Yang and Yang (2010a). Recall from Chapter 4, that if an RMP is a maintenance period, then the value of $\alpha^{[y]}$ is non-negative. In the case with learning

effects, the RMPs can either be associated with replacing a machine/operator or be associated with an activity which further enhances the learning rate of the machine, see, e.g., Ji and Cheng (2010). As discussed earlier, it is also possible to have an arbitrary non-monotone positional effect, which could possibly arise due to a combination of deterioration and learning effects. In such a case, no restrictions are imposed on the type of the RMPs to include in a schedule.

The problem of minimising a certain objective function F, under the general settings defined by (7.1) and (4.1) can be denoted by $1\left|p_j g_j^{[x]}(r), RMP\right| F$. In this chapter, we only consider the problem of minimising the makespan, i.e., $F = C_{\max}$, for the case of job-dependent positional deterioration. We denote the resulting problem as $1\left|p_j g_j^{[x]}(r)\text{-det}, MP\right| C_{\max}$, where the first term in the middle field represents the presence of job-dependent positional factors which follow (7.2), and the second term points out that the RMPs are essentially maintenance periods (MPs).

An optimal solution to problem $1\left|p_j g_j^{[x]}(r)\text{-det}, MP\right| C_{\max}$ must deliver optimal choices for each of the Decisions 1-4 defined in Chapter 4. In what follows, we consider various versions of problem $1\left|p_j g_j^{[x]}(r)\text{-det}, MP\right| C_{\max}$ and provide polynomial time algorithms to solve them. Similar to Chapter 6, we differentiate between different versions based on three criteria: (i) deterioration factors are group-dependent or group-independent, (ii) MPs are identical or distinct, and (iii) duration of the MPs are constant or start-time dependent. All versions of problem $1\left|p_j g_j^{[x]}(r)\text{-det}, MP\right| C_{\max}$ reduce to a linear assignment problem in its full form.

Notice that if the objective function is to minimise the total flow time, i.e., $F = \sum C_j$, it is found that irrespective of the ordering of the positional factors within a group, no version of problem $1\left|p_j g_j^{[x]}(r), RMP\right| \sum C_j$ can be solved in less than $O\left(n^{K+3}\log n\right)$ time. A similar observation is made for the problem of minimising the makespan, under conditions in which the positional factors are non-monotonically ordered within a group. A solution to both of these problems is obtained by solving $O\left(n^K\right)$ instances of a full form LAP given by (2.2). We re-visit both these problems in Chapter 9.

7.2 Computing Positional Weights

To solve problem $1\left|p_j g_j^{[x]}(r)\text{-det}, MP\right| C_{\max}$, we first assume that Decisions 1-3 are taken in advance, so that we know that a total of $k-1$ MPs have been included in the schedule. As a result the jobs are split into k, $1 \leq k \leq K+1$, groups. Renumber the

indices of duration parameters of the MPs, in order of their occurrence in the schedule, so that the duration of the MP scheduled after the x-th group is given by (6.4). Denote the resulting problem as $1\left|p_j g_j^{[x]}(r)\text{-det}, MP(k-1)\right|C_{\max}$.

To solve problem $1\left|p_j g_j^{[x]}(r)\text{-det}, MP(k-1)\right|C_{\max}$ consider a schedule $S(k)$ with a permutation of jobs $\pi = \left(\pi^{[1]}, \pi^{[2]}, \ldots, \pi^{[k]}\right)$. Assume that each group contains a total of $n^{[x]}$ jobs, so that $\pi^{[x]} = \left(\pi^{[x]}(1), \pi^{[x]}(2), \ldots, \pi^{[x]}\left(n^{[x]}\right)\right)$, $1 \leq x \leq k$, where $\sum_{x=1}^{k} n^{[x]} = n$. The actual processing time of a job $j = \pi^{[x]}(r)$, scheduled in position r, $1 \leq r \leq n^{[x]}$, of the x-th group, $1 \leq x \leq k$, is given by (7.1). Similar to the computation of the makespan for problem $1\left|p_j g^{[x]}(r)\text{-det}, MP(k-1)\right|C_{\max}$ as given in (6.7), the makespan for problem $1\left|p_j g_j^{[x]}(r)\text{-det}, MP(k-1)\right|C_{\max}$ can be given as

$$C_{\max}(S(k)) = \sum_{x=1}^{k-1}\sum_{r=1}^{n^{[x]}}(1+\alpha^{[x]})g_j^{[x]}(r)\,p_{\pi^{[x]}(r)} + \sum_{r=1}^{n^{[k]}} g_j^{[k]}(r)\,p_{\pi^{[k]}(r)} + \sum_{x=1}^{k-1}\beta^{[x]}.$$

The above objective function can be written as generic function given by (4.4), with the job-dependent positional weights

$$W_j^{[x]}(r) = \begin{cases} (1+\alpha^{[x]})g_j^{[x]}(r), & 1 \leq r \leq n^{[x]},\ 1 \leq x \leq k-1, \\ g_j^{[x]}(r) & 1 \leq r \leq n^{[x]},\ x = k, \end{cases} \tag{7.4}$$

and the constant term still given by (6.9).

As shown in Section 4.2.2, if the number of jobs in each group, $n^{[x]}$, $1 \leq x \leq k$, is known in advance, the problem of minimising the generic objective function (4.4), can be solved by reduction to a linear assignment problem (LAP) of the form (2.2), with its cost function $c_{j,(x,r)}$ given as (4.5). To find an optimal solution to problem $1\left|p_j g_j^{[x]}(r)\text{-det}, MP(k-1)\right|C_{\max}$, however, we must find a way of determining the optimal number of jobs $n^{[x]}$, that are scheduled in each group x, $1 \leq x \leq k$, before we go on to solve the LAP. Further, to obtain a solution to the original problem $1\left|p_j g_j^{[x]}(r)\text{-det}, MP\right|C_{\max}$, we solve all possible instances of problem $1\left|p_j g_j^{[x]}(r)\text{-det}, MP(k-1)\right|C_{\max}$ by modifying the outcomes of Decisions 1-3 and choose the instance with the smallest value of the objective function as our optimal solution.

Similar to Chapter 6, we shall consider eight different versions of problem $1\left|p_j g_j^{[x]}(r)\text{-det}, MP\right|C_{\max}$. For each version, the computed positional weights are found by making an appropriate substitution in (7.4). Recall that (7.4) is computed for the

most general version, in which the deterioration factors are group-dependent, MPs are distinct and their duration depends on the start-time. Depending on the found positional weights, it might be possible to solve problem $1\left|p_j g_j^{[x]}(r)\text{-det}, MP(k-1)\right|C_{\max}$ by reducing it to a revised version of the LAP, one in which the optimal values of $n^{[x]}$, $1 \leq x \leq k$, can be found on the fly. Moreover, for some versions of problem $1\left|p_j g_j^{[x]}(r)\text{-det}, MP\right|C_{\max}$, it is even possible to compute the optimal outcomes of Decisions 1-3 on the fly. Based on such differences, we classify different versions of problem $1\left|p_j g_j^{[x]}(r)\text{-det}, MP\right|C_{\max}$ into two separate sections and present two new solution approaches that handle them. Table 7.1 lists out all the versions under consideration and mentions the section that deals with them.

	Constant Duration MPs		Start-time dependent MPs	
	Identical	Distinct	Identical	Distinct
Group-indep	Section 7.4	Section 7.4	Section 7.4	Section 7.3
Group-dep	Section 7.4	Section 7.3	Section 7.3	Section 7.3

Table 7.1: Different versions of problem $1\left|p_j g_j^{[x]}(r)\text{-det}, MP\right|C_{\max}$

7.3 Solution Approach PosiJDGD

In this section, we describe a solution approach, which can solve all versions of problem $1\left|p_j g_j^{[x]}(r)\text{-det}, MP\right|C_{\max}$ without prior knowledge of the number of jobs $n^{[x]}$ in each group. The only pre-requisite condition is that the computed positional weights $W_j^{[x]}(r)$, $1 \leq r \leq n^{[x]}$, should be non-decreasing in every group x, $1 \leq x \leq k$, for each $j \in N$.

Let us begin our consideration with the most general version of problem $1\left|p_j g_j^{[x]}(r)\text{-det}, MP\right|C_{\max}$, such as the one discussed in Section 7.2. Assume that Decisions 1-3 are taken in advance and denote the resulting problem as $1\left|p_j g_j^{[x]}(r)\text{-det}, MP(k-1)\right|C_{\max}$. This problem can be solved by minimising the generic objective function (4.4) with positional weights given by (7.4) and the constant term given by (6.9). It is easy to notice that because the positional factors $g_j^{[x]}(r)$ follow (7.2), the computed positional weights $W_j^{[x]}(r)$, $1 \leq r \leq n^{[x]}$, are non-decreasing within each group x, $1 \leq x \leq k$, for all values of $j \in N$. Below we outline a solution approach that solves an instance of problem $1\left|p_j g^{[x]}(r)\text{-det}, MP(k-1)\right|C_{\max}$.

Define a rectangular assignment problem with n rows, each corresponding to a job $j \in N$, and $m = nk$ columns. As done for the $n \times n$ LAP, number the columns by

a string of the form (x, r), where x refers to a group, $1 \leq x \leq k$, and r, $1 \leq r \leq n^{[x]}$, indicates a position within the group. Create an $n \times m$ cost matrix $C = \left(c_{j,(x,r)}\right)$ by setting $n^{[x]} = n$, $1 \leq x \leq k$, and computing all values of the cost function $c_{j,(x,r)}$, $1 \leq r \leq n$, $1 \leq x \leq k$, $j \in N$, by using the formulae (4.5) and (7.4). More precisely, the value of element $c_{j,(x,r)}$ at the intersection of the j-th row and v-th the column of matrix C for v, $1 \leq v \leq m$, such that $v = n\,(x-1) + r$, where $1 \leq x \leq k$ and $1 \leq r \leq n$, is defined by $c_{j,(x,r)} = p_j W_j^{[x]}(r)$. Notice that the matrix C represents a set of all possible values of $c_{j,(x,r)}$ and can be computed in $O\left(n^2 k\right)$ time.

As a result, problem of minimising the generic objective function (4.4) reduces to a rectangular assignment problem written out below

$$\begin{array}{ll} \text{Min} & \displaystyle\sum_{j \in N} \sum_{x=1}^{k} \sum_{r=1}^{l^{[x]}} c_{j,(x,r)} z_{j,(x,r)} \\ \text{subject to} & \displaystyle\sum_{j \in N} z_{j,(x,r)} \leq 1,\ 1 \leq x \leq k,\ 1 \leq r \leq l^{[x]} \\ & \displaystyle\sum_{x=1}^{k} \sum_{r=1}^{l^{[x]}} z_{j,(x,r)} = 1,\ j \in N \\ & z_{j,(x,r)} \in \{0,1\},\ j \in N,\ 1 \leq x \leq k,\ 1 \leq r \leq l^{[x]}, \end{array} \tag{7.5}$$

where in the case under consideration $l^{[x]} = n$ for $1 \leq x \leq k$.

The algorithm to solve a rectangular assignment problem of the form (7.5) has been outlined by Bourgeois and Lassale (1971). The running time of this algorithm is $O(n^2 m)$, $m \geq n$, for an $n \times m$ cost matrix. Thus, an optimal solution for problem (7.5) can be found in $O(n^3 k)$ time.

Suppose that for some k, $1 \leq k \leq K+1$, the solution of the assignment problem (7.5) related to problem $1 \left| p_j g_j^{[x]}(r)\text{-det}, MP\,(k-1) \right| C_{\max}$ is found. Then $z_{j,(x,r)} = 1$ implies that job j is assigned to the r-th position of group x. The conditions of (7.5) mean that each job will be assigned to a position and no position will be used more than once. The condition (7.2) guarantees that the found assignment admits a meaningful scheduling interpretation, because for each of the k groups either several consecutive positions starting from the first are filled or the group is not used at all. In principle, the same solution approach remains valid even for the case with non-monotone positional factors, since setting $n^{[x]} = n$, $1 \leq x \leq k$, does indeed generate a set of all possible cost functions $c_{j,(x,r)}$ for the latter case as well. However, since the condition (7.2) does not hold for the case with arbitrary positional effects, it cannot be guaranteed that consecutive positions (starting from the first position) are filled in each group, thereby resulting in an infeasible solution. To ensure feasibility of the obtained solution it is

essential that the obtained positional weights be monotonically ordered within a group.

Notice that in philosophy, this solution approach is similar to Solution Approach PosiJIJD provided in Section 6.5: n best positions need to be chosen from a set of nk positions. Algorithm NSmall however, permits a faster running time as the problem under consideration is job-independent and reduces to a special case of the LAP, so that Lemma 2.1 holds. The following statement holds.

Theorem 7.1. *Problem* $1\left|p_j g_j^{[x]}(r)\text{-}det, MP\,(k-1)\right|C_{\max}$ *can be solved in* $O(n^3k)$ *time by reduction to a linear rectangular assignment problem of the form (7.5).*

In an optimal solution for an instance of problem $1\left|p_j g_j^{[x]}(r)\text{-det}, MP\,(k-1)\right|C_{\max}$ it is possible that out of the k groups, certain groups are not assigned any jobs at all, i.e., $n^{[x]}=0$. Such a situation can occur if an MP is not efficient in restoring the machine to a better state, and as a result the group that follows generates positional weights with big values. Such an instance can never result in an optimal schedule for the general problem $1\left|p_j g_j^{[x]}\,(r)\,\text{-det}, MP\right|C_{\max}$, as we are unnecessarily spending time to perform an inefficient MP. This instance, if any, will be automatically eliminated from consideration if we try different combinations of Decision 1-3 to define problem $1\left|p_j g_j^{[x]}(r)\text{-det}, MP\,(k-1)\right|C_{\max}$.

To determine the optimal solution for the general problem $1\left|p_j g_j^{[x]}\,(r)\,\text{-det}, MP\right|C_{\max}$, all options associated with Decisions 1-3 must be enumerated and the solutions of the resulting sub-problems $1\left|p_j g_j^{[x]}\,(r)\,\text{-det}, MP\,(k-1)\right|C_{\max}$ be compared. The best of these solutions is chosen as the optimal solution for problem $1\left|p_j g_j^{[x]}\,(r)\,\text{-det}, MP\right|C_{\max}$. The same procedure holds for all other less general versions of problem $1\left|p_j g_j^{[x]}\,(r)\,\text{-det}, MP\right|C_{\max}$ as well. Table 7.2 states the number of times an LAP of the form (7.5) must be solved in order to solve different versions of problem $1\left|p_j g_j^{[x]}\,(r)\,\text{-det}, MP\right|C_{\max}$.

	Constant Duration MPs		Start-time dependent MPs	
	Identical	Distinct	Identical	Distinct
Group-indep	Section 7.4	Section 7.4	Section 7.4	$\sum_{k=1}^{K+1}\binom{K}{k-1}$
Group-dep	Section 7.4	$\sum_{k=1}^{K+1}\binom{K}{k-1}$	$\sum_{k=1}^{K+1}1$	$\sum_{k=1}^{K+1}\binom{K}{k-1}(k-1)$

Table 7.2: Number of times an LAP of the from (7.5) is solved to solve different versions of problem $1\left|p_j g_j^{[x]}\,(r)\,\text{-det}, MP\right|C_{\max}$.

Notice that although the above described solution approach of reducing the problem to an LAP of the form (7.5) is able to solve all eight versions of problem

$1\left|p_j g_j^{[x]}(r)\text{-det}, MP\right|C_{\max}$, for some cases it is possible to make Decisions 1-3 on the fly by using another solution approach, which enables the optimal solution to be found in faster time. For such cases, a reference to the relevant section has been made in Table 7.2. For all other cases, the number of instances of problem $1\left|p_j g_j^{[x]}(r)\text{-det}, MP(k-1)\right|C_{\max}$ can be computed similarly to the computation of the values of Table 6.2, which was done for case of job-independent positional deterioration in Chapter 6.

Since an LAP of the form (7.5) requires a running time of $O(n^3 k)$ for a given k, $1 \leq k \leq K+1$, the respective running times that are needed to solve different versions of problem $1\left|p_j g_j^{[x]}(r)\text{-det}, MP\right|C_{\max}$ can be computed as follows. For problem $1\left|p_j g_j^{[x]}(r)\text{-det}, MP\right|C_{\max}$ with group-dependent deterioration factors and distinct MPs with start-time dependent durations, an optimal solution can be found in $O\left(n^3 \sum_{k=1}^{K+1}\binom{K}{k-1}(k-1)\,k\right) = O\left(n^3 K^2 2^K\right)$ time. For the problem with group-independent deterioration factors and distinct MPs with constant durations, an optimal solution can be found in $O\left(n^3 \sum_{k=1}^{K+1}\binom{K}{k-1}k\right) = O\left(n^3 K 2^K\right)$ time. The same running time is needed for the problem with group-dependent deterioration rates and identical MPs with start-time dependent durations. Lastly, for problem $1\left|p_j g_j^{[x]}(r)\text{-det}, MP\right|C_{\max}$ with group-dependent deterioration rates and identical MPs with constant durations an optimal solution can be found in $O\left(n^3 \sum_{k=1}^{K+1} k\right) = O(n^3 K^2)$ time.

In our paper Rustogi and Strusevich (2012a), we use the above described solution approach to solve problem $1\left|p_j g_j^{[x]}(r)\text{-det}, MP\right|C_{\max}$ with group-dependent deterioration factors and distinct MPs with start-time dependent durations. However, no emphasis has been made on Decisions 2 and 3. The decision-maker only needs to decide how many MPs to include into the schedule from a given list of $n-1$ available MPs. The published running time to solve this problem is $O(n^5)$, which is consistent with the results of this section for $K = n-1$.

7.4 Solution Approach PosiJDKdomi

In this section, we consider versions of problem $1\left|p_j g_j^{[x]}(r)\text{-det}, MP\right|C_{\max}$ in which Decisions 1-4 can be made on the fly, without having to enumerate all possible options. We list out two versions for which this is possible:

Problem 1: Problem $1\left|p_j g_j^{[x]}(r)\text{-det}, MP\right|C_{\max}$ with group-dependent deterioration factors and distinct MPs with constant durations, subject to the condition that if

a certain MP with duration $\beta^{[y]}$ is included in the schedule, then the group that follows will contain the positional factors $g_j^{[y+1]}(r)$, $1 \leq r \leq n$, such that they can be ordered in a way that

$$g_j^{[1]}(r) \leq g_j^{[2]}(r) \leq \cdots \leq g_j^{[K+1]}(r),\ 1 \leq r \leq n,\ j \in N, \tag{7.6}$$

and (6.11) hold simultaneously, where the positional factors $g_j^{[1]}(r)$, $1 \leq r \leq n$, are associated with the group that is created before the first MP. This version is a generalisation of three of the cases found in Table 7.2; (i)-(ii) problem $1\left|p_j g_j^{[x]}(r)\text{-det}, MP\right|C_{\max}$ in which group-independent deterioration factors are considered along with identical or distinct MPs of constant duration, i.e., $\alpha^{[x]} = 0$, and (iii) problem $1\left|p_j g_j^{[x]}(r)\text{-det}, MP\right|C_{\max}$ in which group-dependent deterioration factors are considered along with identical MPs of constant duration, i.e., $\alpha^{[x]} = 0$, $\beta^{[x]} = \beta$, $1 \leq x \leq K$. For the latter problem, we assume that (7.6) holds, based on the argument provided in Section 6.6.

Problem 2: Problem $1\left|p_j g_j^{[x]}(r)\text{-det}, MP\right|C_{\max}$ with group-independent deterioration factors and distinct MPs with start-time dependent durations, subject to the condition that the duration parameters of the MPs can be ordered such that (6.12) and (6.11) hold simultaneously. This version is a generalisation of one of the cases found in Table 6.2, in which group-independent positional factors are considered along with identical MPs of start-time dependent duration.

In order to solve both versions of problem $1\left|p_j g_j^{[x]}(r)\text{-det}, MP\right|C_{\max}$ described above, the optimal choice for Decisions 2 and 3 can be made easily. If Decision 1 is assumed to be taken, so that $k-1$, $1 \leq k \leq K+1$, MPs are included in the schedule, then for both problems, the MPs with the indices $1, 2, \ldots, k-1$ are chosen and scheduled in the same order. This is the optimal choice for Problem 1 as the MPs with the indices $1, 2, \ldots, k-1$ have the smallest durations and create groups that contain the smallest deterioration factors, owing to (7.6) and (6.11) holding simultaneously. This is the optimal choice for Problem 2 as all MPs create identical groups and the ones with a smaller index have smaller values of the duration parameters, owing to (6.12) and (6.11) holding simultaneously. Notice that the order of the MPs is inconsequential in both cases.

With Decisions 1-3 having been made (with an assumed value of k), denote the resulting problem as $1\left|p_j g_j^{[x]}(r)\text{-det}, MP(k-1)\right|C_{\max}$. This problem can be solved by minimising the generic objective function of the form (4.4). For Problem 1, obtain the required positional weights $W_j^{[x]}(r)$ by substituting $\alpha^{[x]} = 0$, $1 \leq x \leq k$, in (7.4) so that

for each $j \in N$ we have

$$W_j^{[x]}(r) = g_j^{[x]}(r),\ 1 \leq r \leq n^{[x]},\ 1 \leq x \leq k, \tag{7.7}$$

and for Problem 2, substitute $g_j^{[x]}(r) = g_j(r)$, $1 \leq r \leq n^{[x]}$, $1 \leq x \leq k$, so that for each $j \in N$ we have

$$W_j^{[x]}(r) = \begin{cases} (1+\alpha^{[x]})g_j(r), & 1 \leq r \leq n^{[x]},\ 1 \leq x \leq k-1, \\ g_j(r) & 1 \leq r \leq n^{[x]},\ x = k. \end{cases} \tag{7.8}$$

Set the value $n^{[x]} = n$, $1 \leq x \leq k$, and $k = K+1$, and compute all positional weights $W_j^{[x]}(r)$, $1 \leq r \leq n$, $1 \leq x \leq K+1$, $j \in N$, for both problems by using the formulae above. Notice that the computed positional weights represent a set of all possible values of $W_j^{[x]}(r)$, $j \in N$, across all possible groups. Further, notice that because of (7.6), the positional weights associated with Problem 1 are ordered such for each k, $1 \leq k \leq K+1$, that we have

$$W_j^{[1]}(r) \leq W_j^{[2]}(r) \leq \cdots \leq W_j^{[k]}(r),\ 1 \leq r \leq n,\ j \in N,$$

and because of (6.12), the positional weights for Problem 2 are ordered such that for each k, $1 \leq k \leq K+1$, we have

$$W_j^{[k]}(r) \leq W_j^{[1]}(r) \leq W_j^{[2]}(r) \leq \cdots \leq W_j^{[k-1]}(r),\ 1 \leq r \leq n,\ j \in N.$$

Notice that both Problems 1 and 2 satisfy the conditions of '*K-domi*' as laid out in Definition 6.1, for each $j \in N$. For instances of problem $1\left|p_j g^{[x]}(r)\text{-det}, MP\right| C_{\max}$ that satisfy (6.11) and '*K-domi*', it is possible to compute the optimal values of $n^{[x]}$, $1 \leq x \leq k$, and Decisions 1-4 on the fly. Recall that for a fixed value of Decision 1, the optimal values for Decisions 2 and 3 are already known. Thus, to solve problem $1\left|p_j g_j^{[x]}(r)\text{-det}, MP\right| C_{\max}$ we only need to worry about finding the optimal values for Decisions 1 and 4.

Recall from Section 6.6 that under such conditions for a job-independent model, the relevant problem is solvable by Algorithm NSmall2, which finds an optimal schedule by computing the optimal values of $n^{[x]}$, $1 \leq x \leq k$, and Decisions 1 and 4 on the fly. This is achieved by seeking the n smallest deterioration factors that are needed to create an optimal schedule $S(k+1)$, only in positions that were used in the optimal schedule $S(k)$ and positions that become available with the introduction of the $(k+1)$-th group. This idea reduces the running time needed to compute the n smallest deterioration factors

as we only need to search a maximum of $2n$ candidate positions. We shall use the same philosophy to speed up the running times for Problems 1 and 2 in this chapter, however its validity for the job-dependent case is not as obvious.

Below we establish several properties of the algorithm that solves the rectangular assignment problem (7.5). These properties will help us to prove that for a given k, we can indeed limit the number of candidate positions to $2n$, even in the case of job-dependent positional effects. We start our consideration with Problem 1, and later adapt the developed solution approach for Problem 2.

For Problem 1, due to (7.2) and (7.6) holding simultaneously, the matrix C has a special structure that is characterised by:

- non-decreasing order of the elements of the same row that are placed in the columns associated with positions of the same group, and
- non-decreasing order of the elements placed in the same row and in those columns associated with a given position r, $1 \leq r \leq n$, of each group, from group 1 to group k.

Below we present a revised version of Algorithm BourLas (see Section 2.2.3) so that it reflects the special structure of our cost matrix C. Alterations are primarily made to Steps 1-3 of Algorithm BourLas and these alterations affect neither the optimality nor the running time of the algorithm.

Algorithm RecLAP (see Bourgeois and Lassale (1971))

Step 0. Consider a row of the matrix C, subtract the smallest element from each element in the row. Do the same for all other rows.

Step 1. Considering the rows in an arbitrary order, search for a zero, Z, in the current row that is located in the left-most column with no starred zeros. If Z is found, star Z. Repeat for each row of the matrix. Go to Step 2.

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

Step 3. Choose a non-covered zero and prime it; in the case of several available zeros prime the one in the left-most column. Consider the row containing the primed zero. If there is no starred zero in this row, go to Step 4. If there is a starred zero Z in this row, cover this row and uncover the column of Z. Repeat until all zeros are covered. Go to Step 5.

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

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

Below we analyse the outcome of an iteration of this algorithm.

Lemma 7.1. *Suppose that after some iteration of Algorithm RecLAP, for each x, $1 \leq x \leq k$, the column $\left(x, l^{[x]}\right)$ is such that it contains a 0^*, while none of the columns (x, r) for $r > l^{[x]}$ contain a 0^*. If no column (x, r) for $1 \leq r \leq n$ contains a 0^*, then define $l^{[x]} = 0$. Then, for each x, $1 \leq x \leq k$, such that $l^{[x]} \geq 1$, it follows that for each r, $1 \leq r \leq l^{[x]}$, column (x, r) contains a 0^*.*

Proof: Suppose that the lemma does not hold, i.e., for some x there exists a column (x, r') that does not contain a 0^*, where $r' < l^{[x]}$. Assume that a 0^* appears in position $\left(j, \left(x, l^{[x]}\right)\right)$. Since each covered line contains a 0^*, it follows that the column (x, r') is uncovered.

Observe that the only way for a zero to lose its "star" label is Step 4 of the algorithm, but in this case a $0'$ from the same column becomes a 0^*. In short, once a column gets a 0^*, then it will contain a 0^* (possibly, in a different row) in all subsequent iterations. On the other hand, if a column does not have a 0^*, then it has not contained a 0^* in all preceding iterations. Thus, column (x, r') has not contained a 0^* in all previous iterations, and this column has always been uncovered.

Suppose that in some iteration i, the element in position $\left(j, \left(x, l^{[x]}\right)\right)$ is reduced to zero as the current minimal element (see Step 5). At the time the element is uncovered, i.e., in all previous iterations column $\left(x, l^{[x]}\right)$ has not contained a 0^* and has not been covered, exactly as column (x, r'). Thus, up to the i-th iteration both columns (x, r') and $\left(x, l^{[x]}\right)$ have been subject to the same transformations in Step 5. In particular, the elements in positions $(j, (x, r'))$ and $\left(j, \left(x, l^{[x]}\right)\right)$ either have been left the same in all previous iterations with row j covered or have been reduced by the value of the current minimal element in each previous iteration when row j was not covered. Since originally $p_j g_j^{[x]}(r') \leq p_j g_j^{[x]}(l^{[x]})$, we deduce that in the beginning of iteration i the

element in position $(j,(x,r'))$ is less or equal to the element in position $\left(j,\left(x,l^{[x]}\right)\right)$. At the end of iteration i, we know that position $\left(j,\left(x,l^{[x]}\right)\right)$ becomes zero. Now since the element in position $(j,(x,r'))$ cannot be negative, we can deduce that it must be zero at the end of iteration i. Therefore, all elements in the consecutive positions $(j,(x,r')),(j,(x,r'+1)),\cdots,(j,\left(x,l^{[x]}\right))$ of row j are equal to zero. We know that Step 3 of Algorithm RecLAP processes the zero in position $(j,(x,r'))$ earlier than all other uncovered zeros in this row. Thus, the zero in position $(j,(x,r'))$ will be primed.

If there is no 0^* in row j, Step 4 of the algorithm will star the primed zero in position $(j,(x,r))$, as we know that column (x,r) does not contain any 0^* either.

If there is a 0^* in row j, then the corresponding column, say, column v, is covered. The algorithm in Step 3 will uncover column v and cover row j. If this uncovers a 0 in column v, say, in row $u \neq j$, then Step 4 of the algorithm will find a path that traverses through the three positions $(u,v),(j,v)$ and $(j,(x,r'))$, and redistribute the stars. As a result, a 0^* would appear in position $(j,(x,r'))$.

Now we consider the situation when there are no uncovered zeros in column v, row j is covered, the zero in position (j,v) is starred, the zero in position $(j,(x,r'))$ is primed and the zeros in positions $(j,(x,r'+1)),\cdots,(j,\left(x,l^{[x]}\right))$ have no labels. By the lemma conditions, we know that eventually the zero in position $(j,\left(x,l^{[x]}\right))$ becomes starred. In the iterations that follow iteration i, the only way to get a 0^* in row j in a position other than (j,v) is to start with some $0'$ in the uncovered part of the current matrix, and to find a path (as described in Step 4) that starts with the chosen $0'$ and finishes with the two positions (j,v) and $(j,(x,r'))$, that contain a 0^* and a $0'$, respectively. However, as a result of the corresponding redistributions of stars, a 0^* will appear in position $(j,(x,r'))$.

We have proved that once column (x,r') gets a 0^*, it will always contain a 0^* in all subsequent iterations. Hence, our assumption that for some x, there exists a column (x,r') that does not contain a 0^*, where $r' < l^{[x]}$, is false; thereby proving Lemma 7.1. □

Lemma 7.2. *Under the conditions of Lemma 7.1, while processing all uncovered zeros, the values $l^{[x]}$ for each x, $1 \leq x \leq k$, either remain the same or exactly one of them increases by 1 for every zero considered.*

Proof: Among all uncovered zeros, in Step 3 choose zero Z that appears in the earliest column, and prime it. If the row containing the primed zero contains a 0^*, then we cover that row and uncover the column, so that Z does not become a 0^* yet. Notice that as a result of this transformation, all zeros contained in the same row with Z

are covered, including those found in Step 5, and they will not be considered in this iteration of the algorithm. Thus, the values $l^{[x]}$ for each x, $1 \leq x \leq k$, remain the same for all remaining zeros in this row.

If Z does not have any 0^* in its row (see Step 4), a path is formed which is an alternating sequence of the primed and starred zeros that starts with the primed zero Z ends with another $0'$. In such a situation, all 0^*'s in the path are unstarred and each $0'$ is converted to a 0^*. Since the number of primed zeros in the path is always one greater than the number of starred zeros, it follows that once the swap is performed we have exactly one extra 0^* that will replace the last $0'$ in the path. This includes a situation which happens when for zero Z neither its row, nor its column contains a 0^*, so that the path simply consists of Z alone and Z itself becomes a 0^*. Thus, a new 0^* will appear in the column that has not had a starred zero earlier, while all other columns will maintain the number of contained 0^*'s. Suppose that the new 0^* appears in position (x, r) for some group x, $1 \leq x \leq k$, and $r \geq 1$. If $r = 1$, then the old value $l^{[x]} = 0$ grows by 1. Otherwise, we know from Lemma 7.1 that in a particular group x, all 0^*'s appear in consecutive columns $(x, 1), \ldots, \left(x, l^{[x]}\right)$. Since column (x, r) is the next to the column $\left(x, l^{[x]}\right)$, the value $l^{[x]}$ for group x grows by 1.

Whenever a new 0^* is added to the matrix, the columns containing the 0^*'s are covered and the remaining uncovered zeros are processed one by one in the same manner. □

Now consider the sub-problem $1 \left| p_j g_j^{[x]}(r)\text{-det}, MP(k-1) \right| C_{\max}$ of Problem 1, in which jobs from the set N are scheduled with $k-1$, $1 \leq k \leq K+1$, MPs. Due to condition '*K-domi*', without loss of generality it can be said that each of the k groups in an optimal schedule $S^*(k)$ is not empty. Let $l^{[x]}$ denote the number of positions used in a group x, $1 \leq x \leq k$, so that $\sum_{x=1}^{x=k} l^{[x]} = n$.

Next, find a schedule $\tilde{S}(k)$ that is optimal for an instance of problem $1 \left| p_j g_j^{[x]}(r)\text{-det}, MP(k-1) \right| C_{\max}$ with a set of jobs $\tilde{N} \subset N$. Let in schedule $\tilde{S}(k)$ the number of filled positions in a group x, $1 \leq x \leq k$, be denoted by $\tilde{l}^{[x]}$. Lemmas 7.1 and 7.2 immediately imply that $\tilde{l}^{[x]} \leq l^{[x]}$ for each x, $1 \leq x \leq k$.

For a set of jobs N, consider problem $1 \left| p_j g_j^{[x]}(r)\text{-det}, MP(k-1) \right| C_{\max}$ with k groups and another problem $1 \left| p_j g_j^{[x]}(r)\text{-det}, MP(k) \right| C_{\max}$ with $k+1$ groups. Let $S^*(k)$ and $S^*(k+1)$ be the corresponding optimal schedules. Suppose that $\tilde{N} \subset N$ is the subset of jobs that are assigned to the first k groups in schedule $S^*(k+1)$. Then, as observed above, none of these groups uses more positions in $S^*(k+1)$ than it does in $S^*(k)$. Given the fact, $\sum_{x=1}^{x=k} l^{[x]} = n$, this implies the following statement.

Theorem 7.2. *Let $S^*(k)$ be an optimal schedule for problem $1\left|p_j g_j^{[x]}(r)\text{-}det, MP(k-1)\right|C_{\max}$. In order to find schedule $S^*(k+1)$ that is optimal for problem $1\left|p_j g_j^{[x]}(r)\text{-}det, MP(k)\right|C_{\max}$, it is sufficient to use the n positions used in schedule $S^*(k)$ together with $n-(k+1)+1$ new positions that appear when the $(k+1)$-th group is introduced.*

Notice that it suffices to consider only $n-(k+1)+1$ positions in the $(k+1)$-th group, as due to condition '*K-domi*' it can be ensured that each of the k earlier groups will have at least 1 job scheduled in them. The algorithm below starts with finding the best schedule with one group, and having found the best schedule with k groups finds the best schedule with $k+1$ groups by solving an assignment problem with $O(n)$ columns and n rows. The columns to be used while solving the problem with $k+1$ groups are the n columns for which an assignment was found in the previous iteration and $n-(k+1)+1=n-k$ new columns corresponding to the new group $k+1$.

Algorithm BestLAP

Step 1. Find an optimal schedule $S(1)$ with no maintenance periods, in which all jobs are placed in group 1. This is done by solving the $n \times n$ assignment problem of the form (2.2) with

$$c_{jr} = p_j W_j^{[1]}(r) = p_j g_j^{[1]}(r),\ j \in N,\ 1 \le r \le n.$$

Compute $P(S^*(1))$ as the optimal value of the objective function in this assignment problem. Determine schedule $S^*(1)$ in which job j is processed in the r-th position of group 1 if and only if $z_{jr}=1$. Define $C_{\max}(S^*(1)) = P(S^*(1))$. Define $\Gamma(1):=0$, $l^{[1]} =: n$, $k:=1$ and $k':=K+1$.

Step 2. With the current value of k do

(a) Update $\Gamma(k+1) = \Gamma(k) + \beta^{[k]}$. Define $l^{[k+1]} := n-k$. Compute all values of the matrix C by (4.5) and (7.7) for columns $(1,1),\cdots\left(1,l^{[1]}\right),\cdots,(k,1),\cdots\left(k,l^{[k]}\right)$ and $(k+1,1),\cdots,\left(k+1,l^{[k+1]}\right)$. Run Algorithm RecLAP to solve the resulting $n \times (2n-k)$ rectangular assignment problem of the form (7.5) with the current values of $l^{[x]}$, $1 \le x \le k+1$.

(b) Compute $P(S^*(k+1))$ as the optimal value of the objective function in that assignment problem and $C_{\max}(S^*(k+1)) = P^*(S(k+1)) + \Gamma(k+1)$. If $P(S^*(k+1)) = P(S^*(k))$ then define $k':=k$ and break the loop by moving

to Step 3; otherwise, determine schedule $S^*(k+1)$ in which job j is processed in the r-th position of group x, $1 \leq x \leq k+1$, if and only if $z_{j,(x,r)} = 1$. For each group x, $1 \leq x \leq k+1$, determine the last filled position $l^{[x]}$.

(c) Update $k := k+1$. If $k < K+1$, repeat Step 2; otherwise go to Step 3.

Step 3. Find the value k^*, $1 \leq k^* \leq k'$, such that

$$C_{\max}(S(k^*)) = \min\left\{C_{\max}\left(S\left(k\right)\right) | 1 \leq k \leq k'\right\}.$$

The condition $P(S^*(k+1)) = P(S^*(k))$ is similar to the loop breaking condition applied in Algorithm NSmall2 in Section 6.6. This implies that the addition of the $(k+1)$-th group does not provide any positions better than those in schedule $S^*\left(k\right)$. Thus, the makespan cannot be reduced by running more MPs after the k'-th group is opened. This implies that no further values of k should be examined and the best schedule should be found from the set $\left\{S\left(k\right) | 1 \leq k \leq k'\right\}$. If the loop is not broken throughout the run of Algorithm BestLAP, the default value of k' is set to $K+1$. Thus, at most $K+1$ iterations on the value of k is needed and the following statement holds.

Theorem 7.3. *Algorithm BestLAP solves an instance of problem* $1\left|p_j g_j^{[x]}\left(r\right)\text{-}det, MP\right|C_{\max}$ *defined by Problems 1 and 2 in* $O(n^3K)$ *time.*

Proof: The correctness of Algorithm BestLAP is justified by Theorem 7.2. To estimate the running time, notice that in Step 1 an $n \times n$ assignment problem is solved in $O(n^3)$ time. For each value of k in Step 2, we solve a rectangular assignment problem which has n rows and $2n-k = O\left(n\right)$ columns, of which n columns, namely $\left(1,1\right),\cdots,\left(1,l^{[1]}\right),\cdots,\left(k,1\right),\cdots,\left(k,l^{[k]}\right)$ are brought forward from the previous iteration and the remaining $n-k$ columns correspond to the new group $k+1$. Algorithm RecLAP will require $O\left(n^3\right)$ time for each k, $1 \leq k \leq K$, so that the overall running time of Algorithm BestLAP is $O(n^3K)$. □

Notice that when Algorithm BestLAP is applied to solve a version of Problem 1 with group-independent positional factors, i.e., $g_j^{[x]}\left(r\right) = g_j\left(r\right)$, then Algorithm BestLAP essentially behaves as the algorithm provided by Zhao and Tang (2010), who use the group balancing principle to solve the special case $1\left|p_j r^{a_j}\text{-det}, MP\left[0\right]\right|C_{\max}$ and provide an $O(n^3K)$ algorithm, where $K = n-1$; see Section 3.3.1 for details.

Let us now consider Problem 2 and see how Algorithm BestLAP can be adapted to solve it. Recall that the computed positional weights for the sub-problem $1\left|p_j g_j^{[x]}\left(r\right)\text{-det}, MP\left(k-1\right)\right|C_{\max}$ are given by (7.8). Suppose that for some value

of k, $1 \leq k \leq K+1$, we have found an optimal schedule $S^*(k)$ for problem $1\left|p_j g_j^{[x]}(r)\text{-det}, MP(k-1)\right| C_{\max}$, so that in each group x, $1 \leq x \leq k$, the number of consecutively used positions is $l^{[x]}$, where $\sum_{x=1}^{k} l^{[x]} = n$. While making a transition to solving the assignment problem associated with problem $1\left|p_j g_j^{[x]}(r)\text{-det}, MP(k)\right| C_{\max}$ with $k+1$ groups, notice that the deterioration factors used in problem $1\left|p_j g_j^{[x]}(r)\text{-det}, MP(k-1)\right| C_{\max}$ for groups from 1 up to $k-1$ remain the same, while the factors previously used in group k will now be used in group $k+1$. Additionally, for group k the previously used factors will be multiplied by $\left(1+\alpha^{[k]}\right)$. By Theorem 7.2, this means that in an optimal schedule $S^*(k+1)$ at most $l^{[x]}$ positions will be used in each group x, $1 \leq x \leq k-1$, and at most $l^{[k]}$ positions will be used in group $k+1$; i.e., at most n positions in total can be used in these groups. Additionally, up to $n-k$ positions can be used in group k. This implies that we can adapt Algorithm BestLAP for solving Problem 2, so that the problem can be solved in $O(n^3K)$ time.

A special case of Problem 2 has been considered by Yang and Yang (2010a), with a polynomial group-independent deterioration effect given by $g_j(r) = r^{a_j}$, $a_j > 0$, $1 \leq r \leq n$, $j \in N$, and identical start time dependent MPs, i.e., $\alpha^{[x]} = \alpha$, $\beta^{[x]} = \beta$, $1 \leq x \leq K+1$. They reduce the problem to a series of square assignment problems and propose an algorithm that requires $O(n^5)$ time for $K = n-1$. For a schedule $S(k)$ with k, $1 \leq k \leq K+1$, groups, recall that the positional weights are given by (7.8). For a case in which $\alpha^{[x]} = \alpha$, $\beta^{[x]} = \beta$, $1 \leq x \leq K+1$, notice that the first $k-1$ groups have identical positional factors for each job $j \in N$. Due to the indistinguishability of the first $k-1$ groups, the authors prove a group-balance principle which allows them to predict the number of jobs in each of the first $k-1$ groups, for an optimal schedule. For the k-th group, which is differently structured, they try all possible values of the number of jobs in that group. This leads to a trial of at most n instances. For each instance, the values $n^{[x]}$, $1 \leq x \leq k$, are known so that $\sum_{x=1}^{k} n^{[x]} = n$. As a result, for a known k, $1 \leq k \leq n$, the problem reduces to solving up to n assignment problems with an $n \times n$ cost matrix. This leads to an overall running time of $O(n^5)$ for the entire problem. Our solution approach based on Algorithm BestLAP on the other hand, solves a more general problem in $O(n^3K)$ time, where $K = n-1$.

Now consider a version of problem $1\left|p_j g_j^{[x]}(r)\text{-det}, MP\right| C_{\max}$ in which the conditions '*K-domi*' and (6.11) do not hold simultaneously. In principle, Algorithm BestLAP can still be used to obtain a solution for Decisions 1 and 4, but to make Decisions 2 and 3, a full enumeration of options might be required. As a result, the overall running time to solve problem $1\left|p_j g_j^{[x]}(r)\text{-det}, MP\right| C_{\max}$ turns out to be no smaller than that

obtained by using Solution Approach PosiJDGD presented in Section 7.3. In our paper Rustogi and Strusevich (2012a), we have used Algorithm BestLAP to solve a version of problem $1\left|p_j g_j^{[x]}(r)\text{-det}, MP\right| C_{\max}$, in which '*K-domi*' holds, but (6.11) doesn't. Additionally, we have assumed that for each k, $1 \leq k \leq K+1$, Decisions 2 and 3 are fixed. As a result, an optimal solution is obtained in $O(n^3K)$ time, where $K = n-1$.

Below we provide a numerical example that illustrates the working of Algorithm BestLAP for Problem 1.

Example 7.1. Consider a version of problem1 $\left|p_j g^{[x]}(r)\text{-det}, MP\right| C_{\max}$ as defined in Problem 1, with five jobs. The decision-maker has a choice of $K = 4$ MPs, whose durations are given as

$$\beta^{[1]} = 3,\ \beta^{[2]} = 3,\ \beta^{[3]} = 3.5,\ \beta^{[4]} = 4.$$

As a result, the jobs can be split in up to five groups. Table 7.3 presents the actual processing times of jobs from the set $N = \{1, 2, 3, 4, 5\}$, with each row containing the values $p_j^{[x]}(r) = p_j g_j^{[x]}(r)$ for a job j when placed in position r of group x. Notice that for each job the values of $p_j^{[x]}(r)$ do not decrease within each group, and do not decrease for each position r as the number x of a group grows. This is consistent with (7.2) and (7.6). Besides, a group x, $1 \leq x \leq K+1$, has only $n-x+1$ positions associated with it, as it is known that each of the $x-1$ earlier groups will have at least 1 job scheduled in them.

	Group 1					Group 2			
j	$p_j^{[1]}(1)$	$p_j^{[1]}(2)$	$p_j^{[1]}(3)$	$p_j^{[1]}(4)$	$p_j^{[1]}(5)$	$p_j^{[2]}(1)$	$p_j^{[2]}(2)$	$p_j^{[2]}(3)$	$p_j^{[2]}(4)$
1	5	6	7	8	9	6	6	8	9
2	10	12	13	15	16	11	12	14	15
3	1	1	2	2	3	1	2	3	4
4	3	5	7	8	9	4	5	7	9
5	7	7	7	8	8	7	7	8	8

	Group 3			Group 4		Group 5
j	$p_j^{[3]}(1)$	$p_j^{[3]}(2)$	$p_j^{[3]}(3)$	$p_j^{[4]}(1)$	$p_j^{[4]}(2)$	$p_j^{[5]}(1)$
1	6	7	8	7	8	7
2	12	13	14	12	14	13
3	2	2	3	3	4	4
4	4	6	8	5	7	5
5	7	7	8	8	8	8

Table 7.3: Actual processing times for Example 7.1

Table 7.4 shows the details of the run of Algorithm BestLAP for the above instance.

The run starts with $k = 1$, i.e., we solve the 5×5 assignment problem. The optimal solution determines a schedule $S^*(1)$ with no MPs, in which the jobs are assigned to the positions marked by the boxes. Since all positions of Group 1 are used in this schedule, the list of possible positions in the next iteration $k = 2$ includes all five positions of Group 1 and all four positions of Group 2. We solve the corresponding rectangular assignment problem, and the numbers marked with boxes determine a schedule $S^*(2)$ in which the jobs $2, 3$ and 5 occupy the first three positions of Group 1, respectively, while the jobs 4 and 1 are respectively assigned to the first two positions of Group 2, after the MP of duration 3. The positions that are not used are crossed out; they will never be used in subsequent iterations. In the next iteration $k = 3$ we use the positions associated with schedule $S^*(2)$ and three positions of the new Group 3. The method stops here, since none of the positions of Group 3 is filled, i.e., $P(S^*(3)) = P(S^*(2))$ as in the loop-breaking rule. Schedules $S^*(1)$ and $S^*(2)$ are the two candidates for a global optimal solution, and we choose $S^*(2)$ with the smaller makespan.

7.5 Conclusion

In this chapter, we solve several problems with job-dependent positional deterioration and maintenance activities. Before this study, only a handful of results existed which combined a study of positional deterioration with maintenance activities. The only two papers we are aware of, which address this problem with job-dependent positional effects are due to Zhao and Tang (2010) and Yang and Yang (2010a). We further extend the models considered by them by introducing group-dependent effects along with distinct MPs, and provide two novel solution approaches that solve the problem of minimising the makespan. The developed algorithms are based on solving a series of rectangular assignment problems and either match or improve upon the running times of existing algorithms, used for solving less general problems. By nature, the two solution approaches that solve different versions of problem $1 \left| p_j g_j^{[x]}(r)\text{-det}, MP \right| C_{\max}$ are similar to the ones that solve the corresponding versions of problem $1 \left| p_j g^{[x]}(r)\text{-det}, MP \right| C_{\max}$, discussed in Chapter 6.

Notice that unlike the problem $|p_j g(r)\text{-det}, MP[0]|\, C_{\max}$, with job-independent group-independent positional effects considered in Section 6.7, we are not able to establish any sort of a convexity that allows us to explore only $\lceil \log_2(K+1) \rceil$ values of k, for problem $1 |p_j g_j(r)\text{-det}, MP[0]|\, C_{\max}$ with job-dependent group-independent positional effects. Repeated numerical experiments suggest that it is possible that the sequence

$k=1$	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)
1	5	6	$\boxed{7}$	8	9
2	$\boxed{10}$	12	13	15	16
3	1	1	2	$\boxed{2}$	3
4	3	$\boxed{5}$	7	8	9
5	7	7	7	8	$\boxed{8}$

$P(S^*(1)) = 7 + 10 + 2 + 5 + 8 = 32$
$C_{\max}(S^*(1)) = P(S^*(1)) = 32$

$k=2$	(1,1)	(1,2)	(1,3)	~~(1,4)~~	~~(1,5)~~	(2,1)	(2,2)	~~(2,3)~~	~~(2,4)~~
1	5	6	7	8	9	6	$\boxed{6}$	8	9
2	$\boxed{10}$	12	13	15	16	11	12	14	15
3	1	$\boxed{1}$	2	2	3	1	2	3	4
4	3	5	7	8	9	$\boxed{4}$	5	7	9
5	7	7	$\boxed{7}$	8	8	7	7	8	8

$P(S^*(2)) = 6 + 10 + 1 + 4 + 7 = 28$
$C_{\max}(S^*(2)) = P(S^*(2)) + \beta^{[1]} = 28 + 3 = 31$

$k=3$	(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	(3,1)	(3,2)	(3,3)
1	5	6	7	6	$\boxed{6}$	6	7	8
2	$\boxed{10}$	12	13	11	12	12	13	14
3	1	$\boxed{1}$	2	1	2	2	2	3
4	3	5	7	$\boxed{4}$	5	4	6	8
5	7	7	$\boxed{7}$	7	7	7	7	8

$P(S^*(3)) = 6 + 10 + 1 + 4 + 7 = 28$
$C_{\max}(S^*(3)) = P(S^*(3)) + \beta^{[1]} + \beta^{[2]} = 28 + 3 + 3 = 34$

Table 7.4: Run of Algorithm BestLAP for Example 7.1

$C_{\max}(S^*(k))$, $1 \leq k \leq K+1$, is indeed convex for the latter problem. However, at present we are not able to provide a mathematical proof of this fact. If the sequence $C_{\max}(S^*(k))$, $1 \leq k \leq K+1$, is proved to be convex, we will be able to solve the problem $|p_j g_j(r)\text{-det}, MP[0]|\, C_{\max}$ in $O(n^3 \log K)$ time, instead of the current running time of $O(n^3 K)$ by Algorithm BestLAP.

Table 7.5 summarises all the problems considered in this chapter along with the running times needed to solve them.

	Constant Duration MPs		Start-time dependent MPs	
	Identical	Distinct	Identical	Distinct
Group-indep	$O(n^3K)$	$O(n^3K)$	$O(n^3K)$	$O\left(n^3K2^K\right)$
Group-dep	$O(n^3K)$	$O\left(n^3K2^K\right)$	$O(n^3K^2)$	$O\left(n^3K^22^K\right)$

Table 7.5: Computational complexities of different versions of problem $1\left|p_j g_j^{[x]}(r)\text{-det}, MP\right| C_{\max}$.

CHAPTER 8

Single Machine Scheduling under Time-Dependent Effects and Rate-Modifying Activities

In this chapter, we discuss single machine scheduling problems with time-dependent effects and rate-modifying activities. Similar to previous chapters, our main focus will be to explore models with deterioration effects and maintenance activities. We adapt the solution approaches provided in Chapter 6 to solve different versions of this problem.

The results of this chapter are published in our recent paper Rustogi and Strusevich (2013c). The presentation of content in the paper is very similar to the content provided in this chapter.

8.1 Overview of the Problems

As described in Section 3.2.2, there are many models that have been studied in the past, which describe a time-dependent effect. Among them, a popular time-dependent model of the form (3.15), is a linear function of the start-time of a job, so that the actual processing time of a job $j \in N$ starting at a time $\tau \geq 0$ is given by

$$p_j(\tau) = p_j + a\tau,$$

where a is a job-independent constant, which is strictly positive for the deterioration environment and strictly negative for the learning environment. Cheng, Ding and Lin (2004) mention in their influential survey that the above model is a very realistic setting, particularly in the case of scheduling problems with deteriorating machine,

when all processing times are increased by a common factor caused by the machine.

In this chapter, we mainly concentrate on time-dependent models of the above form and use it to study deterioration effects, so that $a > 0$. Similar to previous chapters, our prime focus will be to combine such deterioration effects with maintenance activities, so that the processing times can be prevented from becoming unacceptably large.

In the past, although many papers have considered the problem of time-dependent deterioration with a maintenance period, most of them however, only see the MP as a fixed non-availability period, which does not necessarily improve the machine conditions. Only a handful of studies have considered problems in which an MP is actually used to restore the machine to its original state, so that the effects of time-dependent deterioration are negated. These include the papers by Lodree and Geiger (2010) and Yang (2012). Lodree and Geiger (2010) consider a problem with time-dependent deterioration, in which the actual processing time of a job $j \in N$ starting at a time $\tau > 0$ is given by $p_j(\tau) = a_j\tau$. They solve the problem of minimising the makespan, provided that a single MP is included in the schedule. Yang (2012) considers the problem in which a time-dependent effect given by (3.15), is combined with a positional polynomial learning effect, so that the actual processing time of a job $j \in N$ starting at a time $\tau \geq 0$ and scheduled in position r is given by $p_j(\tau, r) = (p_j + a\tau)\, r^a$, $a < 0$, $1 \leq r \leq n$. The author solves the problems of minimising the makespan and the total flow time with up to K MPs in the schedule. The required running time for each problem is $O\left(n^{K+1} \log n\right)$. This fairly high running time is attributed to the consideration of combined effects of time-dependent deterioration and position-dependent learning. No comment has been made on what happens if a pure time-dependent deterioration model is considered with maintenance activities.

In this chapter, we solve the problem of minimising the makespan for a pure time-dependent deterioration effect given by (3.15), and explore the effect of including different kinds of maintenance activities in the schedule. As outlined in Chapter 4, consider a general situation, in which the decision-maker is presented with a total of $K \geq 0$ possible rate-modifying activities, which can be either distinct or alike. We further generalise the model (3.15) by allowing each MP to restore the machine to a different state, so that the deterioration rate in every group is different. Let us assume that an MP with an index y creates a group which has a deterioration rate of $a^{[y+1]} > 0$, $1 \leq y \leq K$. Thus, in a schedule $S(k)$ with k groups, depending on which $k-1$, $1 \leq k \leq K+1$, MPs are chosen and the order in which they are performed, the actual processing time of a job $j \in N$, starting at time τ of the x-th group is be given by

$$p_j^{[x]}(\tau) = p_j + a^{[x]}\tau,\ \tau \geq 0,\ 1 \leq x \leq k, \tag{8.1}$$

where it is assumed that the timer is reset after every MP. The deterioration rate $a^{[1]}$ corresponds to the group which is created before the first MP, and the other deterioration rates are renumbered in order of their occurrence in the schedule. Notice that these 'group-dependent' deterioration rates are analogous to the group-dependent positional factors of Chapter 6.

The problem of minimising the makespan under the general settings defined by (8.1) and (4.1) can be denoted by $1\left|p_j + a^{[x]}\tau, MP\right|C_{\max}$. An optimal solution to problem $1\left|p_j + a^{[x]}\tau, MP\right|C_{\max}$ must deliver optimal choices for each of the Decisions 1-4 defined in Chapter 4. In what follows, we consider various versions of problem $1\left|p_j + a^{[x]}\tau, MP\right|C_{\max}$ and provide polynomial time algorithms to solve them. Similar to Chapter 6, we differentiate between different versions based on three criteria: (i) deterioration rates are group-dependent or group-independent, (ii) MPs are identical or distinct, and (iii) duration of the MPs are constant or start-time dependent. All versions of problem $1\left|p_j + a^{[x]}\tau, MP\right|C_{\max}$ reduce to a special case of the linear assignment problem with a product matrix and can be solved using the idea presented in Lemma 2.1.

8.2 Computing Positional Weights

To solve problem $1\left|p_j + a^{[x]}\tau, MP\right|C_{\max}$, we first assume that Decisions 1-3 are taken in advance, so that we know that a total of $k-1$ MPs have been included in the schedule. As a result the jobs are split into k, $1 \leq k \leq K+1$, groups. Renumber the indices of duration parameters of the MPs, in order of their occurrence in the schedule, so that the duration of the MP scheduled after the x-th group is given by (6.4). Denote the resulting problem as $1\left|p_j + a^{[x]}\tau, MP\left(k-1\right)\right|C_{\max}$.

To solve problem $1\left|p_j + a^{[x]}\tau, MP\left(k-1\right)\right|C_{\max}$ consider a schedule $S\left(k\right)$ with a permutation of jobs $\pi = \left(\pi^{[1]}, \pi^{[2]}, \ldots, \pi^{[k]}\right)$. Assume that each group contains a total of $n^{[x]}$ jobs, so that $\pi^{[x]} = \left(\pi^{[x]}\left(1\right), \pi^{[x]}\left(2\right), \ldots, \pi^{[x]}\left(n^{[x]}\right)\right)$, $1 \leq x \leq k$, where $\sum_{x=1}^{k} n^{[x]} = n$. We now derive an expression for the total time it takes to process all jobs in a group x, $1 \leq x \leq k$.

Let us denote the total duration of the first r jobs in a group x by $F_{(x,r)}$. It follows

from (8.1) that

$$\begin{aligned}
F_{(x,1)} &= p_{\pi^{[x]}(1)} \\
F_{(x,2)} &= F_{(x,1)} + \left(p_{\pi^{[x]}(2)} + a^{[x]} F_{(x,1)}\right) = \left(a^{[x]}+1\right) p_{\pi^{[x]}(1)} + p_{\pi^{[x]}(2)} \\
F_{(x,3)} &= F_{(x,1)} + \left(p_{\pi^{[x]}(3)} + a^{[x]} F_{(x,2)}\right) = \left(a^{[x]}+1\right)\left(\left(a^{[x]}+1\right) p_{\pi^{[x]}(1)} + p_{\pi^{[x]}(2)}\right) + p_{\pi^{[x]}(3)} \\
&= \left(a^{[x]}+1\right)^2 p_{\pi^{[x]}(1)} + \left(a^{[x]}+1\right) p_{\pi^{[x]}(2)} + p_{\pi^{[x]}(3)} \\
&\vdots \\
F_{(x,r)} &= \sum_{u=1}^{r} \left(a^{[x]}+1\right)^{r-u} p_{\pi^{[x]}(u)},\ 1 \le r \le n^{[x]},\ 1 \le x \le k.
\end{aligned}$$

Thus, the total time it takes to process all jobs in a group x can be given by

$$F_x = F_{\left(x,n^{[x]}\right)} \sum_{r=1}^{n^{[x]}} \left(a^{[x]}+1\right)^{n^{[x]}-r} p_{\pi^{[x]}(r)},\ 1 \le x \le k. \tag{8.2}$$

The makespan of a schedule $S(k)$ is given by

$$C_{\max}(S(k)) = F_1 + T_1 + F_2 + T_2 + \cdots + F_{k-1} + T_{k-1} + T_k,$$

where T_x is the duration of the MP scheduled after the x-th group. Substituting the value of T_x from (6.4) in the above equation we get

$$C_{\max}(S(k)) = \sum_{x=1}^{k-1}(1+\alpha^{[x]})F_x + F_k + \sum_{x=1}^{k-1}\beta^{[x]}.$$

Now, substituting the value of F_x from (8.2) we get

$$C_{\max}(S(k)) = \sum_{x=1}^{k-1}\sum_{r=1}^{n^{[x]}}(1+\alpha^{[x]})\left(a^{[x]}+1\right)^{n^{[x]}-r} p_{\pi^{[x]}(r)} + \sum_{r=1}^{n^{[k]}}\left(a^{[k]}+1\right)^{n^{[k]}-r} p_{\pi^{[k]}(r)} + \sum_{x=1}^{k-1}\beta^{[x]}. \tag{8.3}$$

Notice that the above objective function can be written as the generic function (4.6) introduced in Chapter 6, with the positional weights

$$W^{[x]}(r) = \begin{cases} (1+\alpha^{[x]})\left(a^{[x]}+1\right)^{n^{[x]}-r}, & 1 \le r \le n^{[x]},\ 1 \le x \le k-1, \\ \left(a^{[k]}+1\right)^{n^{[k]}-r} & 1 \le r \le n^{[x]},\ x = k, \end{cases} \tag{8.4}$$

and the constant term still given by (6.9).

Thus, problem $1\left|p_j + a^{[x]}\tau, MP(k-1)\right| C_{\max}$ reduces to minimising a linear form

$\sum_{x=1}^{k}\sum_{r=1}^{n^{[x]}} W^{[x]}(r)p_{\pi^{[x]}(r)}$ over a set of all permutations. If the number of jobs in each group $n^{[x]}$, $1 \leq x \leq k$, is known, an optimal solution can be obtained by running Algorithm Match2. To obtain a solution to the original problem $1\left|p_j + a^{[x]}\tau, MP\right|C_{\max}$, we solve all possible instances of problem $1\left|p_j + a^{[x]}\tau, MP\,(k-1)\right|C_{\max}$ by modifying the outcomes of Decisions 1-3 and choose the instance with the smallest value of the objective function as our optimal solution.

Similar to Chapter 6, we shall consider eight different versions of problem $1\left|p_j + a^{[x]}\tau, MP\right|C_{\max}$. For each version, the computed positional weights are found by making an appropriate substitution in (8.4). Recall that (8.4) is computed for the most general version, in which the deterioration rates are group-dependent, MPs are distinct and their duration depends on the start-time. Notice that unlike the positional weights (6.8) found in Chapter 6, the ones defined by (8.4) are non-increasing within each group. Additionally, their value is dependent on the number of jobs $n^{[x]}$ in a group. Owing to these differences the solution approaches provided in Chapter 6 cannot be used to obtain a solution for problem $1\left|p_j + a^{[x]}\tau, MP\right|C_{\max}$.

In the following three sections, we present three new solution approaches, which handle different versions of problem $1\left|p_j + a^{[x]}\tau, MP\right|C_{\max}$. Each of these solution approaches are similar in philosophy to the corresponding solution approaches in Chapter 6. They only differ in the execution of the idea. Table 8.1 lists out all the versions under consideration and mentions the section that deals with them.

	Constant Duration MPs		Start-time dependent MPs	
	Identical	Distinct	Identical	Distinct
Group-indep	Section 8.5	Section 8.5	Section 8.4	Section 8.3
Group-dep	Section 8.4	Section 8.3	Section 8.3	Section 8.3

Table 8.1: Different versions of problem $1\left|p_j + a^{[x]}\tau, MP\right|C_{\max}$

8.3 Solution Approach TimeJIGD

In this section, we describe a solution approach, which can solve all versions of problem $1\left|p_j + a^{[x]}\tau, MP\right|C_{\max}$ without prior knowledge of the number of jobs $n^{[x]}$ in each group. The only pre-requisite condition is that the computed positional weights $W^{[x]}(r)$, $1 \leq r \leq n^{[x]}$, should be non-increasing in every group x, $1 \leq x \leq k$.

Let us begin our consideration with the most general version of problem $1\left|p_j + a^{[x]}\tau, MP\right|C_{\max}$, such as the one discussed in Section 8.2. Assume

that Decisions 1-3 are taken in advance and denote the resulting problem as $1\left|p_j + a^{[x]}\tau, MP\left(k-1\right)\right| C_{\max}$. This problem can be solved by minimising the generic objective function (4.6) with positional weights given by (8.4) and the constant term given by (6.9). Below we outline a solution approach based on Theorem 6.1 that minimises (4.6).

First, set the value $n^{[x]} = n$, $1 \leq x \leq k$, and compute all positional weights $W^{[x]}\left(r\right)$, $1 \leq r \leq n$, $1 \leq x \leq k$, by (8.4). Notice that these positional weights represent a set of all possible values of $W^{[x]}\left(r\right)$ and are given by

$$\begin{pmatrix} U^{[1]}\left(1+a^{[1]}\right)^{n-1} & U^{[2]}\left(1+a^{[2]}\right)^{n-1} & \cdots & U^{[k]}\left(1+a^{[k]}\right)^{n-1} \\ U^{[1]}\left(1+a^{[1]}\right)^{n-2} & U^{[2]}\left(1+a^{[2]}\right)^{n-2} & \cdots & U^{[k]}\left(1+a^{[k]}\right)^{n-2} \\ \vdots & \vdots & \cdots & \vdots \\ U^{[1]}\left(1+a^{[1]}\right)^{2} & U^{[2]}\left(1+a^{[2]}\right)^{2} & \cdots & U^{[k]}\left(1+a^{[k]}\right)^{2} \\ U^{[1]}\left(1+a^{[1]}\right) & U^{[2]}\left(1+a^{[2]}\right) & \cdots & U^{[k]}\left(1+a^{[k]}\right) \\ U^{[1]} & U^{[2]} & \cdots & U^{[k]} \end{pmatrix}, \tag{8.5}$$

where for convenience we denote $U^{[x]} := \left(1+\alpha^{[x]}\right)$, $1 \leq x \leq k-1$, and $U^{[k]} := 1$. Each column in the above matrix represents all possible positional weights that can be associated with a particular group, the first element of column x representing a weight associated with the first position of group x, while the last element of column x, representing a weight associated with the last, i.e., the n- th position of group x, $1 \leq x \leq k$. The running time required to compute the matrix can be estimated as $O\left(nk\right)$.

According to Theorem 6.1, an optimal schedule can be found by choosing the n smallest of these values and assigning the largest jobs to the positions corresponding to the smallest positional weights. Notice that this approach is similar to Solution Approach PosiJIGD presented in Chapter 6. The main difference lies in the fact that because of the non-increasing order of the positional weights in each group, the n smallest values are found in consecutive positions at the bottom of the matrix (8.5). The smallest positional weight of a group is associated with the last position of that group, irrespective of the number of jobs in that group. Also notice that starting from the last element of a group and moving upwards, the value of the positional weights remain the same if even if $n^{[x]} \neq n$.

The problem of finding the corresponding schedule is structurally similar to that of scheduling jobs on parallel machines to minimise the total flow time and can be solved by a method similar to that described by Conway, Maxwell and Miller (1967), and its implementation given by Brucker (2007). According to the method, the groups are

filled in the reversed order, from the last position to the first one. Scan the jobs in the LPT order. To assign the first job, compare the k multipliers $U^{[x]}$, $1 \leq x \leq k$, and assign the job in the last position of the group associated with the smallest multiplier. The process continues, and for the current job the smallest of the k available multipliers determines the group and the position within the group where the job is assigned. This approach does not require any advance knowledge of the number of jobs $n^{[x]}$ in a group, even though the value of the positional weights are dependent on them.

A formal description of the algorithm is given below. Notice that the algorithm is very similar to Algorithm NSmall in Chapter 6, but uses the opposite direction of filling the groups.

Algorithm NSmallRev

INPUT: An instance of problem $1 \left| p_j + a^{[x]}\tau, MP\left(k-1\right) \right| C_{\max}$ with positional weights $W^{[x]}(r)$, $1 \leq r \leq n^{[x]}$, defined by (8.4)

OUTPUT: An optimal schedule $S^*\left(k\right)$ defined by the processing sequences $\pi^{[x]}$, $1 \leq x \leq k$

Step 1. If required, renumber the jobs in the LPT order. For each group x, $1 \leq x \leq k$, define an empty processing sequence $\pi^{[i]} := (\varnothing)$ and the weight $W^{[x]} = U^{[x]}$.

Step 2. For each job j from 1 to n do

(a) Find the smallest index v with $W^{[v]} = \min\left\{W^{[i]} | 1 \leq i \leq k\right\}$.

(b) Assign job j to group v and place it in front of the current permutation $\pi^{[v]}$, i.e., define $\pi^{[v]} := (j, \pi^{[v]})$. Define $W^{[v]} := W^{[v]}\left(1 + a^{[v]}\right)$.

Step 3. With the found permutation $\pi^* = \left(\pi^{[1]}, \pi^{[2]}, \ldots, \pi^{[k]}\right)$, compute the optimal value of the objective function $C_{\max}\left(S^*\left(k\right)\right)$ by substituting appropriate values in (4.6).

Step 2a of Algorithm NSmallRev requires $O\left(k\right)$ comparisons, while Step 2b is completed in constant time. Thus, in all Step 2, requires $O\left(nk\right)$ time and Step 3 requires $O\left(n\right)$ time. The following statement holds.

Theorem 8.1. *Algorithm NSmallRev solves an instance of problem* $1 \left| p_j + a^{[x]}\tau, MP\left(k-1\right) \right| C_{\max}$ *in* $O\left(nk\right)$ *time, provided that the LPT order of the jobs is known.*

Notice that in an optimal solution for an instance of problem $1\left|p_j + a^{[x]}\tau, MP\,(k-1)\right|C_{\max}$ it is possible that out of the k groups, certain groups are not assigned any jobs at all, i.e., $n^{[x]} = 0$. Such a situation can occur if an MP is not efficient in restoring the machine to a better state, and as a result the group that follows generates positional weights with big values. Such an instance can never result in an optimal schedule for the general problem $1\left|p_j + a^{[x]}\tau, MP\right|C_{\max}$, as we are unnecessarily spending time to perform an inefficient MP. This instance, if any, will be automatically eliminated from consideration if we try different combinations of Decision 1-3 to define problem $1\left|p_j + a^{[x]}\tau, MP\,(k-1)\right|C_{\max}$.

To determine the optimal solution for the general problem $1\left|p_j + a^{[x]}\tau, MP\right|C_{\max}$, all options associated with Decisions 1-3 must be enumerated and the solutions of the resulting sub-problems $1\left|p_j g_j^{[x]}\,(r)\text{-det}, MP\,(k-1)\right|C_{\max}$ be compared. The best of these solutions is chosen as the optimal solution for problem $1\left|p_j + a^{[x]}\tau, MP\right|C_{\max}$. The same procedure holds for all other less general versions of problem $1\left|p_j + a^{[x]}\tau, MP\right|C_{\max}$. Table 8.2 states the number of times Algorithm NSmallRev must be run in order to solve different versions of problem $1\left|p_j + a^{[x]}\tau, MP\right|C_{\max}$.

	Constant Duration MPs		Start-time dependent MPs	
	Identical	Distinct	Identical	Distinct
Group-indep	Section 8.5	Section 8.5	Section 8.4	$\sum_{k=1}^{K+1}\binom{K}{k-1}$
Group-dep	Section 8.4	$\sum_{k=1}^{K+1}\binom{K}{k-1}$	$\sum_{k=1}^{K+1}1$	$\sum_{k=1}^{K+1}\binom{K}{k-1}(k-1)$

Table 8.2: Number of times to run Algorithm NSmallRev to solve different versions of problem $1\left|p_j + a^{[x]}\tau, MP\right|C_{\max}$.

Notice that although Algorithm NSmallRev is able to solve all eight versions of problem $1\left|p_j + a^{[x]}\tau, MP\right|C_{\max}$, for some cases it is possible to make Decisions 1-3 on the fly by using another solution approach, which enables the optimal solution to be found in faster time. For such cases, a reference to the relevant section has been made in Table 8.2. For all other cases, the number of instances of problem $1\left|p_j + a^{[x]}\tau, MP\,(k-1)\right|C_{\max}$ can be computed similarly to the computation of the values of Table 6.2, which was done for case of positional deterioration in Chapter 6.

Since Algorithm NSmallRev requires $O\,(nk)$ time to run for a given k, $1 \leq k \leq K+1$, the respective running times that are needed to solve different versions of problem $1\left|p_j + a^{[x]}\tau, MP\right|C_{\max}$ can again be computed similarly to Section 6.5. For problem $1\left|p_j + a^{[x]}\tau, MP\right|C_{\max}$ with group-dependent deterioration rates and distinct MPs with start-time dependent durations, an optimal solution can be found in $O\left(nK^2 2^K\right)$

time. For the problem with group-independent deterioration rates and distinct MPs with constant durations, i.e., $a^{[x]} = a$, $\alpha^{[x]} = 0$, $1 \leq x \leq K+1$, an optimal solution can be found in $O\left(nK2^K\right)$ time. The same running time is needed for the problem with group-dependent deterioration rates and identical MPs with start-time dependent durations, i.e., $\alpha^{[x]} = \alpha$, $\beta^{[x]} = \beta$, $1 \leq x \leq K+1$. Lastly, for problem $1\left|p_j + a^{[x]}\tau, MP\right| C_{\max}$ with group-dependent deterioration rates and identical MPs with constant durations, i.e., $\alpha^{[x]} = 0$, $\beta^{[x]} = \beta$, $1 \leq x \leq K+1$, an optimal solution can be found in $O\left(nK^2\right)$ time.

8.4 Solution Approach TimeJIKdomi

In this section, we consider versions of problem $1\left|p_j + a^{[x]}\tau, MP\right| C_{\max}$ in which Decisions 1-4 can be made on the fly, without having to enumerate all possible options. We list out two versions for which this is possible:

Problem 1: Problem $1\left|p_j + a^{[x]}\tau, MP\right| C_{\max}$ with group-dependent deterioration rates and distinct MPs with constant durations, i.e., $\alpha^{[x]} = 0$, $1 \leq x \leq K+1$, subject to the condition that the deterioration rates can be ordered such that

$$a^{[1]} \leq a^{[2]} \leq \cdots \leq a^{[K+1]}, \tag{8.6}$$

and (6.11) hold simultaneously. This version is a generalisation of one of the cases found in Table 8.2, in which group-dependent deterioration rates are considered along with identical MPs of constant duration, i.e., $\alpha^{[x]} = 0$, $\beta^{[x]} = \beta$, $1 \leq x \leq K+1$. To see why (8.6) holds for the latter problem, follow the argument presented in Section 6.6, i.e., if identical MPs are performed in a schedule, the deterioration rate of the machine cannot become lower as more groups are created.

Problem 2 Problem $1\left|p_j + a^{[x]}\tau, MP\right| C_{\max}$ with group-independent deterioration rates, i.e., $a^{[x]} = a$, $1 \leq x \leq K+1$, and distinct MPs with start-time dependent durations, subject to the condition that the duration parameters of the MPs can be ordered such that (6.12) and (6.11) hold simultaneously. This version is a generalisation of one of the cases found in Table 8.2, in which group-independent positional rates are considered along with identical MPs of start-time dependent duration, i.e., $\alpha^{[x]} = \alpha$, $\beta^{[x]} = \beta$, $1 \leq x \leq K+1$.

In order to solve both versions of problem $1\left|p_j g_j^{[x]}(r)\text{-det}, MP\right| C_{\max}$ described above, the optimal choice for Decisions 2 and 3 can be made easily. If Decision 1

is assumed to be taken, so that $k-1$, $1 \leq k \leq K+1$, MPs are included in the schedule, then for both problems, the MPs with the indices $1, 2, \ldots, k-1$ are chosen and scheduled in the same order. This is the optimal choice for Problem 1 as the MPs with the indices $1, 2, \ldots, k-1$ have the smallest durations and create groups that contain the smallest deterioration rates, owing to (8.6) and (6.11) holding simultaneously. This is the optimal choice for Problem 2 as all MPs create identical groups and the ones with a smaller index have smaller values of the duration parameters, owing to (6.12) and (6.11) holding simultaneously. Notice that the order of the MPs is inconsequential in both cases.

With Decisions 1-3 having been made (with an assumed value of k), denote the resulting problem as $1 \left| p_j + a^{[x]}\tau, MP\,(k-1) \right| C_{\max}$. This problem can be solved by minimising the generic objective function of the form (4.6). For Problem 1, obtain the required positional weights $W^{[x]}(r)$ by substituting $\alpha^{[x]} = 0$, $1 \leq x \leq k$, in (8.4) so that

$$W^{[x]}(r) = \left(1 + a^{[x]}\right)^{n^{[x]}-r}, \ 1 \leq r \leq n^{[x]}, \ 1 \leq x \leq k, \tag{8.7}$$

and for Problem 2, substitute $a^{[x]} = a$, $1 \leq x \leq k$, so that

$$W^{[x]}(r) = \begin{cases} \left(1 + \alpha^{[x]}\right)(1+a)^{n^{[x]}-r}, & 1 \leq r \leq n^{[x]}, \ 1 \leq x \leq k-1, \\ (1+a)^{n^{[x]}-r} & 1 \leq r \leq n^{[x]}, \ x = k. \end{cases} \tag{8.8}$$

Set the value $n^{[x]} = n$, $1 \leq x \leq k$, and $k = K+1$, and compute all positional weights $W^{[x]}(r)$, $1 \leq r \leq n$, $1 \leq x \leq K+1$, for both problems by using the formulae above. Notice that the computed positional weights represent a set of all possible values of $W^{[x]}(r)$ across all possible groups. Further, notice that because of (8.6), the positional weights associated with Problem 1 are ordered such that for each k, $1 \leq k \leq K+1$, we have

$$W^{[1]}(r) \leq W^{[2]}(r) \leq \cdots \leq W^{[k]}(r), \ 1 \leq r \leq n,$$

and because of (6.12), the positional weights for Problem 2 are ordered such that for each k, $1 \leq k \leq K+1$, we have

$$W^{[k]}(r) \leq W^{[1]}(r) \leq W^{[2]}(r) \leq \cdots \leq W^{[k-1]}(r), \ 1 \leq r \leq n.$$

Notice that both Problems 1 and 2 satisfy the conditions of '*K-domi*' as laid out in Definition 6.1. For instances of problem $1 \left| p_j + a^{[x]}\tau, MP \right| C_{\max}$ that satisfy (6.11) and '*K-domi*', it is possible to compute the optimal values of $n^{[x]}$, $1 \leq x \leq k$, and Decisions 1-4 on the fly. Recall that for a fixed value of Decision 1, the optimal values

for Decisions 2 and 3 are already known. Thus, to solve problem $1\left|p_j + a^{[x]}\tau, MP\right| C_{\max}$ we only need to worry about finding the optimal values for Decisions 1 and 4.

Recall from Section 6.6 that under such conditions the relevant problem is solvable by Algorithm NSmall2, which finds an optimal schedule by computing the optimal values of $n^{[x]}$, $1 \leq x \leq k$, and Decisions 1 and 4 on the fly. Indeed, we can use a version of Algorithm NSmall2 to obtain an optimal solution for Problems 1 and 2 of this section as well.

Similar to the Solution Approach PosiJIKdomi provided in Section 6.6, create a list $H(v)$, $1 \leq v \leq K+1$, which is defined differently for Problems 1 and 2. For Problem 1, $H(v)$ contains the positional weights $W^{[v]}(r)$, $v \leq r \leq n$, for $n^{[x]} = n$, so that by (8.7) we have

$$H(v) := \begin{pmatrix} \left(1+a^{[v]}\right)^{n-v} \\ \left(1+a^{[v]}\right)^{n-(v+1)} \\ \vdots \\ \left(1+a^{[v]}\right)^{2} \\ \left(1+a^{[v]}\right) \\ 1 \end{pmatrix}, \ 1 \leq v \leq K+1.$$

For Problem 2, notice that the values of the positional weights given by (8.8) change dynamically as the value of k is changed. Thus, we define $H(v)$ so that this effect is incorporated; define

$$H(1) := \begin{pmatrix} (1+a)^{n-1} \\ (1+a)^{n-2} \\ \vdots \\ (1+a)^{2} \\ (1+a) \\ 1 \end{pmatrix}, \ H(v) := \begin{pmatrix} U^{[v-1]}(1+a^{v})^{n-v} \\ U^{[v-1]}(1+a)^{n-(v+1)} \\ \vdots \\ U^{[v-1]}(1+a)^{2} \\ U^{[v-1]}(1+a) \\ U^{[v-1]} \end{pmatrix}, \ 2 \leq v \leq K+1,$$

where $U^{[v-1]} = \left(1+\alpha^{[v-1]}\right)$, $2 \leq v \leq K+1$. Notice that for both problems, list $H(v)$ has at most $n-v+1$, $1 \leq v \leq K+1$, elements sorted in a non-increasing order. Similar to the argument in Section 6.6, it suffices to consider only $n-v+1$ positions in a list $H(v)$, as due to condition '*K-domi*' it can be ensured that each of the $v-1$ earlier groups will have at least 1 job scheduled in them.

With the found lists $H(v)$, $1 \leq v \leq K+1$, run Algorithm NSmall2 with a minor alteration made in Step 4. In Step 4 of Algorithm NSmall2, instead of using Algorithm NSmall to obtain an optimal processing sequence $\pi^* = \left(\pi^{[1]}, \pi^{[2]}, \ldots, \pi^{[k^*]}\right)$, use

Algorithm NSmallRev. This is necessary, as the computed positional weights for Problems 1 and 2 are non-increasing, and hence, it must be ensured that the groups are filled in a reversed order. Notice that Steps 1-3 of Algorithm NSmall2 are unaffected by the ordering of the positional weights in their respective groups. The following statement holds. It follows from Theorem 6.3 that an instance of problem $1\left|p_j + a^{[x]}\tau, MP\right| C_{\max}$ defined by Problems 1 and 2 can be solved in $O(nK)$ time, provided that the LPT order of the jobs is known.

8.5 Solution Approach TimeJIGI

In this section, we deal with problems in which the computed positional weights are group-independent, i.e., of the form $W^{[x]}(r) = W(r)$, $1 \leq x \leq k$, and additionally, they are ordered in a way such that $W(1) \geq W(2) \geq \cdots \geq W(n)$. Such a situation arises for versions of problem $1\left|p_j + a^{[x]}\tau, MP\right| C_{\max}$, in which the deterioration rates are group-independent, i.e., $a^{[x]} = a$, $1 \leq x \leq K+1$. The MPs are of constant duration and can be either distinct or identical. The problem with identical MPs is a special case of the problem with distinct MPs and does not result in better running times. Thus, we only consider the latter problem. Formally, we denote the described problem by $1\left|p_j + a\tau, MP[0]\right| C_{\max}$.

Notice that for problem $1\left|p_j + a\tau, MP[0]\right| C_{\max}$, the optimal choice for Decisions 2 and 3 can be made easily. Assume that an optimal solution to problem $1\left|p_j + a\tau, MP[0]\right| C_{\max}$ includes $k-1$ MPs in the schedule, so that the jobs are divided into k, $1 \leq k \leq K+1$, groups. Since it is known that the MPs create identical groups, it follows that the order in which they are performed is not important. Further, it is obvious that in order to choose $k-1$ MPs out of the available K, the ones with smaller durations are given priority. To ensure that the smallest $k-1$ MPs are chosen in an optimal schedule, we renumber the K available MPs in a way that (6.11) holds and select the ones with indices $1, 2, \ldots, k-1$. Lastly, we fix their order as per their index numbers.

With Decisions 1-3 having been made (with an assumed value of k), the resulting problem $1\left|p_j + a^{[x]}\tau, MP[0](k-1)\right| C_{\max}$ can be solved by minimising the generic objective function (4.6). Obtain the required positional weights $W^{[x]}(r)$ by substituting $a^{[x]} = a$, $\alpha^{[x]} = 0$, $1 \leq x \leq k$, in (8.4) so that we have

$$W^{[x]}(r) = (1+a)^{n^{[x]}-r}, \ 1 \leq r \leq n^{[x]}, \ 1 \leq x \leq k.$$

To solve an instance of problem $1\left|p_j + a^{[x]}\tau, MP\left[0\right](k-1)\right|C_{\max}$, below we outline a solution approach which is again based on Theorem 6.1 and is similar to the one given in Section 6.7.

First, set the value $n^{[x]} = n$, $1 \leq x \leq k$, and from the resulting set of positional weights

$$\begin{pmatrix} (1+a)^{n-1} & (1+a)^{n-1} & \cdots & (1+a)^{n-1} \\ (1+a)^{n-2} & (1+a)^{n-2} & \cdots & (1+a)^{n-2} \\ \vdots & \vdots & \cdots & \vdots \\ (1+a)^{2} & (1+a)^{2} & \cdots & (1+a)^{2} \\ (1+a) & (1+a) & \cdots & (1+a) \\ 1 & 1 & \cdots & 1 \end{pmatrix},$$

choose the n smallest of them. Obviously, the n smallest weights are found in consecutive positions at the bottom of the matrix. The smallest k positional weights are due to the last positions of each of the k groups. The next smallest k positional weights are due to the second last positions of each of the k groups, and so on. Next, scan the jobs in LPT order and assign the first k jobs to the first positions in each of the k groups, then the next k jobs going to the second positions in each of the k groups, and so on, until all jobs have been sequenced.

This is similar to the treatment of problem $1\left|p_j g\left(r\right)\text{-det}, MP\left[0\right](k-1)\right|C_{\max}$ considered in Section 6.7, the only difference being that for problem $1\left|p_j + a^{[x]}\tau, MP\left[0\right](k-1)\right|C_{\max}$, the groups need to be filled in a reversed order, as the smallest positional weights are found at the end of a group. Still, the optimal values for the number of jobs in each group can be given by (6.16). With known values of $W^{[x]}(r)$, $1 \leq r \leq n^{[x]}$, and $n^{[x]}$, $1 \leq x \leq k$, an optimal makespan $C_{\max}\left(S^*\left(k\right)\right)$ for problem $1\left|p_j + a^{[x]}\tau, MP\left[0\right](k-1)\right|C_{\max}$ can be found in $O\left(n\right)$ time by running Algorithm Match2.

To determine the optimal solution for problem $1\left|p_j + a^{[x]}\tau, MP\left[0\right]\right|C_{\max}$, all options associated with Decisions 1-3 must be enumerated. Decisions 2 and 3 have already been chosen optimally, thus, we only need to determine the optimal value of the number of MPs. We can do this by solving problem $1\left|p_j + a^{[x]}\tau, MP\left[0\right](k-1)\right|C_{\max}$ for all values of k, $1 \leq k \leq K+1$, and choosing the instance that delivers the smallest value of $C_{\max}(S^*\left(k\right))$. Thus, problem $1\left|p_j g\left(r\right)\text{-det}, MP\left[0\right]\right|C_{\max}$ can be solved in $O\left(nK\right)$ time. However, we prove that the sequence $C_{\max}(S^*(k))$, $1 \leq k \leq K+1$, is in fact V-shaped and thus, in order to search for the smallest value of $C_{\max}(S^*(k))$, we only need to evaluate $\lceil \log_2\left(K+1\right) \rceil$ options of k, $1 \leq k \leq K+1$.

Lemma 8.1. *For problem* $1\left|p_j + a^{[x]}\tau, MP\left[0\right](k-1)\right|C_{\max}$, *if the jobs be numbered*

in the LPT order (2.7), then the makespan of the optimal schedule can be written as

$$C_{\max}(S^*(k)) = P(S^*(k)) + \Gamma(k) = \sum_{j=1}^{n} p_j h\left(\left\lceil \frac{j}{k} \right\rceil\right) + \sum_{x=1}^{k-1} \beta^{[x]}. \tag{8.9}$$

where we denote $h(r) := (1+a)^{r-1}$, $1 \leq r \leq n$.

The proof of Lemma 8.1 is similar to that of Lemma 6.1.

Theorem 8.2. *For problem* $1\left|p_j + a^{[x]}\tau, MP[0]\right| C_{\max}$*, the sequence* $C_{\max}(S^*(k)) = P(S^*(k)) + \Gamma(k)$, $1 \leq k \leq K+1 \leq n$*, given by (8.9), is V-shaped.*

Proof: The proof of Theorem 8.2 is similar to that of Theorem 6.5. Since $h(r) = (1+a)^{r-1}$, $1 \leq r \leq n$, is a non-decreasing function, it follows from Theorem 5.2 that the sequence $P(S^*(k))$, $1 \leq k \leq n$, is convex. The convexity of the sequence $\Gamma(k)$, $1 \leq k \leq n$, is proved in Section 6.7 provided that (6.11) holds. Thus, the sequence $C_{\max}(S^*(k)) = P(S^*(k)) + \Gamma(k)$, $1 \leq k \leq K+1 \leq n$, is convex as the sum of two convex sequences, and is also V-shaped as Lemma 5.1 holds. □

Theorem 6.5 allows us to find the optimal number of groups k^*, $1 \leq k \leq K+1$, by binary search; see Algorithm BinarySearch presented in Section 6.7. All together, Algorithm BinarySearch explores at most $\lceil \log_2(K+1) \rceil$ values of k, so that the optimal solution for problem $1\left|p_j g(r)\text{-det}, MP[0]\right| C_{\max}$ can be found in $O(n \log K)$ time.

8.6 Conclusion

In this chapter, we solve several problems with time-dependent deterioration effects and maintenance activities. We are not aware of any other papers that study the problem of minimising the makespan, for a model in which a time-dependent effect of the form (3.15) is combined with multiple MPs. Moreover, in line with the rest of the thesis, we even consider enhanced models in which the deterioration rates can be group-dependent and the MPs can have different durations. We propose three solution approaches that solve different versions of problem $1\left|p_j + a^{[x]}\tau, MP\right| C_{\max}$. These solution approaches directly follow from the corresponding solution approaches presented in Chapter 6, which were used for solving different versions of problem $1\left|p_j g^{[x]}(r), MP\right| C_{\max}$. Notice that even though the underlying effect in both problems is very different, similar algorithmic ideas can be used to solve them. In the next chapter, we explore models that combine both of these effects and see if the same methodology can be extended to solve a very general problem with changing processing times.

Table 8.3 summarises all the problems considered in this chapter along with the running times needed to solve them.

	Constant Duration MPs		Start-time dependent MPs	
	Identical	Distinct	Identical	Distinct
Group-indep	$O(n \log K)$	$O(n \log K)$	$O(nK)$	$O\left(nK2^K\right)$
Group-dep	$O(nK)$	$O\left(nK2^K\right)$	$O(nK^2)$	$O\left(nK^2 2^K\right)$

Table 8.3: Computational complexities of different versions of problem $1\left|p_j + a^{[x]}\tau, MP\right| C_{\max}$.

CHAPTER 9

Combined Effects and Rate-Modifying Activities

In this chapter, we study models which combine many of the effects that we have discussed so far, namely:

- Positional and time-dependent effects;
- Learning and deterioration effects;
- Rate-modifying activities.

We develop a very general model and solve the problems of minimising the makespan and the total flow time by reducing them to a linear assignment problem with a product matrix. The results of this chapter are published in our recent paper Rustogi and Strusevich (2013b). The presentation of content in the paper is very similar to the content provided in this chapter.

9.1 Overview of the Problems

Recall that in Section 3.2.2, we have reviewed models in which a time-dependent effect of the form (3.15) is combined with a positional effect, so that the actual processing time of a job $j \in N$ sequenced in position r and starting at time $\tau \geq 0$ of a schedule is given by

$$p_j(\tau, r) = (p_j + a\tau)\, g(r). \tag{9.1}$$

Notice that most of the earlier papers on such combined models have emphasised on a clear distinction between the models with learning and deterioration. For example, Wang (2006) and Yang and Kuo (2009), consider a model with time-dependent

deterioration ($a > 0$) and positional learning ($g(r) = r^b, b < 0$), while Yang (2010) considers a model with time-dependent learning ($a < 0$) and positional deterioration ($g(r) = r^b$, $b > 0$).

In Chapter 6, we discuss that in the case of a positional effect, it is possible to study a model with both deterioration and learning effects, if we redefine our positional factors as $g(r) := g_d(r)\, g_l(r)$, $1 \le r \le n$, where $g_d(r)$ represents a deterioration factor so that $g_d(1) \le g_d(2) \le \cdots \le g_d(n)$ holds, and $g_l(r)$ represents a learning factor so that $g_l(1) \ge g_l(2) \ge \cdots \ge g_l(n)$ holds. As a result, the positional factors $g(r)$, $1 \le r \le n$, are non-monotone with respect to r. Example 4.1 illustrates such an effect. Similarly, in the case of a time-dependent effect of the form 3.15, we can combine learning and deterioration effects by simply redefining the rate $a := d + l$, where $d > 0$ is a deterioration rate and $l < 0$ is a learning rate. As a result, the resulting rate a can be of an arbitrary sign.

In this chapter, we mainly concentrate on a combined model of the form (9.1), in which there are no assumptions made on the monotone behaviour of the positional factors and on the sign of the rate a that defines a time-dependent effect. Such a model covers all known models with job-independent effects and allows handling simultaneous learning and deterioration effects of both types (time-dependent and positional). In all the problems that we consider, we assume that the jobs of set $N = \{1, 2, \ldots, n\}$ have to be processed on a single machine. The jobs are available for processing at time zero and are independent, i.e., there are no precedence constraints and any processing sequence is feasible. At time zero, the machine is assumed to be in a perfect processing state, and its conditions change with the course of processing, both with respect to time and position.

Below we demonstrate that if no rate-modifying activities are included in a schedule then under the conditions defined above, the problems of minimising the makespan and the total flow time can be solved by Algorithm Match1 described in Chapter 4. Suppose that the jobs are processed according to a sequence $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$. Extending the arguments by Wang (2006) and Yang and Kuo (2009), it is easy to prove that the problems of minimising the makespan and the total flow time can be reduced to minimising (4.3), with $\Gamma = 0$ and positional weights

$$W(r) = g(r) \prod_{i=r+1}^{n} (1 + ag(i)), \ 1 \le r \le n,$$

and

$$W(r) = g(r)\left[\sum_{u=r}^{n}\prod_{i=r+1}^{u}\left(1+ag\left(i\right)\right)\right],\ 1 \leq r \leq n,$$

respectively. Thus, both problems reduce to minimising a linear form $\sum_{r=1}^{n} W(r)p_{\pi(r)}$ over a set of all permutations and are solvable in $O(n \log n)$ time each. This reduction holds, irrespective of the sign of a and the shape of the function $g(r)$, i.e. the sequence $g(r)$, $1 \leq r \leq n$, can be made up of arbitrary positional factors, even non-monotone. In the case of a net learning effect, where $a < 0$, care should be taken to guarantee that the actual processing times do not become negative.

In the following sections, we further extend this general model by combining it with rate-modifying activities.

9.2 Models with Rate Modifying Activities

Similar to earlier chapters, consider a general situation in which the decision-maker is presented with a total of $K \geq 0$ possible rate-modifying activities, which can be either distinct or alike. For each RMP, it is exactly known how it affects the processing conditions of the machine, should the decision-maker decide to include it into a schedule. Recall that an RMP can either be seen as maintenance period, which is aimed at restoring the machine conditions to a better state so that the processing times become smaller, or on the other hand, it can be seen as an activity in which a machine/operator is replaced, so that all learning advantages are lost and the actual processing times of jobs become larger. Another form of RMP which is studied by Ji and Cheng (2010), can be of the form in which the learning rate of the machine is further enhanced after the RMP.

So far in this thesis, we have made a clear distinction between different kinds of RMPs and have only considered problems in which an RMP is treated as a maintenance period and is used in models with a deterioration effect only. In this chapter, however, we do not impose any restrictions on the RMPs. They can have any arbitrary effect on the machine conditions. An illustration of a schedule with different types of RMPs is provided in Example 4.1, in which we combine learning and deterioration effects for a pure positional model. In the same example, we also consider a situation in which the positional factors $g^{[x]}\left(r\right)$ are found to be dependent on the number of jobs in previous groups. In this chapter, we extend such a model by combining it with a time-dependent effect. As a result in our main model, the actual processing time of a job is seen as affected by the following factors:

- the group a job is assigned to;
- the position of the job within its group;
- the number of jobs scheduled in each of the previous groups;
- the time elapsed before processing the job within its group;
- the time elapsed in each of the previous groups.

Additionally, the duration D_{RMP} of an RMP with an index y, $1 \leq y \leq K$, is given by

$$D_{RMP} = \left(\alpha^{[y]}\tau + \psi^{[y]}\right) f^{[y]}(\eta), \tag{9.2}$$

where

- τ is the time since the last RMP was performed;
- $\alpha^{[y]}$ and $\psi^{[y]} > 0$ are given constants known for each RMP, $1 \leq y \leq K$;
- the factor $f^{[y]}(\eta)$ represents a positional factor which makes the duration of the RMP dependent on the number of jobs η scheduled since the last RMP was performed.

The conceived model allows us to handle a wide range of practical problems, which have not been studied in the scheduling literature so far. Notice that the above defined model to determine the duration of an RMP is a further generalisation of the definition of D_{RMP} given in (4.1), as it additionally allows a positional factor, so that the duration is dependent on the start-time of the RMP and also on the position of the RMP in the processing sequence. The latter is in line with positionally dependent setup times for models that arise in group technology scheduling; see Lee and Wu (2009). As a result, according to the revised model the later an RMP is scheduled, both with respect to time and position, the longer/shorter it takes to complete it, depending on the sign of $\alpha^{[y]}$ and the nature of the function $f^{[y]}$.

Formally, consider a schedule $S(k)$ with $k-1$, $1 \leq k \leq K+1$, RMPs and a permutation of jobs given by $\pi = \left(\pi^{[1]}, \pi^{[2]}, \ldots, \pi^{[k]}\right)$. Assume that each group contains a total of $n^{[x]}$ jobs, so that $\pi^{[x]} = \left(\pi^{[x]}(1), \pi^{[x]}(2), \ldots, \pi^{[x]}\left(n^{[x]}\right)\right)$, $1 \leq x \leq k$, where $\sum_{x=1}^{k} n^{[x]} = n$. Depending on which RMPs are chosen and the order in which they are performed, the actual processing time of a job $j = \pi^{[x]}(r)$, scheduled in position r,

$1 \leq r \leq n^{[x]}$, of the x-th group, $1 \leq x \leq k$, is given by

$$p_j^{[x]}(r) = \left(p_{\pi^{[x]}(r)} + a_1^{[x]} F_1 + a_2^{[x]} F_2 + ... + a_{x-1}^{[x]} F_{x-1} + a_x^{[x]} F_{(x,r-1)}\right) g^{[x]}(r), \quad 1 \leq r \leq n^{[x]},\ 1 \leq x \leq k, \tag{9.3}$$

where F_v denotes the total processing time of all jobs scheduled in a group v, while $F_{(x,r)}$ represents the total duration of the first r jobs in a group x, and for completeness we have the condition $F_0 = F_{(x,0)} = 0$. The terms $g^{[x]}(r)$ represent positive group-dependent job-independent positional factors, which do not have to be monotone within a particular group. The terms $a_1^{[x]}, a_2^{[x]}, \ldots, a_x^{[x]}$ are real numbers and represent the group-dependent rates associated with time-dependent effects. Notice that some of the rates $a_1^{[x]}, a_2^{[x]}, \ldots, a_x^{[x]}$ may be negative, which corresponds to a learning effect. Without going into technical details, in what follows we assume that the rates $a_1^{[x]}, a_2^{[x]}, \ldots, a_x^{[x]}$ are such that the processing times remain positive. See, e.g., Yang (2010) where the required conditions are explicitly written out for a less general combined effect.

For a job scheduled in position r of group x, the rates $a_1^{[x]}, a_2^{[x]}, \ldots, a_{x-1}^{[x]}$, determine how the length of previous groups affects the job's processing time, whereas $a_x^{[x]}$ determines how the processing time of the job is affected by the time elapsed between the opening of the x-th group and the start time of the job in position r, $1 \leq r \leq n^{[x]}$. The superscript $[x]$ associated with these rates stresses that the rates are group-dependent, and can assume different values and signs depending on the group x, $1 \leq x \leq k$, a job is scheduled in. For example, to determine the actual processing time of a job scheduled in the third group, the required rates are $a_1^{[3]}, a_2^{[3]}$ and $a_3^{[3]}$, whereas if a job is placed in the fourth group, the required rates are $a_1^{[4]}, a_2^{[4]}, a_3^{[4]}$ and $a_4^{[4]}$.

Notice that it is always assumed that during an RMP the system undergoes neither learning nor deterioration. Thus, the actual processing times of the jobs given by (9.3) is independent of the duration of the RMPs.

Further, since the number of jobs, $n^{[x]}$, in each group and the total duration of each group, F_x, is known, it follows from (9.2) that the duration of an RMP after the x-th, $1 \leq x \leq k-1$, group can be given by

$$T_x = \alpha_1^{[x]} F_1 + \alpha_2^{[x]} F_2 + \cdots + \alpha_x^{[x]} F_x + \beta^{[x]},\ 1 \leq x \leq k-1, \tag{9.4}$$

where the values $\alpha_1^{[x]}, \alpha_2^{[x]}, \ldots, \alpha_x^{[x]}$, determine how the length of previous groups affect the duration of the RMP scheduled after the x-th group, and $\beta^{[x]} > 0$ is a constant, $1 \leq x \leq k-1$. The values $\alpha_1^{[x]}, \alpha_2^{[x]}, \ldots, \alpha_x^{[x]}$, can be seen as being analogous to the rates $a_1^{[x]}, a_2^{[x]}, \ldots, a_{x-1}^{[x]}$, defined in (9.3). They allow us to incorporate RMPs of a different

nature in a schedule. Consider an instance in which a machine undergoes deterioration and learning simultaneously. Two RMPs are to be included in the processing sequence, the first being used as a training period for the operator, while the second being an MP which repairs the machine. As a result three groups are created, with the number of jobs in each being known as $n^{[1]}$, $n^{[2]}$, and $n^{[3]}$, and the duration of each group being known as F_1, F_2, and F_3. Thus, according to (9.2) the duration of the first RMP will be given by $D_{RMP}(1) = \left[\alpha^{[1]}F_1 + \psi^{[1]}\right] f^{[1]}\left(n^{[1]}\right)$, while the duration of the second RMP will be given by $D_{RMP}(2) = \left[\alpha^{[2]}(F_1 + F_2) + \psi^{[2]}\right] f^{[2]}\left(n^{[1]} + n^{[2]}\right)$. The latter relation holds as the machine is undergoing continuous deterioration during the first two groups, and thus, the time till the first MP is performed is equal to $F_1 + F_1$. As a result, in terms of the formula (9.4), for the first RMP we have $\alpha_1^{[1]} = \alpha^{[1]} f^{[1]}\left(n^{[1]}\right)$, and $\beta^{[1]} = \psi^{[1]} f^{[1]}\left(n^{[1]}\right)$, while for the second RMP we have $\alpha_1^{[2]} = \alpha_2^{[2]} = \alpha^{[2]} f^{[2]}\left(n^{[1]} + n^{[2]}\right)$, and $\beta^{[2]} = \psi^{[2]} f^{[2]}\left(n^{[1]} + n^{[2]}\right)$. Notice that the values $\alpha_1^{[x]}, \alpha_2^{[x]}, \ldots, \alpha_x^{[x]}$, can assume any sign as long as it is ensured that the duration of the RMPs is positive.

Extending the standard three-field notation, we denote the problems of minimising the makespan and the total flow time for scheduling models that satisfy the above conditions by $1\,|Combi, RMP|\,C_{\max}$ and $1\,|Combi, RMP|\,\sum C_j$, respectively. Here, in the middle field we write "*Combi*" to stress that the processing times are subject to a combined effect (9.3), and we write "*RMP*" to indicate that rate modifying activities are being applied in accordance with (9.4). This is the first study in which combined models of such a general kind are being studied. An optimal solution to problem $1\,|Combi, RMP|\,F$ must deliver optimal choices for each of the Decisions 1-4 defined in Chapter 4, for $F \in \{C_{\max}, \sum C_j\}$.

Complicated as it looks, the model essentially incorporates and generalises almost every job-independent effect known in scheduling with changing processing times. It is flexible enough to serve as a model of many plausible scenarios that may be found in practice, e.g., in the area manufacturing or shop floor production planning. Below we give an illustration of a possible application.

Example 9.1. A human operator uses a tool to process n jobs. During the processing of the jobs, two RMPs will be included in the schedule. It is known that the first RMP is actually a maintenance period which restores the machine to its original condition. However, the deterioration rate of the machine becomes greater after the maintenance period, since the original spare parts are not used. This RMP also provides the operator with sufficient rest, so that after the first RMP the operator is as fresh as he/she was at the beginning of the schedule. The duration of this RMP is dependent on its start-time, i.e., it follows (9.4) for $x = 1$, with $\alpha_1^{[1]} = \alpha'$ and $\beta^{[1]} = \beta'$. The second RMP takes a

Group	Parameter	Value
1	$a_1^{[1]}$	$d'_m + d'_w + l'_w$
	$g^{[1]}(r)$	$(d'_m + 1)^{r-1} r^{d'_w + l'_w}, 1 \leq r \leq n^{[1]}$
2	$a_1^{[2]}$	l'_w
	$a_2^{[2]}$	$d''_m + d'_w + l'_w$
	$g^{[2]}(r)$	$(d''_m + 1)^{r-1} \left(n^{[1]} + r\right)^{l'_w} r^{d'_w}, 1 \leq r \leq n^{[2]}$
3	$a_1^{[3]}$	l'''_w
	$a_2^{[3]}$	$d''_m + l'''_w$
	$a_3^{[3]}$	$d''_m + d''_w + l''_w$
	$g^{[3]}(r)$	$(d''_m + 1)^{n^{[2]}+r-1} r^{d''_w + l''_w}, 1 \leq r \leq n^{[3]}$

Table 9.1: Parameters for Example 9.1

constant time β'', i.e., $\alpha_1^{[2]} = \alpha_2^{[2]} = 0$ and $\beta^{[2]} = \beta''$, but does not repair the machine at all. Instead, a new operator is brought in. Below, we distinguish between the learning and deterioration parameters for the machine and those for the operator by using the subscript "m" for the machine and "w" for the operator (worker), respectively.

In a feasible schedule the jobs will be split into $k = 3$ groups. The machine suffers from a linear time-dependent deterioration effect, the deterioration rate being equal to $d'_m > 0$ before the first RMP (the first group), and equal to $d''_m > d'_m > 0$ after the first RMP (for the second and the third groups). Additionally, the machine is also affected by a positional exponential deterioration effect of the form $(d'_m + 1)^{r-1}$ before the first RMP and of the form $(d''_m + 1)^{r-1}$ after that RMP. The operators are also subject to time-dependent effects; the deterioration and learning rates for Operator 1 are $d'_w > 0$ and $l'_w < 0$, respectively, and those for Operator 2 are $d''_w > 0$ and $l''_w < 0$, respectively. It is known that in addition to the skills gained while processing the jobs, Operator 2 also passively learns with a rate $l'''_w < 0$, while he observes Operator 1 process the jobs in groups 1 and 2. Lastly, the performance of the workers is also affected by a polynomial positional effect and is quantified by r^δ, where the appropriate value of δ is one of d'_w, l'_w, d''_w and l''_w, depending on the scenario. There is no positional effect associated with the passive learning effect of Operator 2.

For the described example, the parameters of our model (9.3) can be set as shown in Table 9.1.

Notice that our model allows us to assume (unlike, e.g., Yang (2010)), that during an RMP, if the operator is not replaced, he/she does not lose his/her skills which were improved due to learning in the earlier groups of the schedule. Similarly, if during an RMP a machine is not fully repaired, our model is capable of handling the resulting

situation in which the deterioration effect from the group before the RMP must be carried forward to the next group. The same phenomenon is also observed in the passive learning effect of Operator 2, in which the skills gained during groups 1 and 2 must be carried forward to group 3. These time-dependent effects can be captured by the group-dependent parameters $a_v^{[x]}$, $1 \leq v \leq x-1$, $1 \leq x \leq k$. Thus, to model the situation in the given example, we define $a_1^{[2]} := l'_w$ (implying that Operator 1 has gained an experience of F_1 time units before starting group 2), $a_1^{[3]} := l'''_w$ (implying that Operator 2 has gained a passive learning experience of F_1 time units before starting group 3) and $a_2^{[3]} := d''_m + l'''_w$ (implying that the machine has deteriorated for F_2 time units and Operator 2 has gained a passive learning experience of F_2 time units before starting group 3). These positional effects are simply captured by adjusting the relative position of a job in the relevant group, as illustrated in Example 4.1.

As demonstrated in the above example, with appropriate use of the parameters, $a_v^{[x]}$, $\alpha_v^{[x]}$, $1 \leq v \leq x$, $g^{[x]}(r)$, $1 \leq r \leq n^{[x]}$, and $\beta^{[x]}$, for each x, $1 \leq x \leq k$, our model as defined in (9.3), together with (9.4), can be used to represent a wide variety of practical situations.

9.3 Computing Positional Weights

To solve problem $1\,|Combi, RMP|\,F$, we first assume that Decisions 1-3 are taken in advance, so that we know that a total of $k-1$ MPs have been included in the schedule. As a result the jobs are split into k, $1 \leq k \leq K+1$, groups. Define the parameters $a_1^{[x]}, a_2^{[x]}, \ldots, a_x^{[x]}$, $1 \leq x \leq k$, and $\alpha_1^{[x]}, \alpha_2^{[x]}, \ldots, \alpha_x^{[x]}, \beta^{[x]}$, $1 \leq x \leq k-1$, and denote the resulting problem as $1\,|Combi, RMP\,(k-1)|\,F$.

To solve problem $1\,|Combi, RMP\,(k-1)|\,F$ consider a schedule $S(k)$ with a permutation of jobs $\pi = \left(\pi^{[1]}, \pi^{[2]}, \ldots, \pi^{[k]}\right)$. Assume that each group contains a total of $n^{[x]}$ jobs, so that $\pi^{[x]} = \left(\pi^{[x]}(1), \pi^{[x]}(2), \ldots, \pi^{[x]}\left(n^{[x]}\right)\right)$, $1 \leq x \leq k$, where $\sum_{x=1}^{k} n^{[x]} = n$. We now derive an expression for the total time it takes to process all jobs in a group x, $1 \leq x \leq k$.

Notice that throughout this chapter we assume that an empty product is equal to one, while an empty sum is equal to zero, i.e., $\prod_{i=j}^{r}(\cdot) = 1$ and $\sum_{i=j}^{r}(\cdot) = 0$, for $j > r$.

Lemma 9.1. *Given a group x, $1 \leq x \leq k$, and job $j = \pi^{[x]}(r)$ sequenced in the r-th position, $1 \leq r \leq n^{[x]}$, of the group, the completion time $F_{(x,r)}$ of the job with respect*

to the start time of the group is given by

$$F_{(x,r)} = \sum_{u=1}^{r} \left(A^{[x]} + p_{\pi^{[x]}(u)}\right) B^{[x]}(u,r), \tag{9.5}$$

where $A^{[x]} := \sum_{v=1}^{x-1} a_v^{[x]} F_v$, *and*

$$B^{[x]}(u,r) := g^{[x]}(u) \prod_{i=u+1}^{r} \left(1 + a_x^{[x]} g^{[x]}(i)\right), 1 \le u \le r. \tag{9.6}$$

Proof: We prove this lemma by induction. For job $j = \pi^{[x]}(r)$, the value $F_{(x,r)}$ for $r = 1$ is given in accordance with (9.3) by

$$\begin{aligned} p_j^{[x]}(1) &= \left(p_{\pi^{[x]}(1)} + a_1^{[x]} F_1 + a_2^{[x]} F_2 + ... + a_{x-1}^{[x]} F_{x-1}\right) g^{[x]}(1) \\ &= \left(p_{\pi^{[x]}(1)} + A^{[x]}\right) g^{[x]}(1) = \left(p_{\pi^{[x]}(1)} + A^{[x]}\right) B^{[x]}(1,1), \end{aligned}$$

which corresponds to the right-hand side of (9.5).

Assume that for r, $1 < r \le n^{[x]}$, the equality

$$F_{(x,r-1)} = \sum_{u=1}^{r-1} \left(A^{[x]} + p_{\pi^{[x]}(u)}\right) B^{[x]}(u,r-1), \tag{9.7}$$

holds. We know that

$$F_{(x,r)} = F_{(x,r-1)} + p_j^{[x]}(r).$$

Substituting (9.3) we get

$$\begin{aligned} F_{(x,r)} &= F_{(x,r-1)} + \left(p_{\pi^{[x]}(r)} + A^{[x]} + a_x^{[x]} F_{(x,r-1)}\right) g^{[x]}(r) \\ &= \left(1 + a_x^{[x]} g^{[x]}(r)\right) F_{(x,r-1)} + \left(p_{\pi^{[x]}(r)} + A^{[x]}\right) g^{[x]}(r). \end{aligned}$$

Substituting the value of $F_{(x,r-1)}$ from (9.7), we obtain

$$F_{(x,r)} = \left(1 + a_x^{[x]} g^{[x]}(r)\right) \sum_{u=1}^{r-1} \left(A^{[x]} + p_{\pi^{[x]}(u)}\right) B^{[x]}(u,r-1) + \left(A^{[x]} + p_{\pi^{[x]}(r)}\right) g^{[x]}(r).$$

It can be easily verified that $\left(1 + a_x^{[x]} g^{[x]}(r)\right) B^{[x]}(u,r-1) = B^{[x]}(u,r)$ and

$g^{[x]}(r) = B^{[x]}(r,r)$, so that we obtain

$$\begin{aligned} F_{(x,r)} &= \sum_{u=1}^{r-1} \left(A^{[x]} + p_{\pi^{[x]}(u)}\right) B^{[x]}(u,r) + \left(A^{[x]} + p_{\pi^{[x]}(r)}\right) B^{[x]}(r,r) \\ &= \sum_{u=1}^{r} \left(A^{[x]} + p_{\pi^{[x]}(u)}\right) B^{[x]}(u,r), \end{aligned}$$

which proves the lemma. □

Due to Lemma 9.1, the total processing time of a group x, $1 \le x \le k$, can be written as

$$\begin{aligned} F_x &= F_{\left(x,n^{[x]}\right)} = \sum_{r=1}^{n^{[x]}} \left(A^{[x]} + p_{\pi^{[x]}(r)}\right) B^{[x]}\left(r, n^{[x]}\right) \\ &= \sum_{r=1}^{n^{[x]}} \left(\sum_{v=1}^{x-1} a_v^{[x]} F_v\right) B^{[x]}\left(r, n^{[x]}\right) + \sum_{r=1}^{n^{[x]}} p_{\pi^{[x]}(r)} B^{[x]}\left(r, n^{[x]}\right) \\ &= \sum_{v=1}^{x-1} \left(a_v^{[x]} \sum_{r=1}^{n^{[x]}} B^{[x]}\left(r, n^{[x]}\right)\right) F_v + \sum_{r=1}^{n^{[x]}} p_{\pi^{[x]}(r)} B^{[x]}\left(r, n^{[x]}\right). \end{aligned}$$

For convenience, denote

$$D^{[x]} := \sum_{r=1}^{n^{[x]}} p_{\pi^{[x]}(r)} B^{[x]}\left(r, n^{[x]}\right), \ 1 \le x \le k, \tag{9.8}$$

and

$$b_v^{[x]} := a_v^{[x]} \sum_{r=1}^{n^{[x]}} B^{[x]}\left(r, n^{[x]}\right), \ 1 \le v \le x-1, \ 1 \le x \le k. \tag{9.9}$$

Then F_x can be rewritten as

$$F_x = b_1^{[x]} F_1 + b_2^{[x]} F_2 + \cdots + b_{x-1}^{[x]} F_{x-1} + D^{[x]}. \tag{9.10}$$

Lemma 9.2. *The total processing time of a group x, $1 \le x \le k$, is given by*

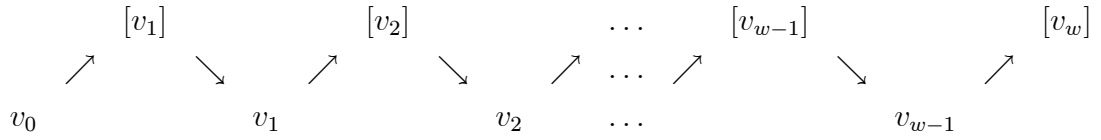
$$F_x = \sum_{v=1}^{x} E^{[v,x]} D^{[v]}, \tag{9.11}$$

where $E^{[x,x]} := 1$ *and for* $v \leq x-1$

$$E^{[v,x]} := \sum_{w=1}^{x-v} \sum_{v=v_0<v_1<\cdots<v_w=x} b_{v_0}^{[v_1]} b_{v_1}^{[v_2]} b_{v_2}^{[v_3]} \cdots b_{v_{w-1}}^{[v_w]}, \tag{9.12}$$

where in (9.12) for each w, $1 \leq w \leq x-v$, *the second summation is taken over all increasing sequences of* $w+1$ *distinct integers* $(v_0, v_1, \ldots, v_w)$ *with* $v_0 = v$ *and* $v_w = x$.

Proof: Please notice the zigzag pattern of the subscripts and superscripts in the right-hand side of (9.12), as outlined below.



The proof of the lemma is by induction. For $x = 1$, it follows from (9.10) that $F_1 = D^{[1]}$. On the other hand, for $x = 1$, (9.11) reduces to $F_1 = E^{[1,1]} D^{[1]}$, and since $E^{[1,1]} = 1$ by definition, we also obtain $F_1 = D^{[1]}$.

Assume now that the lemma holds for all groups $1, 2, \ldots, x-1$, and prove that it holds for group $x \leq k$. Starting with (9.10), we write

$$F_x = \sum_{v=1}^{x-1} b_v^{[x]} F_v + D^{[x]},$$

and use the induction assumption to substitute (9.11) into the above expression to obtain

$$\begin{aligned} F_x &= \sum_{v=1}^{x-1} b_v^{[x]} \sum_{y=1}^{v} E^{[y,v]} D^{[y]} + D^{[x]} \\ &= \sum_{v=1}^{x-1} \sum_{y=v}^{x-1} b_y^{[x]} E^{[v,y]} D^{[v]} + D^{[x]} \\ &= \sum_{v=1}^{x-1} \left(b_v^{[x]} E^{[v,v]} + \sum_{y=v+1}^{x-1} b_y^{[x]} E^{[v,y]} \right) D^{[v]} + D^{[x]}. \end{aligned}$$

Further, using the induction assumption, replace $E^{[v,v]}$ by 1 and substitute

$E^{[v,y]}, v < y$, in accordance with (9.12). We deduce

$$\begin{aligned} F_x &= \sum_{v=1}^{x-1}\left(b_v^{[x]} + \sum_{y=v+1}^{x-1} b_y^{[x]}\left(\sum_{w=1}^{y-v} \sum_{v=v_0<v_1<\cdots<v_w=y} b_{v_0}^{[v_1]} b_{v_1}^{[v_2]} b_{v_2}^{[v_3]} \cdots b_{v_{w-1}}^{[v_w]}\right)\right) D^{[v]} + D^{[x]} \\ &= \sum_{v=1}^{x-1}\left(b_v^{[x]} + \sum_{y=v+1}^{x-1}\sum_{w=1}^{y-v} \sum_{v=v_0<v_1<\cdots<v_w=y} b_{v_0}^{[v_1]} b_{v_1}^{[v_2]} b_{v_2}^{[v_3]} \cdots b_{v_{w-1}}^{[v_w]} b_y^{[x]}\right) D^{[v]} + D^{[x]} \\ &= \sum_{v=1}^{x-1}\left(b_v^{[x]} + \sum_{y=v+1}^{x-1}\sum_{w=1}^{y-v} \sum_{v=v_0<v_1<\cdots<v_w=y} b_{v_0}^{[v_1]} b_{v_1}^{[v_2]} b_{v_2}^{[v_3]} \cdots b_{v_{w-1}}^{[v_w]} b_{v_w}^{[x]}\right) D^{[v]} + D^{[x]}. \end{aligned}$$

Observe that for a fixed w the equality

$$\sum_{y=v+1}^{x-1} \sum_{v=v_0<v_1<\cdots<v_w=y} b_{v_0}^{[v_1]} b_{v_1}^{[v_2]} b_{v_2}^{[v_3]} \ldots b_{v_{w-1}}^{[v_w]} b_{v_w}^{[x]} = \sum_{v=v_0<v_1<\cdots<v_w\le x-1} b_{v_0}^{[v_1]} b_{v_1}^{[v_2]} b_{v_2}^{[v_3]} \ldots b_{v_{w-1}}^{[v_w]} b_{v_w}^{[x]}$$

holds, where the summation in the right-hand side is taken over all increasing sequences of $w+1$ distinct integers $(v_0, v_1, \ldots, v_w)$ with $v_0 = v$ and $v_w \le x-1$.

Notice that for $y = v+1$ and for $y = x-1$ we have that $w = 1$ and $w = x-v-1$, respectively, i.e., $1 \le w \le x-v-1$. This means that

$$F_x = \sum_{v=1}^{x-1}\left(b_v^{[x]} + \sum_{w=1}^{x-v-1} \sum_{v=v_0<v_1<\cdots<v_w\le x-1} b_{v_0}^{[v_1]} b_{v_1}^{[v_2]} b_{v_2}^{[v_3]} \cdots b_{v_{w-1}}^{[v_w]} b_{v_w}^{[x]}\right) D^{[v]} + D^{[x]}.$$

Replacing w with $w-1$, rewrite

$$F_x = \sum_{v=1}^{x-1}\left(b_v^{[x]} + \sum_{w=2}^{x-v} \sum_{v=v_0<v_1<\cdots<v_{w-1}\le x-1} b_{v_0}^{[v_1]} b_{v_1}^{[v_2]} b_{v_2}^{[v_3]} \cdots b_{v_{w-2}}^{[v_{w-1}]} b_{v_{w-1}}^{[x]}\right) D^{[v]} + D^{[x]}.$$

It can be easily verified that

$$b_v^{[x]} = \sum_{w=1}^{1} \sum_{v=v_0<v_1<\cdots<v_{w-1}<v_w=x} b_{v_0}^{[v_1]} b_{v_1}^{[v_2]} \cdots b_{v_{w-2}}^{[v_{w-1}]} b_{v_{w-1}}^{[v_w]}, 1 \le v \le x-1,$$

so that $F_x, 1 \le x \le k$, can be rewritten as

$$\begin{aligned} F_x &= \sum_{v=1}^{x-1}\left(\sum_{w=1}^{x-v} \sum_{v=v_0<v_1<\cdots<v_{w-1}<v_w=x} b_{v_0}^{[v_1]} b_{v_1}^{[v_2]} b_{v_2}^{[v_3]} \cdots b_{v_{w-2}}^{[v_{w-1}]} b_{v_{w-1}}^{[v_w]}\right) D^{[v]} + D^{[x]} \\ &= \sum_{v=1}^{x-1} E^{[v,x]} D^{[v]} + E^{[x,x]} D^{[x]} = \sum_{v=1}^{x} E^{[v,x]} D^{[v]}, \end{aligned}$$

which proves the lemma. □

The expressions in Lemma 9.2 look heavy; below we illustrate them for small values of x, e.g., $x \leq 4$. Recall that for $x = 1$, we have $F_1 = D^{[1]}$. For $x = 2$, the formula (9.10) gives $F_2 = b_1^{[2]} F_1 + D^{[2]} = b_1^{[2]} D^{[1]} + D^{[2]}$, which complies with (9.11) for $x = 2$, since $E^{[1,2]} = b_1^{[2]}$ and $E^{[2,2]} = 1$. Similarly, for $x = 3$, the formula (9.10) reduces to

$$\begin{aligned} F_3 &= b_1^{[3]} F_1 + b_2^{[3]} F_2 + D^{[3]} = b_1^{[3]} D^{[1]} + b_2^{[3]} \left(b_1^{[2]} D^{[1]} + D^{[2]} \right) + D^{[3]} \\ &= \left(b_1^{[3]} + b_1^{[2]} b_2^{[3]} \right) D^{[1]} + b_2^{[3]} D^{[2]} + D^{[3]}. \end{aligned}$$

It can be easily verified that this complies with (9.11) for $x = 3$. For $x = 4$, using (9.10) we obtain

$$\begin{aligned} F_4 &= b_1^{[4]} F_1 + b_2^{[4]} F_2 + b_3^{[4]} F_3 + D^{[4]} \\ &= b_1^{[4]} D^{[1]} + b_2^{[4]} \left(b_1^{[2]} D^{[1]} + D^{[2]} \right) + b_3^{[4]} \left(\left(b_1^{[3]} + b_1^{[2]} b_2^{[3]} \right) D^{[1]} + b_2^{[3]} D^{[2]} + D^{[3]} \right) + D^{[4]} \\ &= \left(b_1^{[4]} + b_1^{[2]} b_2^{[4]} + b_1^{[3]} b_3^{[4]} + b_1^{[2]} b_2^{[3]} b_3^{[4]} \right) D^{[1]} + \left(b_2^{[4]} + b_2^{[3]} b_3^{[4]} \right) D^{[2]} + b_3^{[4]} D^{[3]} + D^{[4]}. \end{aligned}$$

On the other hand, using (9.11) we obtain

$$\begin{aligned} F_4 &= E^{[1,4]} D^{[1]} + E^{[2,4]} D^{[2]} + E^{[3,4]} D^{[3]} + E^{[4,4]} D^{[4]} \\ &= \left(\sum_{w=1}^{3} \sum_{1=v_0<v_1<\cdots<v_w=4} b_{v_0}^{[v_1]} b_{v_1}^{[v_2]} \cdots b_{v_{w-1}}^{[v_w]} \right) D^{[1]} \\ &\quad + \left(\sum_{w=1}^{2} \sum_{2=v_0<v_1<\cdots<v_w=4} b_{v_0}^{[v_1]} b_{v_1}^{[v_2]} \cdots b_{v_{w-1}}^{[v_w]} \right) D^{[2]} \\ &\quad + \left(\sum_{w=1}^{1} \sum_{3=v_0<v_1<\cdots<v_w=4} b_{v_0}^{[v_1]} b_{v_1}^{[v_2]} \cdots b_{v_{w-1}}^{[v_w]} \right) D^{[3]} + D^{[4]} \\ &= \left(\sum_{1=v_0<v_1=4} b_{v_0}^{[v_1]} + \sum_{1=v_0<v_1<v_2=4} b_{v_0}^{[v_1]} b_{v_1}^{[v_2]} + \sum_{1=v_0<v_1<v_2<v_3=4} b_{v_0}^{[v_1]} b_{v_1}^{[v_2]} b_{v_2}^{[v_3]} \right) D^{[1]} + \\ &\quad \left(\sum_{2=v_0<v_1=4} b_{v_0}^{[v_1]} + \sum_{2=v_0<v_1<v_2=4} b_{v_0}^{[v_1]} b_{v_1}^{[v_2]} \right) D^{[2]} + \left(\sum_{3=v_0<v_1=4} b_{v_0}^{[v_1]} \right) D^{[3]} + D^{[4]} \\ &= \left(b_1^{[4]} + b_1^{[2]} b_2^{[4]} + b_1^{[3]} b_3^{[4]} + b_1^{[2]} b_2^{[3]} b_3^{[4]} \right) D^{[1]} + \left(b_2^{[4]} + b_2^{[3]} b_3^{[4]} \right) D^{[2]} + b_3^{[4]} D^{[3]} + D^{[4]}. \end{aligned}$$

Notice that the number of terms contained in the expression $\sum_{v=v_0<v_1<\cdots<v_w=x} b_{v_0}^{[v_1]} b_{v_1}^{[v_2]} b_{v_2}^{[v_3]} \cdot \cdot \cdot b_{v_{w-1}}^{[v_w]}$, involved in the right-hand side of (9.12) is $\binom{x-v-1}{w-1}$. Thus, to determine all values of $E^{[v,x]}$, $1 \leq v \leq x-1$, $2 \leq x \leq k$, the total number of products to be computed is $\sum_{x=2}^{k} \sum_{v=1}^{x-1} \sum_{w=1}^{x-v} \binom{x-v-1}{w-1} = 2^k - k - 1$.

Recall that the durations of the RMPs are given by (9.4). Including the time spent on rate-modifying activities, the completion time $C_{\pi^{[x]}(r)}$ of a job scheduled in position r, $1 \leq r \leq n^{[x]}$, of the x-th group, $1 \leq x \leq k$, can be written as

$$\begin{aligned} C_{\pi^{[x]}(r)} &= F_1 + T_1 + F_2 + T_2 + \cdots + F_{x-1} + T_{x-1} + F_{(x,r)} \\ &= \sum_{v=1}^{x-1} \left[1 + \sum_{w=v}^{x-1} \alpha_v^{[w]} \right] F_v + F_{(x,r)} + \sum_{v=1}^{x-1} \beta^{[v]}. \end{aligned}$$

For convenience, denote

$$\xi^{[v,x-1]} := 1 + \sum_{w=v}^{x-1} \alpha_v^{[w]}, 1 \leq v \leq x-1, 1 \leq x \leq k, \tag{9.13}$$

and rewrite

$$C_{\pi^{[x]}(r)} = \sum_{v=1}^{x-1} \xi^{[v,x-1]} F_v + F_{(x,r)} + \sum_{v=1}^{x-1} \beta^{[v]} \tag{9.14}$$

Applying the results of Lemmas 9.1 and 9.2, the completion time can be written in terms of the original problem parameters.

9.3.1 Minimising The Makespan

Let us now specifically consider problem $1\,|Combi, RMP(k-1)|\,C_{\max}$. For a schedule $S(k)$ with k groups denote the makespan by $C_{\max}(S(k))$. If schedule $S(k)$ is associated with a permutation $\pi = \left(\pi^{[1]}, \pi^{[2]}, \ldots, \pi^{[k]}\right)$, then $C_{\max}(S(k))$ is equal to $C_{\pi^{[k]}\left(n^{[k]}\right)}$, the completion time of the last job in the last k-th group. From (9.14) we have

$$C_{\max}(S(k)) = C_{\pi^{[k]}\left(n^{[k]}\right)} = \sum_{x=1}^{k-1} \xi^{[x,k-1]} F_x + F_k + \Gamma(k),$$

where $\Gamma(k) = \sum_{x=1}^{k-1} \beta^{[x]}$. Substituting (9.11) in the above expression we obtain

$$C_{\max}(S(k)) = \sum_{x=1}^{k-1} \xi^{[x,k-1]} \left(\sum_{v=1}^{x} E^{[v,x]} D^{[v]} \right) + \sum_{v=1}^{k} E^{[v,k]} D^{[v]} + \Gamma(k).$$

Rearranging the terms we get

$$C_{\max}(S(k)) = \sum_{x=1}^{k} \left(\sum_{v=x}^{k-1} \xi^{[v,k-1]} E^{[x,v]} + E^{[x,k]} \right) D^{[x]} + \Gamma(k).$$

Substituting the value of $D^{[x]}$ from (9.8) we further derive

$$\begin{aligned} C_{\max}\left(S\left(k\right)\right) &= \sum_{x=1}^{k}\left(\sum_{v=x}^{k-1}\xi^{[v,k-1]}E^{[x,v]}+E^{[x,k]}\right)\sum_{r=1}^{n^{[x]}}p_{\pi^{[x]}(r)}B^{[x]}\left(r,n^{[x]}\right)+\Gamma\left(k\right) \\ &= \sum_{x=1}^{k}\sum_{r=1}^{n^{[x]}}p_{\pi^{[x]}(r)}B^{[x]}\left(r,n^{[x]}\right)\left(\sum_{v=x}^{k-1}\xi^{[v,k-1]}E^{[x,v]}+E^{[x,k]}\right)+\Gamma\left(k\right). \end{aligned}$$

Notice, that the expression for the makespan $C_{\max}\left(S\left(k\right)\right)$ has reduced to the generic objective function (4.6) defined in Chapter 6. The constant term $\Gamma\left(k\right)$ is still given by (6.9) and the positional weights are defined by

$$W^{[x]}\left(r\right)=B^{[x]}\left(r,n^{[x]}\right)\left(\sum_{v=x}^{k-1}\left(1+\sum_{w=v}^{k-1}\alpha_{v}^{[w]}\right)E^{[x,v]}+E^{[x,k]}\right),\ 1\leq r\leq n^{[x]},\ 1\leq x\leq k. \tag{9.15}$$

The expression for the positional weights, (9.15) can be written in terms of the original parameters by use of (9.6), (9.9) and (9.12). Finding all values of $B^{[x]}\left(r,n^{[x]}\right)$, $1\leq r\leq n^{[x]}$, $1\leq x\leq k$, by (9.6) requires $O\left(\sum_{x=1}^{k}n^{[x]}\right)=O(n)$ time. After that, all values of $b_{v}^{[x]}$, $1\leq v\leq x-1$, $1\leq x\leq k$, can be found by (9.9) in $O\left(k^{2}\right)$ time. As follows from Section 9.3, computing all values $E^{[v,x]}$, $1\leq v\leq x-1$, $1\leq x\leq k$, requires $O\left(2^{k}\right)$ time. Finally, all n positional weights $W^{[x]}\left(r\right)$, $1\leq r\leq n^{[x]}$, $1\leq x\leq k$, will be computed in $O\left(n\right)$ time, provided that all other quantities have been found. Thus, the overall running time needed to compute the positional weights for problem $1\left|Combi,RMP(k-1)\right|C_{\max}$ is $T\left(W\right)=O\left(2n+k^{2}+2^{k}\right)=O\left(n\right)$, provided that k is a given constant.

9.3.2 Minimising The Total Flow Time

Now we address problem $1\left|Combi,RMP(k-1)\right|\sum C_{j}$. For a schedule $S\left(k\right)$ with k groups associated with a permutation $\pi=\left(\pi^{[1]},\pi^{[2]},\ldots,\pi^{[k]}\right)$, the total flow time can be written as

$$\sum C_{j}=\sum_{x=1}^{k}\sum_{r=1}^{n^{[x]}}C_{\pi^{[x]}(r)}.$$

Substituting (9.5) into (9.14), we rewrite the expression for $C_{\pi^{[x]}(r)}$ as

$$\begin{aligned} C_{\pi^{[x]}(r)} &= \sum_{v=1}^{x-1} \xi^{[v,x-1]} F_v + \sum_{u=1}^{r} \left(A^{[x]} + p_{\pi^{[x]}(u)} \right) B^{[x]}(u,r) + \sum_{v=1}^{x-1} \beta^{[v]} \\ &= \sum_{v=1}^{x-1} \xi^{[v,x-1]} F_v + A^{[x]} \sum_{u=1}^{r} B^{[x]}(u,r) + \sum_{u=1}^{r} p_{\pi^{[x]}(u)} B^{[x]}(u,r) + \sum_{v=1}^{x-1} \beta^{[v]}. \end{aligned}$$

Substituting $A^{[x]} = \sum_{v=1}^{x-1} a_v^{[x]} F_v$, into the above equation, we obtain

$$C_{\pi^{[x]}(r)} = \sum_{v=1}^{x-1} \left(\xi^{[v,x-1]} + a_v^{[x]} \sum_{u=1}^{r} B^{[x]}(u,r) \right) F_v + \sum_{u=1}^{r} p_{\pi^{[x]}(u)} B^{[x]}(u,r) + \sum_{v=1}^{x-1} \beta^{[v]}.$$

Thus, the total flow time of schedule $S(k)$ can be written as

$$\begin{aligned} \sum C_j &= \sum_{x=1}^{k} \sum_{r=1}^{n^{[x]}} \left[\sum_{v=1}^{x-1} \left(\xi^{[v,x-1]} + a_v^{[x]} \sum_{u=1}^{r} B^{[x]}(u,r) \right) F_v + \sum_{u=1}^{r} p_{\pi^{[x]}(u)} B^{[x]}(u,r) \right] + \Gamma(k) \\ &= \sum_{x=1}^{k} \left[\sum_{v=1}^{x-1} \left(n^{[x]} \xi^{[v,x-1]} + a_v^{[x]} \sum_{r=1}^{n^{[x]}} \sum_{u=1}^{r} B^{[x]}(u,r) \right) F_v + \sum_{r=1}^{n^{[x]}} \sum_{u=1}^{r} p_{\pi^{[x]}(u)} B^{[x]}(u,r) \right] \\ &\quad + \Gamma(k), \end{aligned}$$

where $\Gamma(k) = \sum_{x=1}^{k} \sum_{r=1}^{n^{[x]}} \sum_{v=1}^{x-1} \beta^{[v]}$. It can be easily verified that the term $\sum_{r=1}^{n^{[x]}} \sum_{u=1}^{r} p_{\pi^{[x]}(u)} B^{[x]}(u,r)$ can be rewritten as $\sum_{r=1}^{n^{[x]}} p_{\pi^{[x]}(r)} \sum_{u=r}^{n^{[x]}} B^{[x]}(r,u)$, $1 \le x \le k$, so that we have

$$\sum C_j = \sum_{x=1}^{k} \left[\sum_{v=1}^{x-1} \left(n^{[x]} \xi^{[v,x-1]} + a_v^{[x]} \sum_{r=1}^{n^{[x]}} \sum_{u=r}^{n^{[x]}} B^{[x]}(r,u) \right) F_v + \sum_{r=1}^{n^{[x]}} p_{\pi^{[x]}(r)} \sum_{u=r}^{n^{[x]}} B^{[x]}(r,u) \right] + \Gamma(k).$$

For convenience, substitute the value $\xi^{[v,x-1]}$ from (9.13) and denote

$$G^{[v,x]} := n^{[x]} \left(1 + \sum_{w=v}^{x-1} \alpha_v^{[w]} \right) + a_v^{[x]} \sum_{r=1}^{n^{[x]}} \sum_{u=r}^{n^{[x]}} B^{[x]}(r,u), \ 1 \le v \le x-1, \ 1 \le x \le k, \quad (9.16)$$

so that we have

$$\sum C_j = \sum_{x=1}^{k} \left(\sum_{v=1}^{x-1} G^{[v,x]} F_v + \sum_{r=1}^{n^{[x]}} p_{\pi^{[x]}(r)} \sum_{u=r}^{n^{[x]}} B^{[x]}(r,u) \right) + \Gamma(k).$$

Substituting the value of F_v from (9.11) into the above equation, we deduce

$$\begin{aligned}
\sum C_j &= \sum_{x=1}^{k}\left(\sum_{v=1}^{x-1} G^{[v,x]} \sum_{w=1}^{v} E^{[w,v]} D^{[w]} + \sum_{r=1}^{n^{[x]}} p_{\pi^{[x]}(r)} \sum_{u=r}^{n^{[x]}} B^{[x]}(r,u)\right) + \Gamma(k) \\
&= \sum_{x=1}^{k}\left[\sum_{v=1}^{x-1}\left(\sum_{w=v}^{x-1} G^{[w,x]} E^{[v,w]}\right) D^{[v]} + \sum_{r=1}^{n^{[x]}} p_{\pi^{[x]}(r)} \sum_{u=r}^{n^{[x]}} B^{[x]}(r,u)\right] + \Gamma(k) \\
&= \sum_{x=1}^{k}\sum_{v=x+1}^{k}\left(\sum_{w=x}^{v-1} G^{[w,v]} E^{[x,w]}\right) D^{[x]} + \sum_{x=1}^{k}\sum_{r=1}^{n^{[x]}} p_{\pi^{[x]}(r)} \sum_{u=r}^{n^{[x]}} B^{[x]}(r,u) + \Gamma(k) \\
&= \sum_{x=1}^{k} D^{[x]} \sum_{v=x+1}^{k}\sum_{w=x}^{v-1} G^{[w,v]} E^{[x,w]} + \sum_{x=1}^{k}\sum_{r=1}^{n^{[x]}} p_{\pi^{[x]}(r)} \sum_{u=r}^{n^{[x]}} B^{[x]}(r,u) + \Gamma(k).
\end{aligned}$$

Further, substituting the value of $D^{[x]}$ from (9.8) and rearranging terms, we obtain

$$\begin{aligned}
\sum C_j &= \sum_{x=1}^{k}\left[\left(\sum_{r=1}^{n^{[x]}} p_{\pi^{[x]}(r)} B^{[x]}\left(r, n^{[x]}\right)\right) \sum_{v=x+1}^{k}\sum_{w=x}^{v-1} G^{[w,v]} E^{[x,w]}\right. \\
&\quad \left. + \sum_{r=1}^{n^{[x]}} p_{\pi^{[x]}(r)} \sum_{u=r}^{n^{[x]}} B^{[x]}(r,u)\right] + \Gamma(k) \\
&= \sum_{x=1}^{k}\sum_{r=1}^{n^{[x]}} p_{\pi^{[x]}(r)}\left(B^{[x]}\left(r, n^{[x]}\right) \sum_{v=x+1}^{k}\sum_{w=x}^{v-1} G^{[w,v]} E^{[x,w]} + \sum_{u=r}^{n^{[x]}} B^{[x]}(r,u)\right) + \Gamma(k).
\end{aligned}$$

Thus, the total flow time can be represented as a function of the form (4.6) with the positional weights defined by

$$W^{[x]}(r) = B^{[x]}\left(r, n^{[x]}\right) \sum_{v=x+1}^{k}\sum_{w=x}^{v-1} G^{[w,v]} E^{[x,w]} + \sum_{u=r}^{n^{[x]}} B^{[x]}(r,u), \ 1 \le r \le n^{[x]}, \ 1 \le x \le k \tag{9.17}$$

and the constant term

$$\Gamma(k) = \sum_{x=1}^{k}\sum_{r=1}^{n^{[x]}}\sum_{v=1}^{x-1} \beta^{[v]}. \tag{9.18}$$

The expression for the positional weights, (9.17) can be written in terms of the original parameters by use of (9.6), (9.16), (9.12) and (9.9). All values $B^{[x]}(u,r)$, $1 \le u \le r \le n^{[x]}, 1 \le x \le k$, can be computed by (9.6) in $O\left(\sum_{x=1}^{k} \frac{n^{[x]}\left(n^{[x]}+1\right)}{2}\right) = O(n^2)$ time. After that, all values of $G^{[v,x]}$, $1 \le v \le x-1$, $1 \le x \le k$, can be found by (9.16) in $O(k^2)$ time. As discussed in Section 9.3.1, computing all values $E^{[v,x]}, 1 \le v \le x-1$,

$1 \leq x \leq k$, requires $O\left(2^k\right)$ time. Finally, all n positional weights $W^{[x]}(r)$, $1 \leq r \leq n^{[x]}, 1 \leq x \leq k$, will be computed in $O(n)$ time, provided that all other quantities have been found. Thus, the overall running time needed to compute the positional weights for problem $1\,|Combi, RMP(k-1)|\,C_{\max}$ is $T(W) = O\left(n + n^2 + k^2 + 2^k\right) = O(n^2)$, provided that k is a given constant.

9.4 The Solution Approach

The purpose of this section, is to consider problem $1\,|Combi, RMP|\,F$ where $F \in \{C_{\max}, \sum C_j\}$ and design a solution approach for it. In order to solve problem $1\,|Combi, RMP|\,F$, we must find the optimal values for Decisions 1-4. First, let us concentrate on the relevant sub-problem $1\,|Combi, RMP(k-1)|\,F$ in which it is assumed that Decisions 1-3 are taken in advance.

In the last section we demonstrate that even without making additional assumptions regarding the sign of $a_v^{[x]}$, $1 \leq v \leq x$, $1 \leq x \leq k$, or the shape of $g^{[x]}(r)$, $1 \leq r \leq n^{[x]}$, $1 \leq x \leq k$, both problems of the form $1\,|Combi, RMP(k-1)|\,F$ reduce to minimising a linear form $\sum_{x=1}^{k}\sum_{r=1}^{n^{[x]}} W^{[x]}(r)p_{\pi^{[x]}(r)}$ over a set of all permutations. However, notice that the computed positional weights $W^{[x]}(r)$, $1 \leq r \leq n^{[x]}$, given by (9.15) and (9.17), are found to be non-monotonically ordered within each group x, $1 \leq x \leq k$. Additionally, the term $n^{[x]}$ appears in the formulae (9.15) and (9.17), and thus, it is not possible to generate a set of all possible values of $W^{[x]}(r)$, $1 \leq r \leq n$, $1 \leq x \leq k$, without prior knowledge of the number of jobs, $n^{[x]}$, in each group. As a result, Theorem 6.1 does not hold and none of the solution approaches presented in the previous chapters can be applied.

Below we describe an algorithm which solves an instance of problem $1\,|Combi, RMP(k-1)|\,F$ by guessing the values $n^{[x]}$, $1 \leq x \leq k$, in advance.

Algorithm Generate

INPUT: An instance of problem $1\,|Combi, RMP(k-1)|\,F$

OUTPUT: An optimal schedule $S^*(k)$

Step 1. If required, renumber the jobs in LPT order, i.e., in non-increasing order of the values p_j.

Step 2. Generate all possible instances in which n jobs are split into k non-empty groups, so that we have the values $n^{[x]}$, $1 \leq x \leq k$, such that $\sum_{x=1}^{k} n^{[x]} = n$.

Step 3. For each instance, compute the positional weights $W^{[x]}(r)$, $1 \leq r \leq n^{[x]}, 1 \leq x \leq k$, by (9.15) or (9.17).

Step 4. With the found values of $W^{[x]}(r)$, $1 \leq r \leq n^{[x]}, 1 \leq x \leq k$, run Algorithm Match2 and select the instance that returns the smallest value of the function $F(k)$ as an optimal solution to problem $1\,|Combi, RMP\,(k-1)|\,F$.

Notice that in order to compute the optimal values of $n^{[x]}$, $1 \leq x \leq k$, Algorithm Generate simply enumerates all possible options in which n jobs can be split into k non-empty groups. It follows from Section 2.2.4, that the number of ways of doing this is equal to $\frac{n^{k-1}}{(k-1)!}$. For each composition, Step 3 requires a running time of $T(W)$, which is already known for both problems. Step 4 requires $O(n \log n)$ time; see Chapter 6. Thus, the following statement holds.

Theorem 9.1. *Algorithm Generate solves an instance of problem* $1\,|Combi, RMP\,(k-1)|\,F$ *in* $O\left((T(W) + n\log n)\,\frac{n^{k-1}}{(k-1)!}\right)$ *time, where* $T(W)$ *denotes the time needed to compute all positional weights* $W^{[x]}(r)$, $1 \leq r \leq n^{[x]}$, $1 \leq x \leq k$, *for a given composition of* n.

To determine the optimal solution for the general problem $1\,|Combi, RMP|\,F$, all options associated with Decisions 1-3 must be enumerated. For a known k, $1 \leq k \leq K+1$, the number of ways to select $k-1$ RMPs and fix their order is equal to the number of $(k-1)$-arrangements of K elements, i.e., the number of permutations of $k-1$ elements selected from K possibilities, which is equal to $\binom{K}{k-1}(k-1)!$. Trying all possible values of k, $1 \leq k \leq K+1$, the total number of options can be approximated by $\sum_{k=1}^{K+1}\binom{K}{k-1}(k-1)!$. Since Algorithm Generate requires $O\left((T(W) + n\log n)\,\frac{n^{k-1}}{(k-1)!}\right)$ time for a given k, $1 \leq k \leq K+1$, the total running time required to solve problem $1\,|Combi, RMP|\,F$ can be approximated by

$$\begin{aligned}
\sum_{k=1}^{K+1}\binom{K}{k-1}n^{k-1}\left(T(W) + n\log n\right) &\leq \left(\sum_{k=1}^{K+1}\binom{K}{k-1}\right)\left(\sum_{k=1}^{K+1}n^{k-1}\right)\left(T(W) + n\log n\right)\\
&= \left(\sum_{k=0}^{K}\binom{K}{k}\right)\left(\sum_{k=0}^{K}n^{k}\right)\left(T(W) + n\log n\right)\\
&= 2^K\left(\frac{n^{K+1}-1}{n-1}\right)\left(T(W) + n\log n\right)\\
&= O\left(n^K\left(T(W) + n\log n\right)\right).
\end{aligned}$$

Recall from Sections 9.3.1 and 9.3.2 that for the problems of minimising the makespan and the total flow time, the required positional weights $W^{[x]}(r)$, $1 \leq r \leq n^{[x]}$,

$1 \leq x \leq k$, can be computed in $T(W) = O(n)$ and $T(W) = O(n^2)$ time, respectively. It follows that an optimal solution to problem $1|Combi, RMP|C_{\max}$ can be found in $O\left(n^K (n + n\log n)\right) = O\left(n^{K+1}\log n\right)$ time, and an optimal solution to problem $1|Combi, RMP|\sum C_j$ can be found in $O\left(n^K (n^2 + n\log n)\right) = O\left(n^{K+2}\right)$ time.

Below we provide a numerical example, in which we solve the problems of minimising the makespan and the total flow time for the situation discussed in Example 1. We present the solution for known outcomes of Decisions 1-3 and a given composition of n.

Example 9.2: Eight jobs must be processed on a single machine. The normal processing times of the jobs, after renumbering them in LPT order are:

$$p_1 = 8,\ p_2 = 7,\ p_3 = 6,\ p_4 = 6,\ p_5 = 4,\ p_6 = 2,\ p_7 = 1,\ p_8 = 1.$$

The number of jobs in each of the three groups is known in advance as

$$n^{[1]} = 3,\ n^{[2]} = 2,\ n^{[3]} = 3.$$

The values of the parameters for the machine and the operators as defined in Example 9.1 are also known and given by

$$\begin{aligned} d'_m &= 0.1,\ d''_m = 0.15,\ d'_w = 0.2,\ l'_w = -0.15,\ d''_w = 0.15,\ l''_w = -0.1,\ l'''_w = -0.02, \\ \alpha' &= 2,\ \beta' = 5,\ \beta'' = 6. \end{aligned}$$

Using the formulae provided in Table 9.1, the functions for the positional factors reduce to

$$\begin{aligned} g^{[1]}(r) &= (d'_m + 1)^{r-1} r^{d'_w + l'_w} = (1.1)^{r-1} r^{0.05},\ 1 \leq r \leq 3; \\ g^{[2]}(r) &= (d''_m + 1)^{r-1} \left(n^{[1]} + r\right)^{l'_w} r^{d'_w} = (1.15)^{r-1} \frac{r^{0.2}}{(r+3)^{0.15}},\ 1 \leq r \leq 2; \\ g^{[3]}(r) &= (d''_m + 1)^{n^{[2]} + r - 1} r^{d''_w + l''_w} = (1.15)^{r+1} r^{0.05},\ 1 \leq r \leq 3, \end{aligned}$$

and all required input parameters are computed as

$$\begin{aligned} &g^{[1]}(1) = 1.00,\ g^{[1]}(2) = 1.14,\ g^{[1]}(3) = 1.28; && a_1^{[1]} = 0.15; \\ &g^{[2]}(1) = 0.81,\ g^{[2]}(2) = 1.04; && a_1^{[2]} = -0.15,\ a_2^{[2]} = 0.20; \\ &g^{[3]}(1) = 1.32,\ g^{[3]}(2) = 1.57,\ g^{[3]}(3) = 1.85; && a_1^{[3]} = -0.02,\ a_2^{[3]} = 0.13,\ a_3^{[3]} = 0.20. \end{aligned}$$

Group 1	$B^{[1]}(1,1) = 1.00;$		
	$B^{[1]}(1,2) = 1.17,$	$B^{[1]}(2,2) = 1.14;$	
	$B^{[1]}(1,3) = 1.40,$	$B^{[1]}(2,3) = 1.36,$	$B^{[1]}(3,3) = 1.28;$
Group 2	$B^{[2]}(1,1) = 0.81;$		
	$B^{[2]}(1,2) = 0.98,$	$B^{[2]}(2,2) = 1.04;$	
Group 3	$B^{[3]}(1,1) = 1.32;$		
	$B^{[3]}(1,2) = 1.74,$	$B^{[3]}(2,2) = 1.57;$	
	$B^{[3]}(1,3) = 2.38,$	$B^{[3]}(2,3) = 2.16,$	$B^{[3]}(3,3) = 1.85.$

Table 9.2: Values of $B^{[x]}(u,r), 1 \leq u \leq r \leq n^{[x]}, 1 \leq x \leq 3$, for Example 9.2

and

$$\alpha_1^{[1]} = 2,\ \beta^{[1]} = 5,\ \alpha_1^{[2]} = \alpha_2^{[2]} = 0,\ \beta^{[2]} = 6.$$

Notice that while solving this example, we have computed all values with high precision, but here and below for the ease of presentation we report them rounded to two decimal places. Applying (9.6), all values of $B^{[x]}(u,r)$, $1 \leq u \leq r \leq n^{[x]}$, $1 \leq x \leq 3$, are computed and are presented in Table 9.2. For the problem of minimising the makespan, we will only need to use the values given in the last row of each block in Table 9.2, whereas for the problem of minimising the total flow time we will require all of them.

Applying (9.9), all values of $b_v^{[x]}$, $1 \leq v \leq x-1$, $1 \leq x \leq 3$, are computed as

$$\begin{aligned} b_1^{[2]} &= (-0.15)(0.98 + 1.04) = -0.30; \\ b_1^{[3]} &= (-0.02)(2.38 + 2.16 + 1.85) = -0.13, \\ b_2^{[3]} &= (0.13)(2.38 + 2.16 + 1.85) = 0.83. \end{aligned}$$

Applying (9.12), all values of $E^{[v,x]}, 1 \leq v \leq x, 1 \leq x \leq 3$, are computed as

$$\begin{aligned} E^{[1,1]} &= 1.00; \\ E^{[1,2]} &= b_1^{[2]} = -0.30,\ E^{[2,2]} = 1.00; \\ E^{[1,3]} &= b_1^{[3]} + b_1^{[2]} b_2^{[3]} = -0.38,\ E^{[2,3]} = b_2^{[3]} = 0.83,\ E^{[3,3]} = 1.00. \end{aligned}$$

Finally, applying (9.15), the positional weights $W^{[x]}(r), 1 \leq r \leq n^{[x]}, 1 \leq x \leq 3$,

for the problem of minimising the makespan can be rewritten as

$$\begin{aligned}
W^{[1]}(r) &= B^{[1]}(r,3)\left(\left(1+\alpha_1^{[1]}+\alpha_1^{[2]}\right)E^{[1,1]}+\left(1+\alpha_2^{[2]}\right)E^{[1,2]}+E^{[1,3]}\right),\ 1\le r\le 3;\\
W^{[2]}(r) &= B^{[2]}(r,2)\left(\left(1+\alpha_2^{[2]}\right)E^{[2,2]}+E^{[2,3]}\right),\ 1\le r\le 2;\\
W^{[3]}(r) &= B^{[3]}(r,3)\left(E^{[3,3]}\right),\ 1\le r\le 3,
\end{aligned}$$

and their values computed as

$$\begin{aligned}
W^{[1]}(1) &= 3.23,\ W^{[1]}(2)=3.15,\ W^{[1]}(3)=2.96;\\
W^{[2]}(1) &= 1.80,\ W^{[2]}(2)=1.90;\\
W^{[3]}(1) &= 2.38,\ W^{[3]}(2)=2.16,\ W^{[3]}(3)=1.85.
\end{aligned}$$

To solve problem $1\,|Combi, RMP(k-1)|\,C_{\max}$, the computed positional weights are sorted in non-decreasing order and stored in list $L' := (\gamma'_1, \gamma'_2, \ldots, \gamma'_8)$. The constant term $\Gamma(3)$, can be computed as $\Gamma(3) = \beta^{[1]} + \beta^{[2]} = 11.00$. The resulting optimal schedule $S'(3)$ is associated with a permutation $\mu^* = \left(\mu^{[1]}, \mu^{[2]}, \mu^{[3]}\right)$; all relevant computation is presented in Table 9.3.

Next, for the problem of minimising the total flow time, we also must additionally compute the values of $G^{[v,x]}$, $1 \le v \le x-1$, $1 \le x \le 3$. Using (9.16) and the values obtained in Table 9.2, we compute

$$G^{[1,2]} = 5.58,\ G^{[1,3]} = 8.78,\ G^{[2,3]} = 4.43.$$

Applying (9.17), the positional weights $W^{[x]}(r), 1 \le r \le n^{[x]}, 1 \le x \le 3$, for the problem of minimising the total flow time can be rewritten as

$$\begin{aligned}
W^{[1]}(r) &= B^{[1]}(r,3)\left(G^{[1,2]}E^{[1,1]}+G^{[1,3]}E^{[1,1]}+G^{[2,3]}E^{[1,2]}\right)+\sum_{u=r}^{3}B^{[1]}(r,u),\ 1\le r\le 3;\\
W^{[2]}(r) &= B^{[2]}(r,2)\left(G^{[2,3]}E^{[2,2]}\right)+\sum_{u=r}^{2}B^{[2]}(r,u),\ 1\le r\le 2;\\
W^{[3]}(r) &= \sum_{u=r}^{3}B^{[3]}(r,u),\ 1\le r\le 3,
\end{aligned}$$

j	p_j	γ'_j	μ^*	$\gamma'_j p_j$	γ''_j	φ^*	$\gamma''_j p_j$
		Makespan			Flow time		
1	8	1.80	$\mu^{[2]}(1)$	14.36	1.85	$\varphi^{[3]}(3)$	14.78
2	7	1.85	$\mu^{[3]}(3)$	12.93	3.73	$\varphi^{[3]}(2)$	26.12
3	6	1.90	$\mu^{[2]}(2)$	11.39	5.44	$\varphi^{[3]}(1)$	32.66
4	6	2.16	$\mu^{[3]}(2)$	12.94	5.64	$\varphi^{[2]}(2)$	33.82
5	4	2.38	$\mu^{[3]}(1)$	9.53	6.14	$\varphi^{[2]}(1)$	24.56
6	2	2.96	$\mu^{[1]}(3)$	5.93	17.91	$\varphi^{[1]}(3)$	35.83
7	1	3.15	$\mu^{[1]}(2)$	3.15	20.16	$\varphi^{[1]}(2)$	20.16
8	1	3.23	$\mu^{[1]}(1)$	3.23	21.72	$\varphi^{[1]}(1)$	21.72
		Total		73.46	Total		209.65
$C_{\max}(S'(3)) = 73.46 + 11.00 = 84.46;$							
$\sum C_j(S''(3)) = 209.65 + 43.00 = 252.65;$							
$\mu^* = (8,7,6,1,3,5,4,2)\,;\ \varphi^* = (8,7,6,5,4,3,2,1)$							

Table 9.3: Calculation of the optimal value of the makespan and the optimal value of the total flow time for the problem outlined in Example 2

and their values computed as

$$\begin{aligned} W^{[1]}(1) &= 21.72,\ W^{[1]}(2) = 20.16,\ W^{[1]}(3) = 17.91; \\ W^{[2]}(1) &= 6.14,\ W^{[2]}(2) = 5.64; \\ W^{[3]}(1) &= 5.44,\ W^{[3]}(2) = 3.73,\ W^{[3]}(3) = 1.85. \end{aligned}$$

To solve problem $1\,|Combi, RMP(k-1)|\sum C_j$, the computed positional weights are sorted in non-decreasing order and stored in list $L'' := (\gamma''_1, \gamma''_2, \ldots, \gamma''_n)$. The constant term $\Gamma(3)$, can be computed as $\Gamma(3) = \beta^{[1]} n^{[2]} + \left(\beta^{[1]} + \beta^{[2]}\right) n^{[3]} = 43.00$. The resulting optimal schedule $S''(3)$ is associated with a permutation $\varphi^* = \left(\varphi^{[1]}, \varphi^{[2]}, \varphi^{[3]}\right)$; all relevant computation is presented in Table 9.3.

9.5 Some Reduced Models

In this section, we explore single machine models which can be expressed as special cases of the general problems $1\,|Combi, RMP|\,C_{\max}$ and $1\,|Combi, RMP|\sum C_j$. We look at their simplified versions with effects less general than that given by (9.3) and (9.4), and possibly with additional simplifications. The main purpose of this section, is to verify whether the running time of the solution approach given in Section 9.4 for the general problems $1\,|Combi, RMP|\,C_{\max}$ and $1\,|Combi, RMP|\sum C_j$ can be reduced for their simplified counterparts.

Let us begin our consideration with the problems studied by Yang (2010) and Yang (2012). Recall from Section 3.3.2 that both of these papers study a simple combined effect of the form (9.1). The former paper considers a problem in which the system undergoes time-dependent learning, positional polynomial deterioration and a single start-time dependent maintenance period, while the latter considers a problem in which the system undergoes time-dependent deterioration, positional polynomial learning and K identical start-time dependent maintenance periods, all of which must be run. Both these models assume that the effects are group-independent and after an RMP, the learning advantages are lost and both the operator and the machine are brought to the original conditions; these assumptions can hardly be justified. Notice that for both problems, Decisions 1-3 need not be taken as the RMPs are identical and their number to be included in a schedule is already known in advance. As a result, both these problems can be written as special cases of problem $1\left|Combi, RMP\left(k-1\right)\right|F$ with the parameters $g^{[x]}\left(r\right)=r^{b}$, and $a_{1}^{[x]}=a_{2}^{[x]}=...=a_{x-1}^{[x]}=0$, $a_{x}^{[x]}=a$, for all x, $1\leq x\leq K+1$, and $\alpha_{1}^{[x]}=\alpha_{2}^{[x]}=...=\alpha_{x-1}^{[x]}=0$, $\alpha_{x}^{[x]}=\alpha$, and $\beta^{[x]}=\beta$, for all x, $1\leq x\leq K$. For the former case studied by Yang (2010), we have the additional conditions $K=1$, $b>0$, and $a<0$, while for the latter we have $b<0$ and $a>0$. For both models, the required running time for the problem of minimising the makespan and the problem of minimising the total flow time is given as $O\left(n^{K+1}\log n\right)$ and $O\left(n^{K+2}\right)$ respectively; see Section 3.3.2. Notice that these running times coincide with the ones we obtain for solving the general problems $1\left|Combi, RMP\right|C_{\max}$ and $1\left|Combi, RMP\right|\sum C_{j}$, respectively.

We notice, however, that a small reduction in the running time can be achieved if a and b are of the same sign. In this case, for the problem of minimising the makespan the resulting positional weights

$$W^{[x]}(r)=\begin{cases}r^{b}\prod\limits_{i=r+1}^{n^{[x]}}\left(1+ai^{b}\right)\left(1+\alpha\right), & 1\leq r\leq n^{[x]},\ 1\leq x\leq K;\\ r^{b}\prod\limits_{i=r+1}^{n^{[x]}}\left(1+ai^{b}\right) & 1\leq r\leq n^{[x]},\ x=K+1,\end{cases}$$

are monotonically ordered in each group, and thus, sorting all positional weights in a non-decreasing order requires $O\left(n\min\left\{K,\log n\right\}\right)$ time. As a result, Step 4 of Algorithm Generate requires $O\left(n\min\left\{K,\log n\right\}\right)$ time instead of $O\left(n\log n\right)$ time and the overall running time of the problem can be given as $O\left(\left(n+n\min\left\{K,\log n\right\}\right)\frac{n^{K}}{(K)!}\right)=O\left(n^{K+1}\min\left\{K,\log n\right\}\right)$. The same observation also holds for the problem of minimising the total flow time, for a special case in which a and b are of the same sign. However,

a smaller running time is not possible in the latter case, as Step 3 of Algorithm Generate still requires $O(n^2)$ time.

Hence, it can be seen that as long as a model incorporates a combined time-dependent and a positional effect, even the simplest versions of the general problems $1|Combi, RMP|C_{\max}$ and $1|Combi, RMP|\sum C_j$ require $O\left(n^{K+1}\log n\right)$ and $O\left(n^{K+2}\right)$ time, respectively. However, if a pure positional effect, or a pure time-dependent effect is considered, faster solutions are possible.

9.5.1 Pure Positional Effects

In this section, we discuss problems with a pure positional effect, so that for known outcomes of Decisions 1-3, the actual processing time of a job j scheduled in position r of a group x, $1 \leq x \leq k$, is given by (6.1). We consider such a model in Chapter 6 for a deterioration environment, in which the positional factors $g^{[x]}(r)$ are in a non-decreasing order (6.2) and the RMPs are strictly maintenance periods. Moreover, we do not allow the positional factors to be dependent on the number of jobs scheduled in previous groups. In this section, we study positional effects without these restrictions, so that the positional factors can be non-monotone within a group and the RMPs can be of an arbitrary nature with their durations given by (9.4). In other words, we consider position-dependent models that combine deterioration and learning effects with rate-modifying activities. An illustration of such a scenario is presented in Example 4.1. Let us denote problems of this type by $1|Posi, RMP|F$, where $F \in \{C_{\max}, \sum C_j\}$.

Problem $1|Posi, RMP|F$ can be seen as a special case of problem $1|Combi, RMP|F$. Assuming that Decisions 1-3 are taken in advance, the resulting problem $1|Posi, RMP(k-1)|F$ can be written in the form of problem $1|Combi, RMP(k-1)|F$ with $a_1^{[x]} = a_2^{[x]} = ... = a_{x-1}^{[x]} = a_x^{[x]} = 0$, for all x, $1 \leq x \leq k$. To solve problem $1|Posi, RMP(k-1)|C_{\max}$, the required positional weights can be obtained by making relevant substitutions in (9.15), so that we have

$$W^{[x]}(r) = \begin{cases} \left(1 + \sum_{w=x}^{k-1} \alpha_x^{[w]}\right) g^{[x]}(r), & 1 \leq r \leq n^{[x]},\ 1 \leq x \leq k-1, \\ g^{[x]}(r) & 1 \leq r \leq n^{[x]},\ x = k. \end{cases} \tag{9.19}$$

Notice that all positional weights $W^{[x]}(r)$, $1 \leq r \leq n^{[x]}$, $1 \leq x \leq k$, can be computed by (9.19) in $T(W) = O(n)$ time. Similarly, to solve problem $1|Posi, RMP(k-1)|\sum C_j$, the required positional weights can be obtained by making

relevant substitutions in (9.17), so that we have

$$W^{[x]}(r) = \begin{cases} \left[\sum_{v=x+1}^{k} n^{[v]} \left(1 + \sum_{w=x}^{v-1} \alpha_x^{[w]}\right) + \left(n^{[x]} - r + 1\right)\right] g^{[x]}(r), & 1 \leq r \leq n^{[x]}, \\ & 1 \leq x \leq k-1, \\ \left(n^{[x]} - r + 1\right) g^{[x]}(r), & 1 \leq r \leq n^{[x]}, \\ & x = k. \end{cases} \tag{9.20}$$

Again, all positional weights $W^{[x]}(r)$, $1 \leq r \leq n^{[x]}$, $1 \leq x \leq k$, can be computed by (9.17) in $T(W) = O(n)$ time. Notice that the computed positional weights for both problems are non-monotonically ordered within each group and do not allow a set to be found in advance. Thus, Algorithm Generate must be used so that an optimal solution to each of the problems $1|Posi, RMP(k-1)|C_{\max}$ and $1|Posi, RMP(k-1)|\sum C_j$ can be found in $O\left(\frac{n^k \log n}{(k-1)!}\right)$ time; see Theorem 9.1. Trying all possible options for Decisions 1-3, an optimal solution to each of the problems $1|Posi, RMP|C_{\max}$ and $1|Posi, RMP|\sum C_j$ can be found in $O\left(n^{K+1} \log n\right)$ time.

It can be noted that although there is no change in status for the problem of minimising the makespan, the problem of minimising the total flow time is solved faster for a model with a pure positional effect. This speed up is possible because the time taken to compute the positional weights $W^{[x]}(r)$, $1 \leq r \leq n^{[x]}$, $1 \leq x \leq k$, for problem $1|Posi, RMP(k-1)|\sum C_j$ is $T(W) = O(n)$, as opposed to $T(W) = O(n^2)$ required for problem $1|Combi, RMP(k-1)|\sum C_j$.

Recall from Section 3.3.1 that Yang and Yang (2010b) achieve the same running time of $O\left(n^{K+1} \log n\right)$ for the problem of minimising the total flow time for a much simpler model. In fact, their problem can be written as a special case of our problem $1|Posi, RMP|\sum C_j$, with a group-independent polynomial deterioration effect, i.e., $g^{[x]}(r) = r^a$, $a > 0$, and K identical maintenance periods that have start-time dependent durations, i.e., $\alpha_1^{[x]} = \alpha_2^{[x]} = ... = \alpha_{x-1}^{[x]} = 0$, $\alpha_x^{[x]} = \alpha$, $\beta^{[x]} = \beta$, for all x, $1 \leq x \leq K$.

Let us now extend our consideration to problems in which a pure job-dependent positional effect (7.1) is observed. Recall that we consider such a model in Chapter 7 for a deterioration environment, in which the positional factors $g_j^{[x]}(r)$ are in a non-decreasing order (7.2) and the RMPs are strictly maintenance periods. In this section, we study positional effects without these restrictions and denote the problem by $1|Posi\text{-JD}, RMP|F$, where $F \in \{C_{\max}, \sum C_j\}$. It can be easily verified by analogy to the job-independent case studied above, that if Decisions 1-3 are assumed to be taken, the resulting problem $1|Posi\text{-JD}, RMP(k-1)|F$ reduces to minimising a generic objective function of the form (4.4). For problem $1|Posi\text{-JD}, RMP(k-1)|C_{\max}$ the con-

stant term is given by (6.9) and for each $j \in N$, the job-dependent positional weights are given by

$$W_j^{[x]}(r) = \begin{cases} \left(1 + \sum_{w=x}^{k-1} \alpha_x^{[w]}\right) g_j^{[x]}(r), & 1 \leq r \leq n^{[x]},\ 1 \leq x \leq k-1, \\ g_j^{[x]}(r) & 1 \leq r \leq n^{[x]},\ x = k. \end{cases}$$

For problem $1\,|Posi\text{-JD}, RMP\,(k-1)|\sum C_j$ the constant term is given by (9.18) and for each $j \in N$, the job-dependent positional weights are given by

$$W_j^{[x]}(r) = \begin{cases} \left[\sum_{v=x+1}^{k} n^{[v]} \left(1 + \sum_{w=x}^{v-1} \alpha_x^{[w]}\right) + \left(n^{[x]} - r + 1\right)\right] g_j^{[x]}(r), & 1 \leq r \leq n^{[x]}, \\ & 1 \leq x \leq k-1, \\ \left(n^{[x]} - r + 1\right) g_j^{[x]}(r), & 1 \leq r \leq n^{[x]}, \\ & x = k. \end{cases}$$

Since the computed positional weights for both problems are non-monotone within each group, none of the solution approaches presented in Chapter 7 can be applied. Recall from Section 7.2, however, that if the number of jobs in each group is known in advance, a function of the form (4.4) can be minimised by reducing it to an $n \times n$ linear assignment problem of the form (2.2) with the cost function given by (4.5). Thus, to solve problem $1\,|Posi\text{-JD}, RMP\,(k-1)|\,F$, where $F \in \{C_{\max}, \sum C_j\}$, generate all possible compositions of n into k summands, and for each instance compute the cost function $c_{j,(x,r)} = p_j W_j^{[x]}(r)$, $1 \leq r \leq n^{[x]}$, $1 \leq x \leq k$, $j \in N$, by substituting the relevant values of the positional weights. Notice that for both problems all values of the cost function $c_{j,(x,r)}$, $1 \leq r \leq n^{[x]}$, $1 \leq x \leq k$, $j \in N$, can be computed in $T(W) = O(n^2)$ time each. The resulting LAP can be solved in $O(n^3)$ time by the Hungarian Algorithm.

Since there are a total of $\frac{n^{k-1}}{(k-1)!}$ compositions of n, an optimal solution to each of the problems $1\,|Posi\text{-JD}, RMP\,(k-1)|\,C_{\max}$ and $1\,|Posi\text{-JD}, RMP\,(k-1)|\sum C_j$ can be found in $O\left(\frac{n^{k+2}}{(k-1)!}\right)$ time. Trying all possible options for Decisions 1-3, an optimal solution to each of the problems $1\,|Posi\text{-JD}, RMP|\,C_{\max}$ and $1\,|Posi\text{-JD}, RMP|\sum C_j$ can be found in $O\left(n^{K+3}\right)$ time.

Recall from Section 3.3.1, that Yang and Yang (2010b) and Ji and Cheng (2010) achieve the same running time of $O\left(n^{K+3}\right)$ for the problem of minimising the total flow time for much simpler models. Yang and Yang (2010b) consider a model with a group-independent polynomial deterioration effect, i.e., $g_j^{[x]}(r) = r^{a_j}$, $a > 0$, $j \in N$, and K identical maintenance periods that have start-time dependent durations, i.e., $\alpha_1^{[x]} = \alpha_2^{[x]} = ... = \alpha_{x-1}^{[x]} = 0$, $\alpha_x^{[x]} = \alpha$, $\beta^{[x]} = \beta$, for all x, $1 \leq x \leq K$. Ji and Cheng

(2010) consider a slightly more complicated model with a group-dependent polynomial learning effect, so that we have $g_j^{[x]}(r) = \lambda_j^{[x]} \left(\sum_{y=1}^{x-1} n^{[y]} + r \right)^{a_j}$, $a_j < 0$, $j \in N$, where $\lambda_j^{[x]}$, $0 < \lambda_j^{[x]} \leq 1$, $1 \leq x \leq k$, $j \in N$, represents a job-dependent group-dependent learning factor with $\lambda_j^{[1]} = 1$, $j \in N$. Notice that the positional factors in this model are dependent on the number of jobs scheduled in previous groups. This indicates a continuous learning process across groups. The RMPs are included to further enhance the learning capabilities of the operator. Further, the duration of each RMP is a constant so that we have $\alpha_1^{[x]} = \alpha_2^{[x]} = ... = \alpha_{x-1}^{[x]} = 0$, $\alpha_x^{[x]} = \alpha$, for all x, $1 \leq x \leq K$.

9.5.2 Pure Time-Dependent Effects

In this section, we discuss problems with a pure time-dependent effect, so that for known outcomes of Decisions 1-3,the actual processing time of a job j scheduled in position r of a group x, $1 \leq x \leq k$, is given by

$$p_j^{[x]}(r) = p_{\pi^{[x]}(r)} + a_1^{[x]} F_1 + a_2^{[x]} F_2 + ... + a_{x-1}^{[x]} F_{x-1} + a_x^{[x]} F_{(x,r-1)}, \ 1 \leq r \leq n, \ 1 \leq x \leq k. \tag{9.21}$$

We consider a reduced form of the above time-dependent model in Chapter 8, in which we do not allow the duration of previous groups to affect the actual processing time of the current job, so that $a_1^{[x]} = a_2^{[x]} = ... = a_{x-1}^{[x]} = 0$, for all x, $1 \leq x \leq k$. Moreover, we only study a deterioration environment, so that $a_x^{[x]} > 0$, $1 \leq x \leq k$, and the RMPs are strictly maintenance periods. In this section, we do not impose these restrictions, so that the rates $a_1^{[x]}, a_2^{[x]}, ..., a_x^{[x]}$, $1 \leq x \leq k$, can be of an arbitrary sign and the RMPs can be of an arbitrary nature with their durations given by (9.4). Denote such a problem by $1 \left| Time, RMP \right| F$, where $F \in \{C_{\max}, \sum C_j\}$.

Problem $1 \left| Time, RMP \right| F$ can be seen as a special case of problem $1 \left| Combi, RMP \right| F$. Assuming that Decisions 1-3 are taken in advance, the resulting problem $1 \left| Time, RMP\left(k-1\right) \right| F$ can be written in the form of problem $1 \left| Combi, RMP\left(k-1\right) \right| F$ with $g^{[x]}(r) = 1$, $1 \leq r \leq n$, $1 \leq x \leq k$. To solve problem $1 \left| Time, RMP\left(k-1\right) \right| C_{\max}$, the required positional weights can be obtained by making relevant substitutions in (9.15), so that we have

$$W^{[x]}(r) = \left(1 + a_x^{[x]}\right)^{n^{[x]} - r} \left(\sum_{v=x}^{k-1} \left(1 + \alpha^{[v]}\right) E^{[x,v]} + E^{[x,k]} \right), \ 1 \leq r \leq n^{[x]}, \ 1 \leq x \leq k, \tag{9.22}$$

where the quantities $E^{[v,x]}$, $1 \leq v \leq x-1$, $1 \leq x \leq k$, are given by (9.12), and by (9.9)

the quantities $b_v^{[x]}$ reduce to

$$b_v^{[x]} = \frac{a_v^{[x]}}{a_x^{[x]}} \left(\left(1 + a_x^{[x]}\right)^{n^{[x]}} - 1 \right), \ 1 \leq v \leq x-1, \ 1 \leq x \leq k.$$

Notice that all positional weights $W^{[x]}(r)$, $1 \leq r \leq n^{[x]}$, $1 \leq x \leq k$, can be computed by (9.22) in $T(W) = O(n)$ time. Similarly, to solve problem $1\,|Time, RMP\,(k-1)|\sum C_j$, the required positional weights can be obtained by making relevant substitutions in (9.17), so that we have

$$W^{[x]}(r) = \left(1 + a_x^{[x]}\right)^{n^{[x]}-r} \sum_{v=x+1}^{k} \sum_{w=x}^{v-1} G^{[w,v]} E^{[x,w]} + \sum_{u=r}^{n^{[x]}} B^{[x]}(r,u), \ 1 \leq r \leq n^{[x]}, \ 1 \leq x \leq k, \tag{9.23}$$

where the quantities $G^{[v,x]}$, $1 \leq v \leq x-1$, $1 \leq x \leq k$, are given by (9.16), and by (9.6) the quantities $B^{[x]}(r,u)$ reduce to

$$B^{[x]}(u,r) = \left(1 + a_x^{[x]}\right)^{r-u}, \ 1 \leq u \leq r \leq n^{[x]}, \ 1 \leq x \leq k.$$

For a fixed x, $1 \leq x \leq k$, the difference $r-u$ takes at most $n^{[x]}-1$ values, so that at most $n^{[x]}-1$ distinct values of $B^{[x]}(u,r)$ need to be computed. Summing up for all x, we deduce that the number of all distinct values $B^{[x]}(u,r)$ to be found in order to compute the positional weights $W^{[x]}(r)$ is $O(n)$. As a result, all positional weights $W^{[x]}(r)$, $1 \leq r \leq n^{[x]}$, $1 \leq x \leq k$, can be computed by (9.23) in $T(W) = O(n)$ time.

Notice that the computed positional weights for both problems are non-monotonically ordered within each group and do not allow a set to be found in advance. Thus, Algorithm Generate must be used so that an optimal solution to each of the problems $1\,|Time, RMP\,(k-1)|\,C_{\max}$ and $1\,|Time, RMP\,(k-1)|\sum C_j$ can be found in $O\left(\frac{n^k \log n}{(k-1)!}\right)$ time. Trying all possible options for Decisions 1-3, an optimal solution to each of the problems $1\,|Time, RMP|\,C_{\max}$ and $1\,|Time, RMP|\sum C_j$ can be found in $O\left(n^{K+1}\log n\right)$ time.

It can be noted that although there is no change in status for the problem of minimising the makespan, the problem of minimising the total flow time is solved faster for a model with a pure time-dependent effect. This speed up is possible because the time taken to compute the positional weights $W^{[x]}(r)$, $1 \leq r \leq n^{[x]}$, $1 \leq x \leq k$, for problem $1\,|Time, RMP\,(k-1)|\sum C_j$ is $T(W) = O(n)$, as opposed to $T(W) = O(n^2)$ required for problem $1\,|Combi, RMP\,(k-1)|\sum C_j$.

Recall from Section 3.3.2 that Yang and Yang (2010b) achieve the same running

time of $O\left(n^{K+1}\log n\right)$ for the problem of minimising the total flow time for a much simpler model. In fact, their problem can be written as a special case of our problem $1\,|Time, RMP|\sum C_j$, with group-independent deterioration rates, i.e., $a_1^{[x]} = a_2^{[x]} = ... = a_{x-1}^{[x]} = a_x^{[x]} = 0$, and $a_x^{[x]} = a$, $a > 0$, for all x, $1 \leq x \leq K+1$, and K identical maintenance periods that have start-time dependent durations, i.e., $\alpha_1^{[x]} = \alpha_2^{[x]} = ... = \alpha_{x-1}^{[x]} = 0$, $\alpha_x^{[x]} = \alpha$, $\beta^{[x]} = \beta$, for all x, $1 \leq x \leq K$.

9.6 Conclusion

The chapter addresses single machine scheduling problems to minimise the makespan and the total flow time. A very general model for changing processing times is introduced, in which the actual processing times of the jobs depend on both the position and the start-time of a job in the schedule. Unlike most other papers in which changing processing times are considered, we do not insist on monotone effects. Our main motivation behind this chapter was to create a model which is most general and stills allows a polynomial ($O\left(n^{K+1}\log n\right)$ or $O\left(n^{K+2}\right)$) running time. Previous studies on combined effects also resulted in the same running times, but were only able to handle very specific cases. The model introduced in this chapter covers all previously known models, provided that the introduced effects are job-independent.

We reduce these generalised problems to linear assignment problems with product matrices, solvable by a matching algorithm, and present close form relations for computing the necessary input parameters, such as positional weights. We observe that if a pure positional or time-dependent effect is considered, the problem of minimising the total flow time can be solved in $O\left(n^{K+1}\log n\right)$ time, instead of $O\left(n^{K+2}\right)$ time. For a pure positional model, we also solve a job-dependent version of the problem, which utilises reduction to a full form assignment problem.

It should be noted that many results discussed in this chapter can be transferred to other objective functions with no major technical difficulties, since we have provided the formula for the completion time of an arbitrary job in the schedule. A further extension of the main model of this chapter would involve problems with changing processing times subject to job-dependent combined effects. For these models, the main algorithmic tool is expected to be a full form assignment problem.

CHAPTER 10

Cumulative Effects and Rate-Modifying Activities

In this chapter, we discuss single machine scheduling problems with cumulative effects and rate-modifying activities. We consider a very basic model, in which a machine is subject to cumulative deterioration and the decision-maker decides when to schedule a single maintenance period that completely restores the processing conditions. The only objective function considered is the makespan. We link the problem to the Subset-sum problem (if the duration of maintenance is constant) and to the Half-product problem (if the duration of maintenance depends on its start time). For both versions of the problem, we adapt the existing fully polynomial-time approximation schemes to our problems by handling the additive constants. The duration of the MP is either a constant or is start-time dependent.

The results of this chapter have been published in our paper Kellerer, Rustogi and Strusevich (2012c). The presentation of content in the paper is very similar to the content provided in this chapter.

10.1 Overview of the Problem

Notice that makespan minimisation problems with multiple RMPs and positional or/and time-dependent effects are polynomially solvable even in the most general settings, see Chapters 5-8. However, in the case of a cumulative deterioration effect of a fairly simple structure with only a single MP, the problem under consideration is NP-hard. In this chapter, we concentrate on the design of approximation schemes for the given problem.

The approach that we pursue in this study is based on linking the corresponding

scheduling problem to problems of Boolean programming; see Section 2.2.3. In particular, for the problem with a constant MP, we show that the variable part of the objective function is related to the Subset-sum problem, see Kellerer, Pferschy and Pisinger (2004). On the other hand, if the duration of the MP depends linearly on its start time (4.1), we establish its link to a problem of quadratic Boolean programming, known as the Half-product problem, see Badics and Boros (1998) and Kellerer and Strusevich (2012). Although each of the mentioned Boolean programming problems admits an FPTAS, a challenge remains to adapt such an FPTAS to handling the original objective function. The latter task is not straightforward due to the presence of an additive constant of the sign that is opposite to the sign of the variable part of the function.

To illustrate this, consider a function of the form

$$F(\mathbf{x}) = G(\mathbf{x}) + K > 0,$$

where $G(\mathbf{x})$ represents a variable part of the overall function $F(\mathbf{x})$ to be minimised, and K is a constant. If $\mathbf{x}^*$ minimises the function $G(\mathbf{x})$, it will obviously minimise the function $F(\mathbf{x})$ as well. Suppose that for minimising function $G(\mathbf{x})$ an FPTAS is available that delivers a solution $\mathbf{x}^H$, such that $G(\mathbf{x}^H) - G(\mathbf{x}^*) \leq \varepsilon \left|G(\mathbf{x}^*)\right|$.

Recall from Section 2.2.2, that for $\mathbf{x}^H$ to be accepted as an ε-approximate solution for minimising the function $F(\mathbf{x})$, we must establish the inequality

$$F(\mathbf{x}^H) \leq (1+\varepsilon)\, F(\mathbf{x}^*). \tag{10.1}$$

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

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

This leads to two cases.

Case 1: For $G(\mathbf{x}^*) \geq 0$, we have $F(\mathbf{x}^H) \leq F(\mathbf{x}^*) + \varepsilon G(\mathbf{x}^*) = (1+\varepsilon)\, F(\mathbf{x}^*) - \varepsilon K$. If $K \geq 0$, the inequality (10.1) holds; however, if $K < 0$, there is no evidence that (10.1) will hold, and further analysis must be performed. We face the latter situation when studying the problem with a constant time MP in Section 10.3.

Case 2: For $G(\mathbf{x}^*) < 0$, we have $F(\mathbf{x}^H) \leq F(\mathbf{x}^*) - \varepsilon G(\mathbf{x}^*) = (1-\varepsilon)\, F(\mathbf{x}^*) + \varepsilon K$. Here $K > 0$. There is no evidence that (10.1) will hold, and further analysis must be performed. We face this situation when studying the problem with a start-time

dependent MP in Section 10.4.

The remainder of this chapter is organised as follows. Section 10.2 formally describes the problems under consideration; we also provide some preliminary calculations which enable us to show that the problem with a constant MP is NP-hard. In Section 10.3, we show how an FPTAS by Kellerer et al. (2003) developed for the Subset-sum problem can be adapted to the scheduling problem with a constant MP. In Section 10.4, we show how an FPTAS by Erel and Ghosh (2008) developed for the Half-product problem can be adapted to the scheduling problem with a MP of a variable duration.

10.2 Preliminaries

In this section, we give formal statements of the problems under consideration and establish their computational complexity. The jobs of set $N = \{1, 2, \ldots, n\}$ have to be processed on a single machine, which is subject to a deterioration effect. The deterioration model we consider in this chapter is related to a cumulative effect as described in Section 3.2.3. Recall that all the problems reviewed in Section 3.2.3 were solvable by an SPT rule. In fact, as a rule, for the problems with cumulative deterioration and no machine maintenance, polynomial-time algorithms are derived.

Throughout this chapter, for a non-empty subset N' define $p(N') := \sum_{j \in N'} p_j$; additionally define $p(\varnothing) := 0$. In a similar sense, we write $e(N')$, $q(N')$, etc.

In this chapter, we focus on the models with a specific cumulative deterioration effect which is a version of (3.17). Assume that the jobs are processed on a single machine in accordance with some permutation $\pi = (\pi(1), \ldots, \pi(n))$. The actual processing time of a job $j = \pi(r)$ that is sequenced in position r, $1 \leq r \leq n$, is given by

$$p_j(r) = p_j \left(A_j + B \sum_{k=1}^{r-1} p_{\pi(k)} \right), \tag{10.2}$$

where A_j, $j \in N$, and B are positive constants. Comparing the above model with (3.17), we assume that $Z = 1$, while on the other hand, we extend the model (3.17) by introducing the additional coefficients A_j and B. These coefficients allow us to handle other variations of the problem, without the need to alter our methodology in any way. In a special case of our model, with $A_j = 1$, $j \in N$, and $B = p(N)^{-1}$, the effect (10.2) becomes equivalent to the model introduced by Koulamas and Kyparisis (2007).

Additionally, we allow a maintenance period to be run exactly once during the planning period and it is known that it will restore the machine conditions completely,

i.e., after the MP the machine is as good as new. Using the notation defined in Chapter 4, for all the problems considered in this thesis, we denote the problem of minimising the makespan under these conditions by $1\,|Cumu, MP|\,C_{\max}$.

Notice that since we only perform a single MP in the schedule, which completely restores the machine conditions, there is no question of having any group-dependent effects. As a result, unlike Chapters 5-7, we only need to consider two different versions of problem $1\,|Cumu, MP|\,C_{\max}$, namely:

(i) Constant maintenance: the duration of the MP is β time units, where $\beta > 0$.

(ii) Start-time dependent maintenance: the duration of the MP is $\alpha\tau + \beta$ time units, provided that the MP starts at time τ; here $\alpha > 0$ and $\beta \geq 0$.

The former problem is denoted by $1\,|Cumu, MP\,[0]|\,C_{\max}$, while the latter is denoted by $1\,|Cumu, MP\,[\alpha]|\,C_{\max}$.

An instance of problem $1\,|Cumu, MP\,[\alpha]|\,C_{\max}$ is defined by the sequences p_j and A_j, $j \in N$, and numbers B and β, which are arbitrary positive integers. However, for $\alpha > 0$, we assume that α is bounded from above by a constant. This assumption is well justified by the fact that the duration of an MP is at least α times longer than the preceding period during which the machine was used. Without the made assumption, maintaining the machine would take considerably longer than the total processing time before the maintenance, which is hardly realistic.

In a schedule with a single MP the jobs are split into two groups: group 1 consists of the jobs scheduled before the maintenance and group 2 contains all other jobs. To solve problem $1\,|Cumu, MP\,[\alpha]|\,C_{\max}$, consider a schedule S with a permutation of jobs (π, σ). Assume that each group contains a total of $n^{[x]}$, $x \in \{1, 2\}$ jobs which belong to a set N_x, so that the permutations $\pi = \left(\pi(1), \ldots, \pi(n^{[1]})\right)$ and $\sigma = \left(\sigma(1), \ldots, \sigma(n^{[2]})\right)$, where $n^{[1]} + n^{[2]} = n$. In accordance with (10.2), the makespan of schedule S is given by

$$\begin{aligned} C_{\max}(S) \;=\; & p_{\pi(1)}A_{\pi(1)} + \sum_{r=2}^{n_1} p_{\pi(r)} \left(A_{\pi(r)} + B \sum_{k=1}^{r-1} p_{\pi(k)} \right) \\ & + \alpha \left(p_{\pi(1)}A_{\pi(1)} + \sum_{r=2}^{n_1} p_{\pi(r)} \left(A_{\pi(r)} + B \sum_{k=1}^{r-1} p_{\pi(k)} \right) \right) + \beta \\ & + p_{\sigma(1)}A_{\sigma(1)} + \sum_{r=2}^{n_2} p_{\sigma(r)} \left(A_{\sigma(r)} + B \sum_{k=1}^{r-1} p_{\sigma(k)} \right). \end{aligned}$$

The total processing time of the jobs in the first group can be computed as

$$\begin{aligned} p_{\pi(1)}A_{\pi(1)} + \sum_{r=2}^{n_1} p_{\pi(r)}\left(A_{\pi(r)} + B\sum_{k=1}^{r-1} p_{\pi(k)}\right) &= \sum_{r=1}^{n_1} p_{\pi(r)}A_{\pi(r)} + B\sum_{1\le k<r\le n_1} p_{\pi(k)}p_{\pi(r)} \\ &= \sum_{r=1}^{n_1} p_{\pi(r)}A_{\pi(r)} + \frac{B}{2}\left(p(N_1)^2 - \sum_{r=1}^{n_1} p_{\pi(r)}^2\right). \end{aligned}$$

Similarly, the total processing time of the jobs in the second group can be computed as

$$p_{\sigma(1)}A_{\sigma(1)} + \sum_{r=2}^{n_2} p_{\sigma(r)}\left(A_{\sigma(r)} + B\sum_{k=1}^{r-1} p_{\sigma(k)}\right) = \sum_{r=1}^{n_2} p_{\sigma(r)}A_{\sigma(r)} + \frac{B}{2}\left(p(N_2)^2 - \sum_{r=1}^{n_2} p_{\sigma(r)}^2\right).$$

Define

$$q_j = p_j A_j,\ j = 1, 2, \ldots, n,$$

so that

$$q(N_1) := \sum_{r=1}^{n_1} p_{\pi(r)}A_{\pi(r)},\ q(N_2) := \sum_{r=1}^{n_2} p_{\sigma(r)}A_{\sigma(r)},\ q(N) := q(N_1) + q(N_2).$$

Thus, the makespan can be written as

$$\begin{aligned} C_{\max}(S) &= q(N_1) + \frac{B}{2}\left(p(N_1)^2 - \sum_{r=1}^{n_1} p_{\pi(r)}^2\right) + \alpha\left(q(N_1) + \frac{B}{2}\left(p(N_1)^2 - \sum_{r=1}^{n_1} p_{\pi(r)}^2\right)\right) \\ &\quad + q(N_2) + \frac{B}{2}\left(p(N_2)^2 - \sum_{r=1}^{n_2} p_{\sigma(r)}^2\right) + \beta, \end{aligned}$$

which implies

$$\begin{aligned} C_{\max}(S) &= q(N) + \frac{B}{2}\left(p(N_1)^2 + p(N_2)^2 - \sum_{j\in N} p_j^2\right) + \\ &\quad \alpha\left(q(N_1) + \frac{B}{2}\left(p(N_1)^2 - \sum_{j\in N_1} p_j^2\right)\right) + \beta \end{aligned} \tag{10.3}$$

for problem $1\,|Cumu, MP\,[\alpha]|\,C_{\max}$ and

$$C_{\max}(S) = q(N) + \frac{B}{2}\left(p(N_1)^2 + p(N_2)^2\right) - \frac{B}{2}\sum_{j\in N} p_j^2 + \beta \tag{10.4}$$

for problem $1\,|Cumu, MP\,[0]|\,C_{\max}$.

Notice that (10.3) and (10.4) demonstrate that for problems $1\,|Cumu, MP\,[\alpha]|\,C_{\max}$ and $1\,|Cumu, MP\,[0]|\,C_{\max}$ respectively, the order of jobs in each group does not affect the makespan. This complies with Gordon et al. (2008), where the makespan has been shown to be sequence independent for the single machine problem with the deterioration effect (3.17), with $Z = 1$, and no maintenance period. Thus, the main issue in solving problem $1\,|Cumu, MP\,[\alpha]|\,C_{\max}$, including its simpler version $1\,|Cumu, MP\,[0]|\,C_{\max}$, is to find an appropriate partition of the jobs into two groups.

Proposition 10.1. *Problem* $1\,|Cumu, MP\,[0]|\,C_{\max}$ *is* NP*-hard in the ordinary sense.*

The correctness of the above statement can be verified as follows. Let $x_j = 1$ if job j is assigned to set N_1; otherwise, define $x_j = 0$. It follows from (10.4), that problem $1\,|Cumu, MP\,[0]|\,C_{\max}$, reduces to minimising $p(N_1)^2 + p(N_2)^2 = \left(\sum_{j\in N} p_j x_j\right)^2 + \left(\sum_{j\in N} p_j(1-x_j)\right)^2 = p(N) - 2\left(\sum_{j\in N} p_j x_j\right)\left(\sum_{j\in N} p_j(1-x_j)\right)$, i.e., to maximising $\left(\sum_{j\in N} p_j x_j\right)\left(\sum_{j\in N} p_j(1-x_j)\right)$. Jurisch, Kubiak and Józefowska (1997) show that the problem $P2\,||\,C_{\max}$ of minimising the makespan on two parallel identical machines with no preemption allowed, reduces to maximising the product $\left(\sum_{j\in N} p_j x_j\right)\left(\sum_{j\in N} p_j(1-x_j)\right)$ for $x_j \in \{0,1\},\ j \in N$. From this we immediately derive that problems $1\,|Cumu, MP\,[0]|\,C_{\max}$ and $P2\,||\,C_{\max}$ are essentially equivalent and it is well known that problem $P2\,||\,C_{\max}$ is NP-hard in the ordinary sense; see Garey and Johnson (1978, 1979).

It is clear that problem $1\,|Cumu, MP\,[\alpha]|\,C_{\max}$ is no easier than problem $1\,|Cumu, MP\,[0]|\,C_{\max}$. Thus, the best possible approximation result that can be derived for either problem is an FPTAS. In the subsequent sections we develop such approximation schemes.

10.3 FPTAS by Subset-Sum

In this section, we consider problem $1\,|Cumu, MP\,[0]|\,C_{\max}$. In Proposition 10.1 we state that this problem is NP-hard and show that it is essentially equivalent to problem $P2\,||\,C_{\max}$. The latter problem can be formulated as a Subset-sum problem of the form

(2.4), which is rewritten for problem $1\,|Cumu, MP\,[0]|\,C_{\max}$ as

$$\begin{array}{ll} \max & \sum_{j\in N} p_j x_j \\ \text{subject to} & \sum_{j\in N} p_j x_j \le \Delta \\ & x_j \in \{0,1\},\ j \in N, \end{array} \tag{10.5}$$

where $\Delta := p(N)/2$. The following statements hold.

Proposition 10.2. *Suppose that $x_j^* \in \{0,1\}$, $j \in N$, are the optimal values of the decision variables for the problem (10.5). Define $N_1^* := \left\{j \in N | x_j^* = 1\right\}$ and $N_2^* = N\backslash N_1^*$. Then for problem $1\,|Cumu, MP\,[0]|\,C_{\max}$ there exists an optimal schedule S^* in which the jobs of set N_1^* are scheduled in one group and the jobs of set N_2^* are scheduled in the other group.*

Corollary 10.1. *For a schedule S^* that is optimal for problem $1\,|Cumu, MP\,[0]|\,C_{\max}$ the following lower bound*

$$C_{\max}(S^*) \ge q(N) + B\Delta^2 - \frac{B}{2}\sum_{j\in N} p_j^2 + \beta \tag{10.6}$$

holds.

To see this, observe that for any partition of set N into subsets N_1 and N_2 the inequality $p(N_1)^2 + p(N_2)^2 \ge 2\Delta^2$ holds. Indeed, if for some non-negative δ we have that $p(N_1) = \Delta - \delta$ and $p(N_2) = \Delta + \delta$, then $p(N_1)^2 + p(N_2)^2 = 2\Delta^2 + 2\delta^2 \ge 2\Delta^2$.

Our further consideration is based on the following statement; see Kellerer et al. (2003) and Lemma 4.6.1 in Kellerer, Pferschy and Pisinger (2004).

Theorem 10.1. *A Subset-sum problem of the form (10.5) admits an FPTAS that for a given positive ε, either finds an optimal solution $x_j^* \in \{0,1\}$, $j \in N$, such that*

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

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

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

Such an FPTAS requires no more than $O\left(\min\left\{n/\varepsilon, n + \frac{1}{\varepsilon^2}\log\left(\frac{1}{\varepsilon}\right)\right\}\right)$ time.

The algorithm below assigns the jobs to groups in accordance with the above mentioned FPTAS, applied to problem (10.5) with $\Delta = p(N)/2$.

Algorithm Eps1

INPUT: An instance of problem $1\,|Cumu, MP\,[0]|\,C_{\max}$ and an $\varepsilon > 0$

OUTPUT: A schedule S^{ε} such that $C_{\max}(S^{\varepsilon}) \leq (1+\varepsilon)\,C_{\max}(S^{*})$

Step 1. For a given $\varepsilon > 0$ define $\varepsilon_0 := \frac{\varepsilon}{\varepsilon+1}$.

Step 2. With the defined ε_0, run an FPTAS for problem (10.5) to find the values $x_j^{\varepsilon} \in \{0,1\}$, $j \in N$. Define $N_1^{\varepsilon} := \left\{j \in N | x_j^{\varepsilon} = 1\right\}$ and $N_2^{\varepsilon} := N \backslash N_1^{\varepsilon}$.

Step 3. Output schedule S^{ε} for the original problem $1\,|Cumu, MP\,[0]|\,C_{\max}$, in which the jobs of set N_1^{ε} are assigned to one group and sequenced before the maintenance and the jobs of set N_2^{ε} are assigned to the other group to be scheduled after the maintenance. Stop.

Recall that the makespan as given in (10.4), consists of a variable part and a constant. Due to Proposition 10.2 and Theorem 10.1, the variable part can be minimised by means of an FPTAS. However, as discussed in Section 1, a direct application of that FPTAS does not necessarily result into an FPTAS for the original problem, since (10.4) contains a constant $q(N) + \beta - \frac{B}{2}\sum_{j\in N} p_j^2$, which can be negative. Below we prove that Algorithm Eps1 gives an appropriate treatment to the negative constant, and therefore allows us to adapt the existing FPTAS to deliver an ε-approximate solution for minimising the overall original objective function.

Theorem 10.2. *Algorithm Eps1 is an FPTAS for problem* $1\,|Cumu, MP\,[0]|\,C_{\max}$ *that runs in* $O\left(\min\left\{n/\varepsilon, n + \left(1+\frac{1}{\varepsilon}\right)^2 \log\left(1+\frac{1}{\varepsilon}\right)\right\}\right)$ *time.*

Proof: Using an FPTAS by Kellerer et al. (2003) from Theorem 10.1 with $\varepsilon_0 := \frac{\varepsilon}{\varepsilon+1}$, we observe that $O(n/\varepsilon_0) = O\left(n\frac{\varepsilon+1}{\varepsilon}\right) = O(n/\varepsilon)$ and $\frac{1}{\varepsilon_0^2}\log\left(\frac{1}{\varepsilon_0}\right) = \left(1+\frac{1}{\varepsilon}\right)^2 \log\left(1+\frac{1}{\varepsilon}\right)$, so that the required running time is achieved. To complete the proof, we need to prove that $C_{\max}(S^{\varepsilon}) \leq (1+\varepsilon)\,C_{\max}(S^{*})$.

Due to Theorem 10.1, we only need to consider the case that the FPTAS in Step 2 does not find an optimal solution to problem (10.5); otherwise schedule S^{ε} is optimal. Below we only look at the instances of problem $1\,|Cumu, MP\,[0]|\,C_{\max}$ for which $p_j \leq \Delta$, $j \in N$, since otherwise an optimal solution can be obtained by scheduling the largest job in one group and the remaining jobs in the other.

We assume that there exists a δ, $\delta \leq \varepsilon_0$, such that $(1-\varepsilon_0)\Delta \leq p(N_1^\varepsilon) = \Delta(1-\delta) < \Delta$ and $p(N_2^\varepsilon) = (1+\delta)\Delta$. Applying (10.4) and (10.6), we have that

$$\begin{aligned} C_{\max}(S^\varepsilon) &= q(N) + \frac{B}{2}\left(p(N_1^\varepsilon)^2 + p(N_2^\varepsilon)^2\right) + \beta - \frac{B}{2}\sum_{j\in N} p_j^2 \\ &= q(N) + B\left(\Delta^2 + \delta^2\Delta^2\right) + \beta - \frac{B}{2}\sum_{j\in N} p_j^2 \leq C_{\max}(S^*) + B\delta^2\Delta^2. \end{aligned}$$

Below we demonstrate that $B\delta(1-\delta)\Delta^2$ is a lower bound on the optimal makespan $C_{\max}(S^*)$. Consider the problem

$$\begin{aligned} \max \quad & \sum_{j\in N} p_j^2 \\ \text{subject to} \quad & \sum_{j\in N_1^\varepsilon} p_j = (1-\delta)\Delta \\ & \sum_{j\in N_2^\varepsilon} p_j = (1+\delta)\Delta \\ & 0 \leq p_j \leq \Delta,\ j \in N. \end{aligned} \tag{10.7}$$

This problem is related to one of the basic problems of submodular optimisation, a so-called resource allocation problem with a convex separable objective function; see Hochbaum and Hong (1995) and Katoh and Ibaraki (1998). The problem is known to be solvable by the greedy algorithm, which in the case under consideration, scans the values p_j in any order and gives each of them the largest possible value. In our case, the greedy algorithm will find an optimal solution to (10.7) in which one of the p_j's is equal to Δ, one to $(1-\delta)\Delta$ and one to $\delta\Delta$, while all others are equal to zero. Thus,

$$\sum_{j\in N} p_j^2 \leq (1-\delta)^2\Delta^2 + \Delta^2 + (\delta\Delta)^2 = 2\Delta^2(1+\delta^2-\delta)$$

provides an upper bound on the sum of squares of the processing times for all instances of the problem for which Step 2 of Algorithm Eps1 delivers $p(N_1^\varepsilon) = \Delta(1-\delta)$ and $p(N_2^\varepsilon) = (1+\delta)\Delta$, including the instance under consideration.

Substituting this into (10.6) we derive a lower bound

$$C_{\max}(S^*) \geq q(N) + B\left(\delta - \delta^2\right)\Delta^2 + \beta \geq B\delta(1-\delta)\Delta^2.$$

This lower bound implies that

$$C_{\max}(S^{\varepsilon}) \leq C_{\max}(S^*) + B\delta^2\Delta^2 \leq \left(1 + \frac{\delta}{1-\delta}\right) C_{\max}(S^*).$$

Since $\frac{\delta}{1-\delta}$ increases, we have that

$$C_{\max}(S^{\varepsilon}) \leq \left(1 + \frac{\varepsilon_0}{1-\varepsilon_0}\right) C_{\max}(S^*).$$

Thus, to obtain an FPTAS for our problem with the accuracy ε, we need to use the FPTAS for problem (10.5) with $\varepsilon_0 = \frac{\varepsilon}{\varepsilon+1}$. □

10.4 FPTAS by Half-Product

In this section, we show that problem $1\,|Cumu, MP\,[\alpha]|\,C_{\max}$ can be formulated in terms of quadratic Boolean programming. We discuss an opportunity that this reformulation offers regarding the design of an FPTAS for the problem under consideration.

Given problem $1\,|Cumu, MP\,[\alpha]|\,C_{\max}$, introduce a Boolean variable x_j in such a way that

$$x_j = \begin{cases} 1, \text{ if job } j \text{ is scheduled in the first group} \\ 0, \text{ otherwise} \end{cases}$$

for each job j, $1 \leq j \leq n$. Taking the jobs in any order, i.e., in the order of their numbering, if job j is scheduled in the first group then it completes at time

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

so that the MP starts at time $\sum_{j=1}^{n} p_j x_j \left(A_j + B\sum_{i=1}^{j-1} p_i x_i\right)$. If job j is scheduled in the second group, then

$$\begin{aligned} C_j \;=\; & \sum_{j=1}^{n} p_j x_j \left(A_j + B\sum_{i=1}^{j-1} p_i x_i\right) + \alpha\left(\sum_{j=1}^{n} p_j x_j \left(A_j + B\sum_{i=1}^{j-1} p_i x_i\right)\right) + \beta \\ & + \sum_{j=1}^{n} p_j(1-x_j)\left(A_j + B\sum_{i=1}^{j-1} p_i(1-x_i)\right). \end{aligned}$$

This implies that in order to solve problem $1\,|Cumu, MP\,[\alpha]|\,C_{\max}$, we need to

minimise the function

$$\begin{aligned} F_\alpha(\mathbf{x}) &= (\alpha+1)\left(\sum_{j=1}^{n} p_j x_j \left(A_j + B\sum_{i=1}^{j-1} p_i x_i\right)\right) \\ &\quad + \sum_{j=1}^{n} p_j(1-x_j)\left(A_j + B\sum_{i=1}^{j-1} p_i(1-x_i)\right) + \beta, \end{aligned}$$

which can be rewritten as

$$\begin{aligned} F_\alpha(\mathbf{x}) &= (\alpha+1)\left(B\sum_{1\le i<j\le n} p_i p_j x_i x_j + \sum_{j=1}^{n} p_j A_j x_j\right) \\ &\quad + B\sum_{1\le i<j\le n} p_i p_j (1-x_i)(1-x_j) + \sum_{j=1}^{n} p_j A_j (1-x_j) + \beta. \end{aligned} \tag{10.8}$$

Function (10.8) is similar to the objective function for the Symmetric Quadratic Knapsack problem studied by Kellerer and Strusevich (2010a, 2010b). The lemma below links the function $F_\alpha(\mathbf{x})$ to the Half-product problem of the form (2.5).

Lemma 10.1. *Function $F_\alpha(\mathbf{x})$ can be represented as $F_\alpha(\mathbf{x}) = H(\mathbf{x}) + K$, where $H(\mathbf{x})$ is the half-product function of the form (2.5), with $a_i := (\alpha+2)Bp_i, b_j := p_j$ and $h_j := B\left(p_j p(N) - p_j^2\right) + \alpha p_j A_j$, $j \in N$, and the constant K is defined as*

$$K := \beta + q(N) + B\sum_{1\le i<j\le n} p_i p_j,$$

where $q(N) = \sum_{j=1}^{n} p_j A_j$.

Proof: It follows that

$$\begin{aligned} \sum_{1\le i<j\le n} p_i p_j (1-x_i)(1-x_j) &= \sum_{1\le i<j\le n} p_i p_j x_i x_j + \sum_{1\le i<j\le n} p_i p_j \\ -\sum_{j=1}^{n} p_j x_j \left(\sum_{i=1}^{j-1} p_i\right) &- \sum_{j=1}^{n} p_j \left(\sum_{i=1}^{j-1} p_i x_i\right). \end{aligned}$$

Notice that

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

so that

$$\sum_{1\leq i<j\leq n} p_i p_j (1-x_i)(1-x_j) = \sum_{1\leq i<j\leq n} p_i p_j x_i x_j + \sum_{1\leq i<j\leq n} p_i p_j - \sum_{j=1}^{n} \left(p_j p(N) - p_j^2\right) x_j.$$

Thus, (10.8) becomes

$$\begin{aligned} F_\alpha(\mathbf{x}) &= (\alpha+2)\, B \sum_{1\leq i<j\leq n} p_i p_j x_i x_j - \sum_{j=1}^{n} \left(B p_j p(N) - B p_j^2 + \alpha p_j A_j\right) x_j \\ &\quad + \left(\beta + \sum_{j=1}^{n} p_j A_j + B \sum_{1\leq i<j\leq n} p_i p_j\right), \end{aligned}$$

which proves the lemma. □

Consider the problem of minimising the function $F(\mathbf{x}) = H(\mathbf{x}) + K$, where $H(\mathbf{x})$ is a Half-product function of the form (2.5), and K is a constant. It is known that an FPTAS for minimising the function $H(\mathbf{x})$ does not necessarily behave as an FPTAS for minimising the function $F(\mathbf{x})$. This is due to the fact the optimal value of $H(\mathbf{x})$ is negative; see Erel and Ghosh (2008) and Kellerer and Strusevich (2012) for discussion and examples. Suppose that a lower bound LB and an upper bound UB on the optimal value of the function $F(\mathbf{x})$ are available, i.e., $LB \leq F(\mathbf{x}^*) \leq UB$. Erel and Ghosh (2008) adopt their FPTAS for minimising the function $H(\mathbf{x})$ to minimising the function $F(\mathbf{x})$. They develop an algorithm that delivers a solution $\mathbf{x}^0$ such that $F(\mathbf{x}^0) - LB \leq \varepsilon LB$ in $O(\gamma n^2/\varepsilon)$ time, where $\gamma \geq UB/LB$. We refer to this version of the scheme as γ-FPTAS.

The makespan $C_{\max}(S)$ associated with a partition of the jobs $N = N_1 \cup N_2$ into two groups will be denoted by $F_\alpha(N_1, N_2)$ and defined by (10.3); for $\alpha = 0$ the makespan will be denoted by $F_0(N_1, N_2)$ and defined by (10.4).

Below we describe how to adapt the γ-FPTAS for solving problem $1\,|Cumu, MP\,[\alpha]|\,C_{\max}$.

Algorithm Eps2

INPUT: An instance of problem $1\,|Cumu, MP\,[\alpha]|\,C_{\max}$ with α bounded by a constant and an $\varepsilon > 0$

OUTPUT: A schedule S^ε such that $C_{\max}(S^\varepsilon) \leq (1+\varepsilon)\, C_{\max}(S^*)$

Step 1. Given an instance for problem $1\,|Cumu, MP\,[\alpha]|\,C_{\max}$, take an arbitrary pos-

itive ε' and run Algorithm Eps1 with $\varepsilon = \varepsilon'$, applied to the counterpart of the original problem with constant maintenance ($\alpha = 0$). Let $N_1^{\varepsilon'}$ and $N_2^{\varepsilon'}$ be the groups found by Algorithm Eps1. Compute $F_0(N_1^{\varepsilon'}, N_2^{\varepsilon'})$ by (10.4) with $N_1 = N_1^{\varepsilon'}$ and $N_2 = N_2^{\varepsilon'}$.

Step 2. Define $UB := \left(1 + \frac{\alpha}{2}\right) F_0(N_1^{\varepsilon'}, N_2^{\varepsilon'})$, $\gamma := \left(1 + \frac{\alpha}{2}\right)(1 + \varepsilon')$. Take a small positive ε and run the γ-FPTAS by Erel and Ghosh (2008). With the found values $x_j^{\varepsilon} \in \{0, 1\}$, $j \in N$, define $N_1^{\varepsilon} := \left\{j \in N | x_j^{\varepsilon} = 1\right\}$ and $N_2^{\varepsilon} = N \backslash N_1^{\varepsilon}$. If

$$q(N_1^{\varepsilon}) + \frac{B}{2} p(N_1^{\varepsilon})^2 - \frac{B}{2} \sum_{j \in N_1^{\varepsilon}} p_j^2 > q(N_2^{\varepsilon}) + \frac{B}{2} p(N_2^{\varepsilon})^2 - \frac{B}{2} \sum_{j \in N_2^{\varepsilon}} p_j^2,$$

swap N_1^{ε} and N_2^{ε}.

Step 3. Output schedule S^{ε} for the original problem $1 \left| Cumu, MP\left[\alpha\right] \right| C_{\max}$, in which the jobs of set N_1^{ε} are assigned to one group and sequenced before the maintenance and the jobs of set N_2^{ε} are assigned to the other group to be scheduled after the maintenance. Stop.

Theorem 10.3. *Algorithm Eps2 is an FPTAS for problem* $1 \left| Cumu, MP\left[\alpha\right] \right| C_{\max}$ *that runs in* $O(n^2/\varepsilon)$ *time.*

Proof: It follows from (10.3) and (10.4) that

$$F_{\alpha}(N_1, N_2) = F_0(N_1, N_2) + \alpha \left(q(N_1) + \frac{B}{2} p(N_1)^2 - \frac{B}{2} \sum_{j \in N_1} p_j^2 \right).$$

Besides, for the purpose of finding the best schedule for problem $1 \left| Cumu, MP\left[\alpha\right] \right| C_{\max}$ defined by a partition $N = N_1 \cup N_2$ we may assume that

$$q(N_1) + \frac{B}{2} p(N_1)^2 - \frac{B}{2} \sum_{j \in N_1} p_j^2 \leq q(N_2) + \frac{B}{2} p(N_2)^2 - \frac{B}{2} \sum_{j \in N_2} p_j^2,$$

otherwise, we will swap the groups scheduled before and after the maintenance. This implies that

$$q(N_1) + \frac{B}{2} p(N_1)^2 - \frac{B}{2} \sum_{j \in N_1} p_j^2 \leq \frac{1}{2} F_0(N_1, N_2),$$

and therefore

$$F_{\alpha}(N_1, N_2) \leq \left(1 + \frac{\alpha}{2}\right) F_0(N_1, N_2). \tag{10.9}$$

Let S_{α}^* denote a schedule that is optimal for problem $1 \left| Cumu, MP\left[\alpha\right] \right| C_{\max}$. That schedule is defined by a partition of the set N of jobs into two subsets, which we denote

by $N_1^*(\alpha)$ and $N_1^*(\alpha)$. In particular, $N_1^*(0)$ and $N_1^*(0)$ define an optimal schedule for problem $1\,|Cumu, MP\,[\alpha]|\,C_{\max}$. Let also $N_1^{\varepsilon'}$ and $N_2^{\varepsilon'}$ be the sets that are found in Step 1 of Algorithm Eps2. Due to (10.9) we have

$$F_\alpha(N_1^*(\alpha), N_2^*(\alpha)) \le F_\alpha(N_1^{\varepsilon'}, N_2^{\varepsilon'}) \le \left(1 + \frac{\alpha}{2}\right) F_0(N_1^{\varepsilon'}, N_2^{\varepsilon'}).$$

On the other hand,

$$F_\alpha(N_1^*(\alpha), N_2^*(\alpha)) \ge F_0(N_1^*(0), N_2^*(0)) \ge \frac{F_0(N_1^{\varepsilon'}, N_2^{\varepsilon'})}{(1+\varepsilon')}.$$

Thus, for the optimal makespan in problem $1\,|Cumu, MP\,[\alpha]|\,C_{\max}$, we deduce that $\frac{F_0(N_1^{\varepsilon'}, N_2^{\varepsilon'})}{(1+\varepsilon')}$ is a lower bound, while $\left(1 + \frac{\alpha}{2}\right) F_0(N_1^{\varepsilon'}, N_2^{\varepsilon'})$ is an upper bound, and therefore the values of UB and γ in Step 2 are correct. The overall running time of Algorithm Eps2 is determined by the time complexity of Step 2. According to Erel and Ghosh (2008), the γ-FPTAS requires $O(\gamma n^2/\varepsilon)$, which in our case becomes $O(n^2/\varepsilon)$, since γ only depends on a given α bounded by a constant and on a chosen constant ε'. Algorithm Eps2 will deliver a solution of the required accuracy, i.e., $C_{\max}(S^\varepsilon)/C_{\max}(S^*) \le 1 + \varepsilon$. □

10.5 Conclusion

In this chapter, we solve two problems with cumulative deterioration effects and a single maintenance activity. This is the first study, in which a cumulative effect is combined with a rate-modifying activity. Mathematically, the considered problems are linked to linear and quadratic problems of Boolean programming that admit an FPTAS. Our main technical task has been to adapt the known FPTASs to our problems, which is not straightforward due to the opposite signs of the variable and constant parts of the objective function.

The next step in studying the models with cumulative deterioration could be a search for approximation algorithms or schemes that would allow us to handle multiple MPs. Also, the problem of minimising the total flow time under these conditions remains an open problem.

Part IV

Parallel Machine Scheduling

CHAPTER 11

Impact of Adding Extra Machines

In this chapter, we consider the classical scheduling problems of processing jobs on identical parallel machines to minimise (i) the makespan or (ii) the total flow time. The processing times of the jobs are assumed to be fixed, so that unlike the previous chapters in this thesis, they do not change with respect to their location in the schedule. The focus of this chapter, is to perform an analytical study on the impact that additional machines may have, if added to the system. We measure such a machine impact by the ratio of the value of the objective function computed with the original number of machines to the one computed with extra machines. We give tight bounds on the machine impact for the problem of minimising the makespan, for both the preemptive and non-preemptive versions, as well as for the problem of minimising the total flow time. We also present polynomial-time exact and approximation algorithms to make a cost-effective choice of the number of machines, provided that each machine incurs a cost and the objective function captures the trade-off between the cost of the used machines and a scheduling objective.

This study formally does not belong to scheduling with changing time, but it shares the same ideological point of combining scheduling and logistics decisions with a purpose of improving the overall performance of the processing system. Besides, the use of the convex sequences has appeared to be useful for that study, especially for the model with a total flow time objective.

The results of this chapter are published in our paper Rustogi and Strusevich (2013a). The presentation of content in the paper is very similar to the content provided in this chapter.

11.1 Brief Overview of Problem

In the problems under consideration, we are given a set of jobs, each of which can be processed by any of the available machines. The machines are identical, i.e., the processing time of a job does not depend on the machine assignment decisions. If no preemption is allowed, each job is assigned to exactly one machine and is processed on that machine without interruption. In a preemptive schedule, the processing of a job on a machine can be interrupted at any time and then resumed later on any other machine, provided that a job is not processed on two or more machines at a time and the total duration of its processing is equal to the given processing time.

The main aspect of this study is to investigate the influence that additional machines may have on the objective function. We measure this influence by a *machine impact*, which is defined as a ratio of the objective function value computed without using extra machines over the function value computed with additional machines.

In this chapter, we give tight bounds on the machine impact for the problem of minimising the makespan, for both the preemptive and non-preemptive versions, as well as for the problem of minimising the total flow time. For the latter problem only the non-preemptive version is considered, since, as shown by McNaughton (1959), for this objective function there is no advantage in allowing preemption.

We believe that the machine impact is an important characteristic of the processing system. In manufacturing, the decisions on adding machine-tools to the existing park of similar equipment are often considered. In computing, parallel processors can be added to a computer system to boost its performance. We present several meaningful interpretations of the results of this chapter in Section 11.6.

Computing a machine impact is closely related to scheduling with resource augmentation. This direction of research has been initiated by Kalyanasundaram and Pruhs (2000) who have demonstrated that good competitive ratios of online algorithms can be achieved, provided that extra resources are used by the scheduler, compared to the original settings. Resource augmentation allows the use of machines with faster speeds, as in Kalyanasundaram and Pruhs (2000), or additional parallel machines, as in Azar, Epstein and van Stee (2000) and Brehob, Torng and Uthaisombut (2000), or both, as in Chekuri et al. (2004). In particular, the results presented by Azar, Epstein and van Stee (2000) and Brehob, Torng and Uthaisombut (2000) can be interpreted in terms of computing a machine impact with respect to the makespan of a non-preemptive schedule; see Section 11.2 for details.

Clearly, additional machines may improve the scheduling performance, but in real-

ity, however, adding a machine cannot be seen free. Introducing more machines reduces a scheduling objective function but may be unacceptable due to a high cost of machine usage. In this chapter, we also study the problem of the cost-effective choice of the number of the machines. The total cost function captures the trade-off between the gain in reducing the value of a scheduling performance measure and a loss associated with increasing the number of machines. The study of online versions of the problems of the cost-optimal selection of the machines has been initiated by Imreh and Noga (1999), who study the non-preemptive problem with an objective that is the sum of the makespan and the cost of the machines used. In their model, the decision-maker has no machines in the beginning and when a job is revealed, he/she may buy as many machines as needed. The jobs are either released according to a list (The List Model) or arrive over time (The Time Model). The focus is on deriving lower and upper bounds on the competitive ratios of the online algorithms. The best known bounds for the online non-preemptive problem are due to Dósa and Tan (2010). The preemptive version of the List Model is studied by Jiang and He (2005). For the List Model, the semi-online scenarios are considered by He and Cai (2002) (no preemption allowed) and by Jiang and He (2006) (both preemptive and non-preemptive versions). In the semi-online scenarios, the decision-maker is either aware of the longest processing time of arriving jobs or of the total processing time, and this leads to better bounds on the competitive ratios. A more general cost function for using the machines is analysed by Imreh (2009).

The remainder of this chapter is organised as follows. In Section 11.2, we give tight estimates of the machine impact for the problem of minimising the makespan, for both the preemptive and non-preemptive versions. Section 11.3 handles the machine impact for the problem of minimising the total flow time. Here we start with the problem with unit processing times, derive both upper and lower bounds on the impact of adding only one extra machine, and then extend the obtained upper bound to the general case. The cost-effective choice of the number of machines for the makespan is addressed in Section 11.4. We present a linear time algorithm for the preemptive version and an approximation algorithm accompanied by its worst-case analysis and computational experiments for the non-preemptive version. A similar problem with the total flow time as a scheduling measure is studied in Section 11.5. We establish a form of discrete convexity of the total cost function and this results into a fast exact algorithm. Section 11.6 gives examples of practical implications of the obtained results.

11.2 Estimating Machine Impact: Makespan

Formally, in all problems in this chapter, we are given the jobs of set $N = \{1, 2, \ldots, n\}$ and m identical parallel machines M_1, $M_2, \ldots, M_m$, where $m \geq 2$ and $n \geq m$. No matter to which machine a job $j \in N$ is assigned, its processing time is equal to p_j. We denote the sum of all processing times by $P = \sum_{j \in N} p_j/m$ and the largest processing time by $p_{\max}$.

Definition 11.1. *Let $S^*(m)$ denote a schedule that minimises an objective function F on m identical parallel machines. For $\widehat{m} > m$, we define the machine impact as the ratio*

$$I(m, \widehat{m}) = \frac{F\left(S^*(m)\right)}{F\left(S^*(\widehat{m})\right)}. \tag{11.1}$$

Clearly, $I(m, \widehat{m}) \geq 1$ for any regular objective function F. Using standard scheduling notation, the scheduling problem of finding a non-preemptive schedule $S^*_{np}(m)$ that minimises an objective function F on m identical parallel machines is denoted by $Pm \,||\, F$. The problems of finding an optimal preemptive schedule $S^*_p(m)$ is denoted by $Pm \,|pmtn|\, F$.

In this section, we focus on minimising the makespan. For a schedule S, the makespan is denoted by $C_{\max}(S)$. Thus, $F = C_{\max}$, and the problems under consideration are $Pm \,||\, C_{\max}$ and $Pm \,|pmtn|\, C_{\max}$. Refer to Section 2.2.5 for individual results on these two problems. Also see a recent focused survey on parallel machine scheduling with makespan as the objective function, by Chen (2004).

Recall from Section 2.2.5 that problem $Pm \,||\, C_{\max}$ is NP-hard for each $m \geq 2$, while problem $Pm \,|pmtn|\, C_{\max}$ is polynomially solvable by a 'wrap-around' algorithm due to McNaughton (1959). It is clear that for any schedule $S\,(m)$ on m machines, preemptive or not, the bounds (2.9) and (2.10) hold for the makespan $C_{\max}\,(S(m))$. For the non-preemptive case, these bounds need not be tight.

If $C_{\max}\left(S^*(m)\right) = p_{\max} \geq \frac{P}{m} > \frac{P}{\widehat{m}}$, for $\widehat{m} > m$, then $C_{\max}\left(S^*(\widehat{m})\right) = p_{\max}$ and $I(m, \widehat{m}) = 1$, irrespective of whether preemption is allowed or not. Thus, in general there is no non-trivial lower bound on the machine impact for the problem of minimising the makespan and we focus our attention on finding an upper bound on the machine impact.

For problem $Pm \,|pmtn|\, C_{\max}$, McNaughton's algorithm (see Section 2.2.5) requires $O(n)$ time and finds an optimal schedule $S^*_p(m)$ with $C_{\max}\left(S^*_p(m)\right) = \max\left\{p_{\max}, \frac{P}{m}\right\}$.

For this case, it is very easy to compute the machine impact $I(m, \widehat{m})$, i.e., the ratio

$$I(m, \widehat{m}) = \frac{C_{\max}\left(S_p^*(m)\right)}{C_{\max}\left(S_p^*(\widehat{m})\right)}.$$

We know that $C_{\max}\left(S_p^*(m)\right) = p_{\max}$ implies $I(m, \widehat{m}) = 1$. On the other hand, if $C_{\max}\left(S_p^*(m)\right) = \frac{P}{m}$, then due to $C_{\max}\left(S_p^*(\widehat{m})\right) \geq \frac{P}{\widehat{m}}$, we obtain that for problem $Pm\,|pmtn|\,C_{\max}$, the machine impact

$$I(m, \widehat{m}) \leq \frac{\widehat{m}}{m}. \tag{11.2}$$

This bound is tight, which can be seen by considering an instance of the problem with $m\widehat{m}$ jobs of unit duration each. In this case, $C_{\max}\left(S_p^*(m)\right) = \widehat{m}$ and $C_{\max}\left(S_p^*(\widehat{m})\right) = m$.

Notice that the bound (11.2) will appear again in Section 11.3, as an upper bound on the machine impact for the problem of minimising the total flow time.

Now we pass to considering the machine impact for problem $Pm\,||\,C_{\max}$. Here the situation is different because the optimal values of makespan are non-available. Still, a tight upper bound on the machine impact

$$I(m, \widehat{m}) = \frac{C_{\max}\left(S_{np}^*(m)\right)}{C_{\max}\left(S_{np}^*(\widehat{m})\right)}$$

can be derived.

Theorem 11.1. *For problem $Pm\,||\,C_{\max}$ and $\widehat{m} = um + v$, where u and v are integers such that $u \geq 1$, $0 \leq v \leq m-1$, the following bound*

$$I(m, \widehat{m}) = \frac{C_{\max}\left(S_{np}^*(m)\right)}{C_{\max}\left(S_{np}^*(\widehat{m})\right)} \leq \left\lceil \frac{\widehat{m}}{m} \right\rceil = u + 1 \tag{11.3}$$

holds and this bound is tight.

Proof: Take a schedule $S_{np}^*(\widehat{m})$ that minimises the makespan on $\widehat{m}$ machines. We show that this schedule can be transformed into a non-preemptive schedule $S_{np}(m)$ on m machines such that

$$\frac{C_{\max}\left(S_{np}^*(m)\right)}{C_{\max}\left(S_{np}^*(\widehat{m})\right)} \leq \frac{C_{\max}\left(S_{np}(m)\right)}{C_{\max}\left(S_{np}^*(\widehat{m})\right)} \leq u + 1. \tag{11.4}$$

In schedule $S_{np}^*(\widehat{m})$, split the $\widehat{m}$ machines into v groups of $u+1$ machines and

$m - v$ groups of u machines, and number these groups arbitrarily by the integers $1, 2, \ldots, m$. Create a schedule $S_{np}(m)$ by assigning the jobs processed in schedule $S^*_{np}(\widehat{m})$ on the machines of the i-th group to machine M_i, $1 \leq i \leq m$. In the resulting schedule, there are v machines with a total load of at most $(u+1)\, C_{\max}\left(S^*_{np}(\widehat{m})\right)$ each and $m - v$ machines with a total load of at most $uC_{\max}\left(S^*_{np}(\widehat{m})\right)$ each. Thus, $C_{\max}\left(S_{np}(m)\right) \leq (u+1)\, C_{\max}\left(S^*_{np}(\widehat{m})\right)$, as required.

To see that (11.3) is a tight bound consider an instance with $\widehat{m} = um + v$ jobs, each of duration m. In this case, in any optimal schedule $S^*_{np}(m)$ on m machines there are v machines processing $u+1$ jobs each and $m - v$ machines processing u jobs each, so that $C_{\max}\left(S^*_{np}(m)\right) = (u+1)\, m$. On the other hand, in schedule $S^*_{np}(\widehat{m})$ each machine processes exactly one job, i.e., $C_{\max}\left(S^*_{np}(\widehat{m})\right) = m$.

Thus, the bound of $u+1$ on the machine impact remains the same even if all jobs are identical. □

Notice that Theorem 11.1 is essentially an existence result; in particular it does not provide an algorithm for finding a schedule $S_{np}(m)$ that satisfies (11.4) without prior knowledge of schedule $S^*_{np}(\widehat{m})$.

Recall from Section 2.2.5, that to solve problem $Pm \,||\, C_{\max}$, Graham (1966) introduces Algorithm LS (List Scheduling) which delivers a heuristic schedule $S_{LS}(m)$ with a tight worst-case bound (2.6). Also recall, that if an LPT list scheduling algorithm is used, the resulting schedule $S_{LPT}(m)$, delivers a better bound (2.8).

In the following statement we further generalise this result on the performance of the list scheduling algorithm.

Theorem 11.2. *For problem $Pm \,||\, C_{\max}$, let $S_{LS}(m)$ be a schedule found by Algorithm LS. Then for any number $\widehat{m} \geq 1$ of machines the bound*

$$\frac{C_{\max}\left(S_{LS}(m)\right)}{C_{\max}\left(S^*_{np}(\widehat{m})\right)} \leq 1 + \frac{\widehat{m} - 1}{m} \tag{11.5}$$

holds and this bound is tight.

Notice that for $\widehat{m} = m$, Theorem 11.2 is equivalent to the classical list scheduling result by Graham (1966). For any $\widehat{m} \geq 1$, Theorem 11.2 can be deduced from the statements established by Azar, Epstein and van Stee (2000) and Brehob, Torng and Uthaisombut (2000), who in fact have proved that

$$\frac{C_{\max}\left(S_{LS}(\widehat{m})\right)}{C_{\max}\left(S^*_{np}(m)\right)} \leq 1 + \frac{m - 1}{\widehat{m}} \tag{11.6}$$

for $\widehat{m} \geq m$. It can be verified that (11.6) holds for $\widehat{m} < m$ as well, so that interchanging the roles of m and $\widehat{m}$ in (11.6) we obtain (11.5).

Since

$$\frac{C_{\max}\left(S_{np}^{*}(m)\right)}{C_{\max}\left(S_{np}^{*}(\widehat{m})\right)} \leq \frac{C_{\max}\left(S_{LS}(m)\right)}{C_{\max}\left(S_{np}^{*}(\widehat{m})\right)},$$

it follows that Theorem 11.2 provides an estimate of the upper bound of the machine impact $I\left(m, \widehat{m}\right)$. This bound, however, is not tight in general, since for $\widehat{m} = um + v$, $v > 1$, the right-hand side of (11.5) becomes $u + 1 + (v - 1)/m$, which is greater than the bound of $u + 1$ established in Theorem 11.1. On the other hand, for $v = 1$, we see that the inequality (11.4) holds. In particular, for $u = 1, v = 1$, we obtain that

$$I(m, m+1) \leq \frac{C_{\max}\left(S_{LS}(m)\right)}{C_{\max}\left(S_{np}^{*}(m+1)\right)} \leq 2.$$

Computing $I(m, m+1)$ is a problem that is of interest in its own right, since it measures the influence of adding one extra machine.

Notice that Brehob, Torng and Uthaisombut (2000) demonstrate that the upper bound in (11.6) can be improved, provided that not an arbitrary list schedule $S_{LS}(\widehat{m})$ on $\widehat{m} \geq m$ machines is used, but a schedule $S_{LPT}(\widehat{m})$ is used. The latter schedule is found by a version of the list scheduling algorithm that works with a list of jobs renumbered in accordance with the LPT rule. Brehob, Torng and Uthaisombut (2000) provide upper bounds on the ratio $C_{\max}\left(S_{LPT}(\widehat{m})\right)/C_{\max}\left(S_{np}^{*}(m)\right)$ for $\widehat{m} \geq m$. However, for our purposes, we need an upper bound on $C_{\max}\left(S_{LPT}(m)\right)/C_{\max}\left(S_{np}^{*}(\widehat{m})\right)$ without prior knowledge of schedule $S_{np}^{*}(\widehat{m})$. Unlike the case of the general list scheduling algorithm described above, the results of Brehob, Torng and Uthaisombut (2000) are not transferable to estimating the latter ratio.

Even if a non-preemptive schedule on m machines is found by the LPT rule, the established bound (11.4) on the machine impact cannot be guaranteed. Let $S_{LPT}(m)$ be a schedule on m machines found by the LPT list scheduling algorithm, while $S_{np}^{*}(\widehat{m})$ be an optimal schedule on $\widehat{m} = um + v$ machines, where u and v are integers such that $u \geq 1$, $1 \leq v \leq m$. Below we demonstrate that there are instances of the problem for which

$$\frac{C_{\max}\left(S_{LPT}(m)\right)}{C_{\max}\left(S_{np}^{*}(\widehat{m})\right)} = u + 1 + \frac{1}{3m}. \tag{11.7}$$

Assume that $v > \frac{2m+1}{3}$ and consider an instance with $(u+1)\,m + 3v - 1$ jobs which consists of:

- $(u-1)\,m$ jobs of duration $3m$ each;

- m pairs of jobs of durations $2m, 2m-1, \ldots, m+1$;
- $3v-1$ jobs of duration m.

There exists a schedule $S^*(\widehat{m})$ with $C_{\max}(S^*(\widehat{m})) = 3m$. In this schedule, each of $(u-1)\,m$ machines processes a job of duration $3m$, and each of the $v-1$ machines processes three jobs of duration m each. The remaining jobs are in fact $m+1$ pairs of jobs of duration $2m, 2m-1, \ldots, m+1, m$ and they are distributed over the remaining $m+1$ machines in such a way that each of these machines processes exactly two jobs with the total processing time of $3m$. If m is even, the respective pairs of jobs are given by

$$(2m, m), (2m, m), (2m-1, m+1), (2m-1, m+1), \cdots,$$
$$\left(\frac{3}{2}m+1, \frac{3}{2}m-1\right), \left(\frac{3}{2}m+1, \frac{3}{2}m-1\right), \left(\frac{3}{2}m, \frac{3}{2}m\right),$$

while if m is odd, the following pairs are used:

$$(2m, m), (2m, m), (2m-1, m+1), (2m-1, m+1), \cdots,$$
$$\left(\frac{3m+1}{2}, \frac{3m-1}{2}\right), \left(\frac{3m+1}{2}, \frac{3m-1}{2}\right).$$

In schedule $S_{LPT}(m)$ the jobs are taken in the LPT order and are assigned to m machines. Thus, in $S_{LPT}(m)$, on each of the m machines, there will be $(u-1)$ jobs of duration $3m$ each, such that each of the m machines have a slot of duration $3m\,(u-1)$. The next slot is of total duration $3m+1$; here each of the machines processes exactly two jobs with the total processing time of $3m+1$. If m is even, the following pairs are assigned to the m machines

$$(2m, m+1), (2m, m+1), (2m-1, m+2), (2m-1, m+2), \cdots,$$
$$\left(\frac{3}{2}m+1, \frac{3}{2}m\right), \left(\frac{3}{2}m+1, \frac{3}{2}m\right),$$

while if m is odd, the following assignment is used:

$$(2m, m+1), (2m, m+1), (2m-1, m+2), (2m-1, m+2), \cdots, \left(\frac{3m+1}{2}, \frac{3m+1}{2}\right).$$

Since $3v-1 > 2m$, the remaining $3v-1$ jobs each of duration m are processed in three time slots of length m each. Thus, we have that $C_{\max}(S^*(\widehat{m})) = 3m$ and $C_{\max}(S_{LPT}(m)) = 3m\,(u-1) + (3m+1) + 3m = 3m\,(u+1) + 1$, so that for the

described instance, (11.7) holds.

11.3 Estimating Machine Impact: Total Flow Time

In this section, we consider problem $Pm\,||\,F$ with $F = \sum C_j$, i.e., the problem of minimising the total flow time on m identical parallel machines M_1, $M_2, \ldots, M_m$. We derive bounds on the impact that adding extra machines may have on this objective function.

There are two points of differences between problem $Pm\,||\sum C_j$ and problem $Pm\,||\,C_{\max}$ from Section 11.2:

- as proved by McNaughton (1959), for problem $Pm\,||\sum C_j$ preemption, if allowed, does not reduce the value of the function, so that we may consider only non-preemptive schedules;
- as proved by Conway, Maxwell and Miller (1967), problem $Pm\,||\sum C_j$ is solvable in polynomial time.

For problem $Pm\,||\sum C_j$, let $S^*(m)$ be an optimal schedule and let $G(S^*(m))$ denote the optimal value of the total flow time. It follows from (2.15) that

$$G(S^*(m)) = \sum_{j=1}^{n} C_j(S^*(m)) = \sum_{j=1}^{n} p_j \left\lceil \frac{j}{m} \right\rceil, \tag{11.8}$$

provided that the jobs are numbered in accordance with the LPT rule (2.7).

Thus, the machine impact $I(m, \widehat{m})$ for problem $Pm\,||\sum C_j$ can be written as

$$I(m, \widehat{m}) = \frac{G(S^*(m))}{G(S^*(\widehat{m}))} = \frac{\sum_{j=1}^{n} p_j \left\lceil \frac{j}{m} \right\rceil}{\sum_{j=1}^{n} p_j \left\lceil \frac{j}{\widehat{m}} \right\rceil}.$$

To derive bounds on $I(m, \widehat{m})$ for problem $Pm\,||\sum C_j$, let us first start with finding the bounds on $I(m, m+1)$, for problem $Pm\,|p_j = 1|\sum C_j$ with unit processing times. Later, we shall extend the obtained results to the general case.

11.3.1 Unit Processing Times

In this subsection, we derive both upper and lower bounds on the machine impact $I(m, m+1)$, provided that $p_j = 1$ for each job $j \in N$.

In our working, we use a closed form formula for computing the total flow time $G(S^*(m))$ for an optimal schedule on m machines.

Lemma 11.1. *For problem* $Pm\,|p_j = 1|\sum C_j$ *the objective function can be computed by*

$$G(S^*(m)) = \sum_{j=1}^{n} \left\lceil \frac{j}{m} \right\rceil = \left(\left\lfloor \frac{n}{m} \right\rfloor + 1 \right) \left(n - \frac{m}{2} \left\lfloor \frac{n}{m} \right\rfloor \right). \tag{11.9}$$

Proof: The proof is similar to that of Theorem 5.1 in Chapter 5. Recall that

$$G(S^*(m)) = \sum_{j=1}^{n} \left\lceil \frac{j}{m} \right\rceil$$

due to (11.8). Assume that $j = km + r$, where $k \in \left\{0, 1, \ldots, \left\lfloor \frac{n}{m} \right\rfloor - 1\right\}$, $1 \le r \le m$, and rewrite

$$\begin{aligned} G(S^*(m)) &= \sum_{k=0}^{\left\lfloor \frac{n}{m} \right\rfloor - 1} \sum_{r=1}^{m} \left\lceil \frac{km + r}{m} \right\rceil + \sum_{r=1}^{n - m\left\lfloor \frac{n}{m} \right\rfloor} \left\lceil \frac{\left\lfloor \frac{n}{m} \right\rfloor m + r}{m} \right\rceil \\ &= \sum_{k=0}^{\left\lfloor \frac{n}{m} \right\rfloor - 1} \sum_{r=1}^{m} \left(k + \left\lceil \frac{r}{m} \right\rceil \right) + \sum_{r=1}^{n - m\left\lfloor \frac{n}{m} \right\rfloor} \left(\left\lfloor \frac{n}{m} \right\rfloor + \left\lceil \frac{r}{m} \right\rceil \right). \end{aligned}$$

Since $\left\lceil \frac{r}{m} \right\rceil = 1$, we deduce

$$\begin{aligned} G(S^*(m)) &= \sum_{k=0}^{\left\lfloor \frac{n}{m} \right\rfloor - 1} \sum_{r=1}^{m} (k+1) + \sum_{r=1}^{n - m\left\lfloor \frac{n}{m} \right\rfloor} \left(\left\lfloor \frac{n}{m} \right\rfloor + 1 \right) \\ &= m \sum_{k=0}^{\left\lfloor \frac{n}{m} \right\rfloor - 1} (k+1) + \left(n - m \left\lfloor \frac{n}{m} \right\rfloor \right) \left(\left\lfloor \frac{n}{m} \right\rfloor + 1 \right), \end{aligned}$$

and the required formula (11.9) follows immediately. □

Upper bound

The purpose of this subsection is to prove the following statement.

Theorem 11.3. *For problem* $Pm\,|p_j = 1|\sum C_j$ *the bound on the machine impact*

$$I\,(m, m+1) = \frac{\sum_{j=1}^{n} \left\lceil \frac{j}{m} \right\rceil}{\sum_{j=1}^{n} \left\lceil \frac{j}{m+1} \right\rceil} \leq \frac{m+1}{m}. \tag{11.10}$$

Define the sequence

$$A(j) = m \left\lceil \frac{j}{m} \right\rceil - (m+1) \left\lceil \frac{j}{m+1} \right\rceil,\ j \in N. \tag{11.11}$$

To prove (11.10), we show that

$$\sum_{j=1}^{n} A(j) \leq 0. \tag{11.12}$$

We start with the statement that shows that the sequence $A(j)$, $j \in N$, is periodic.

Lemma 11.2. *The sequence* $A(j)$, $1 \leq j \leq n$, *defined by (11.11) is periodic with a period of* $m(m+1)$, *so that for each* $j \leq n - m\,(m+1)$ *the equality* $A(j + m(m+1)) = A(j)$ *holds.*

Proof: Take an arbitrary j, $1 \leq j \leq n - m\,(m+1)$. The following argument

$$\begin{aligned} A(j + m(m+1)) &= m \left\lceil \frac{j + m(m+1)}{m} \right\rceil - (m+1) \left\lceil \frac{j + m(m+1)}{m+1} \right\rceil \\ &= m \left\lceil \frac{j}{m} \right\rceil + m\,(m+1) - (m+1) \left\lceil \frac{j}{m+1} \right\rceil - m\,(m+1) \\ &= m \left\lceil \frac{j}{m} \right\rceil - (m+1) \left\lceil \frac{j}{m+1} \right\rceil = A(j) \end{aligned}$$

proves the lemma. □

Consider the first $m\,(m+1)$ elements of sequence $A(j)$, $1 \leq j \leq n$, and call it a *block.* It follows from Lemma 11.2 that the sequence $A(j)$, $1 \leq j \leq n$, is a collection of several blocks each containing $m\,(m+1)$ elements, possibly followed by an incomplete block with less than $m\,(m+1)$ elements. It appears that each full block can be further subdivided into m sequences of $m+1$ elements each, which we call *patterns*. For a block $A(j)$, $1 \leq j \leq m\,(m+1)$, its q-th pattern is given by the elements $\{(q-1)\,(m+1) + 1, (q-1)\,(m+1) + 2, \ldots, q\,(m+1)\}$, where $1 \leq q \leq m$.

The following statement proves that the sum of the elements in each pattern is negative.

Lemma 11.3. *For each pattern q, $1 \leq q \leq m$, of the first block of elements $A(j)$, $1 \leq j \leq m\,(m+1)$, of sequence (11.11), the relation*

$$\sum_{j=(q-1)(m+1)+1}^{q(m+1)} A(j) = -q < 0$$

holds.

Proof: It is easy to see that for each $j \in \{(q-1)\,(m+1)+1, \ldots, q(m+1)\}$ within the pattern, the value of $\left\lceil \frac{j}{m+1} \right\rceil$ is equal to q. However, the value of the expression $\left\lceil \frac{j}{m} \right\rceil$ does not remain constant for the entire range of j within the pattern and is given by

$$\left\lceil \frac{j}{m} \right\rceil = \begin{cases} q, & j \in \{(q-1)\,(m+1)+1, \ldots, qm\} \\ q+1, & j \in \{qm+1, \ldots, q(m+1)\} \end{cases}.$$

We rewrite

$$\begin{aligned} \sum_{j=(q-1)(m+1)+1}^{q(m+1)} A(j) &= \sum_{j=(q-1)(m+1)+1}^{qm} \left(m \left\lceil \frac{j}{m} \right\rceil - (m+1) \left\lceil \frac{j}{m+1} \right\rceil \right) \\ &\quad + \sum_{j=qm+1}^{q(m+1)} \left(m \left\lceil \frac{j}{m} \right\rceil - (m+1) \left\lceil \frac{j}{m+1} \right\rceil \right) \\ &= \sum_{j=(q-1)(m+1)+1}^{qm} (mq - (m+1)\,q) + \sum_{j=qm+1}^{q(m+1)} (m\,(q+1) - (m+1)\,q)\,. \end{aligned}$$

In the right-hand side of the last expression, the first term reduces to $(-q)$ taken $m-q+1$ times, and the second term reduces to $m-q$ taken q times, so that

$$\sum_{j=(q-1)(m+1)+1}^{(q+1)m} A(j) = -\,(m-q+1)\,q + q\,(m-q) = -q < 0,$$

which proves the lemma. □

In sequence (11.11), the sum of the elements of each complete block is equal to the sum of the elements of all its patterns and is, therefore, negative due to Lemma 11.3. Besides, in sequence (11.11) the last block may be incomplete, i.e., may contain less than $m\,(m+1)$ elements, or, equivalently, less than m complete patterns of $m+1$ elements each and one incomplete pattern, of less than $m+1$ elements. Due to Lemma 11.3, each complete pattern will make a negative contribution to the sum of the elements of sequence (11.11). As seen from the proof of Lemma 11.3, each in-

complete pattern will have less positive elements than a complete pattern. Thus, the sum of the elements of an incomplete pattern is less than what it would have been, if that pattern were complete. We conclude that an incomplete pattern also makes a negative contribution to the overall sum of the elements of sequence (11.11). Thus, (11.12) holds, and the required upper bound (11.10) is proved.

Lower bound

We now derive a lower bound on the value of the machine impact $I(m, m+1)$ for problem $Pm\,|p_j = 1|\sum C_j$; recall that we only need to consider the instance with $n > m$; otherwise the problem is trivial. Notice that $Pm\,|p_j = 1|\sum C_j$ is the only problem from the range under consideration for which a non-trivial lower bound on $I(m, \widehat{m})$ exists; for the rest of the problems the lower bound is 1 and is tight; see Section 11.2 for a discussion of a lower bound on the machine impact in the case of minimising the makespan.

Below we prove the following statement.

Theorem 11.4. *For problem $Pm\,|p_j = 1|\sum C_j$ the bound on the machine impact*

$$I\,(m, m+1) \geq \begin{cases} 1 + \frac{1}{3m-1}, & \text{if } n \text{ is even} \\ 1 + \frac{1}{3m-3}, & \text{if } n \text{ is odd} \end{cases} \tag{11.13}$$

holds.

We only prove the top inequality in (11.13) and demonstrate tightness of both bounds. Define the sequence

$$B(j) = (3m-1)\left\lceil \frac{j}{m} \right\rceil - 3m\left\lceil \frac{j}{m+1} \right\rceil,\ j \in N. \tag{11.14}$$

To prove the top inequality in (11.13), we show that

$$\sum_{j=1}^{n} B(j) \geq 0. \tag{11.15}$$

Similar to Lemma 11.2, we can prove that the sequence (11.14) is 'quasiperiodic'.

Lemma 11.4. *For each $j \leq n - m\,(m+1)$ the equality $B(j+m(m+1)) = B(j)+2m-1$ holds.*

The sequence of $B(j)$, $1 \leq j \leq n$, can be split into blocks of $m(m+1)$ elements each (plus, possibly, an incomplete last block) defined exactly as the blocks of the sequence $A(j)$, $1 \leq j \leq n$, in Section 11.3.1. Furthermore, each complete block can be split into m patterns of $m+1$ elements each, similar to the patterns of the sequence $A(j)$, $1 \leq j \leq n$, in Section 11.3.1.

Suppose that all patterns in the first block with the elements $1, 2, \ldots, m\,(m+1)$ are complete. The proof of Lemma 11.3 can be modified to show that the sum of a pattern q, $1 \leq q \leq m$, in the first block is equal to $2q\,(m-1)$, and the sum of the elements of the first block is equal to $\sum_{q=1}^{m} 2\,(m-1)\,q = m^3 - m$. Besides, it can be easily verified in light of Lemma 11.4, that each element of a block other than the first is strictly positive.

We only need to address a situation that the first block contains an incomplete pattern. As in Section 11.3.1, for the first block the q-th pattern starts with at most $m-q+1$ negative entries, each equal to $-q$, where q, $1 \leq q \leq m$. The first pattern in the first block must be complete, since otherwise $n < m+1$. If the first block contains an incomplete pattern $q = v$, $2 \leq v \leq m$, then the negative contribution from the v-th pattern is compensated by the positive contribution from the preceding complete patterns, since $\sum_{q=1}^{v-1} 2q\,(m-1) \geq (m-v+1)\,v$, where the equality holds for $v = 2$.

Thus, the inequality (11.15) holds and the required lower bound is proved. To establish the tightness, notice that the reasoning above implies that the sum of the elements $B(j)$, $1 \leq j \leq n$, is the smallest if its last element n is the $(m-1)$-th element of the second pattern of the first block, i.e., $n = (m+1) + (m-1) = 2m$. If n is even, then

$$\sum_{j=1}^{n} B(j) = 0,$$

and the impact factor $I(m, m+1)$ is equal to $1 + \frac{1}{3m-1}$. The behaviour of the machine impact and its bounds is illustrated by Figure 11.1.

If n is odd, we can apply a similar reasoning to demonstrate that $I(m, m+1)$ is bounded from below by $1 + \frac{1}{3m-3}$, the equality holds for $n = 2m-1$. Indeed, for this instance we have $\left\lfloor \frac{n}{m} \right\rfloor = \left\lfloor \frac{n}{m+1} \right\rfloor = 1$ and by (11.9) we deduce that

$$\begin{aligned} G(S^*(m)) &= 2\left((2m-1) - \frac{m}{2}\right) = 3m-2; \\ G(S^*(m+1)) &= 2\left((2m-1) - \frac{m+1}{2}\right) = 3m-3, \end{aligned}$$

as required.

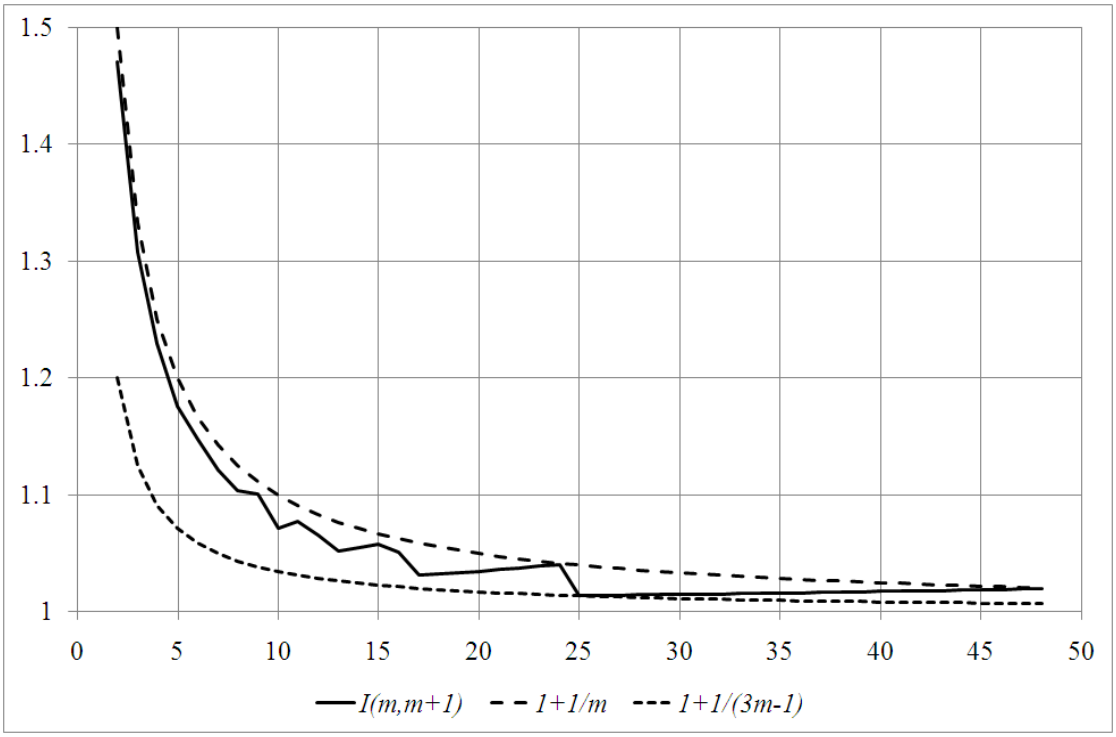


Figure 11.1: The graphs of $I(m, m+1)$ and its lower and upper bounds for problem $Pm\,|p_j = 1|\sum C_j$ with $n = 50$

Furthermore, we can use (11.9) to explain the behaviour of function $I(m, m+1)$. Indeed, for the values of m larger than $n/2$, we derive that $G(S^*(m)) = 2n - m$ and $G(S^*(m+1)) = 2n - m - 1$, so that $I(m, m+1)$ is monotone increasing. However, for m smaller than $n/2$ the function $I(m, m+1)$ is not monotone, which is due to 'jumps' in the values $\lfloor \frac{n}{m} \rfloor$ and $\lfloor \frac{n}{m+1} \rfloor$, which contribute to $G(S^*(m))$ and $G(S^*(m+1))$, respectively. More precisely, it can be proved that when for some $m = m'$ the inequalities $I(m'-1, m') > I(m', m'+1)$ and $I(m', m'+1) < I(m'+1, m'+2)$ hold, i.e., when $I(m', m'+1)$ is a local minimum, then $m' = \lceil n/r \rceil$ for some integer r, with the global minimum achieved for $r = 2$.

11.3.2 Arbitrary Processing Times

We first prove that $\frac{m+1}{m}$ remains an upper bound on the machine impact $I(m, m+1)$ for problem $Pm\,||\sum C_j$ with arbitrary processing times, and then extend this result for the general impact $I(m, \widehat{m})$.

Recall that the jobs are numbered in the LPT order, in accordance with (2.7).

What we need to prove in order to estimate $I(m, m+1)$ is the inequality

$$\sum_{j=1}^{n} p_j A_j \leq 0,$$

where $A(j)$, $1 \leq j \leq n$, is the sequence defined by (11.11). Also recall that the latter sequence is periodic by Lemma 11.2 and can be split into blocks, each of which is in turn split into patterns; see Section 11.3.1 for details.

We start with looking at a complete pattern of the first block.

Lemma 11.5. *Let $\{(q-1)(m+1)+1, \ldots, q(m+1)\}$ be the q-th pattern of the first block of sequence (11.11), and that pattern is complete. Then*

$$\sum_{j=(q-1)(m+1)+1}^{q(m+1)} p_j A_j \leq -q p_{qm} < 0.$$

Proof: As follows from the proof of Lemma 11.3, each of the first $m-q+1$ elements of the pattern of sequence (11.11) under consideration is equal to $-q$, while each of the remaining q elements is equal to $m-q$. Thus,

$$\sum_{j=(q-1)(m+1)+1}^{q(m+1)} p_j A_j = -q \sum_{j=(q-1)(m+1)+1}^{qm} p_j + (m-q) \sum_{j=qm+1}^{q(m+1)} p_j.$$

Due to the LPT numbering of the jobs, we deduce

$$\begin{aligned}
\sum_{j=(q-1)(m+1)+1}^{q(m+1)} p_j A_j &\leq -q(m-q+1) \min\{p_j | (q-1)(m+1)+1 \leq j \leq qm\} + \\
&\quad + (m-q)\, q \max\{p_j | qm+1 \leq j \leq q(m+1)\} \\
&= -q(m-q+1)\, p_{qm} + (m-q)\, q p_{qm+1} \\
&\leq -q(m-q+1)\, p_{qm} + (m-q)\, q p_{qm} = -q p_{qm} < 0,
\end{aligned}$$

as required. □

Now we are ready to present the main result of this section.

Theorem 11.5. *For problem $Pm \,||\, \sum C_j$, the following bound holds*

$$I(m, \widehat{m}) = \frac{\sum C_j\,(S^*(m))}{\sum C_j\,(S^*(\widehat{m}))} \leq \frac{\widehat{m}}{m} \tag{11.16}$$

and this bound is asymptotically tight.

Proof: For $\widehat{m} = m + 1$ the theorem follows immediately from Lemma 11.5. Due to the periodic nature of the sequence (11.11) guaranteed by Lemma 11.2, we can extend Lemma 11.5 to any complete pattern of any block. If there is an incomplete pattern that finishes the sequence (11.11), then the sum of products $p_j A_j$ for j that belong to that pattern can easily be proved negative. As proved in Section 11.3.1, such a pattern of the sequence (11.11) either contains only negative values or all negative values and less positive values than it would if it were complete. This implies that $I(m, m+1) \leq (m+1)/m$.

Multiple application of the latter inequality gives us

$$\begin{aligned} G(S^*(m)) &\leq \frac{m+1}{m} G(S^*(m+1)) \leq \frac{m+1}{m}\frac{m+2}{m+1} G(S^*(m+2)) \\ &\leq \frac{(m+1)(m+2)\cdots\widehat{m}}{m(m+1)\cdots(\widehat{m}-1)} G(S^*(\widehat{m})), \end{aligned}$$

so that (11.16) holds.

To see that this bound is asymptotically tight, consider an instance of problem $Pm\,||\sum C_j$ with $n = Wm(m+y)$ jobs of unit duration each, where $y \geq 1$ and W is a large positive number. We compare $G(S^*(m))$ and $G(S^*(\widehat{m}))$ for $\widehat{m} = m + y$. For this instance, we use (11.9) to compute

$$\begin{aligned} G(S^*(m)) &= \frac{mW(m+y)(Wm+Wy+1)}{2}; \\ G(S^*(m+y)) &= \frac{(m+y)\,Wm(Wm+1)}{2}. \end{aligned}$$

Thus, as $W \to \infty$ the ratio

$$I(m, \widehat{m}) = \frac{(Wm+Wy+1)}{(Wm+1)}$$

approaches $\frac{m+y}{m} = \frac{\widehat{m}}{m}$. □

It is easy to verify that for problem $Pm\,||\sum C_j$, a tight lower bound on $I(m, \widehat{m})$ is 1, which is achieved for instances with a very long job.

11.4 Cost-Effective Choice of The Number of Machines: Makespan

In the previous sections, we have derived bounds on the machine impact that show how a scheduling performance measure F (either the makespan or the total flow time) is affected by the arrival of extra machines. In reality, adding a machine cannot be seen free. From now on, we address the problem of making a cost-effective choice of the number of machines to minimise a function that captures the trade-off between the value of F and the cost of using the machines.

Formally, assume that using a machine incurs a cost K, and we are interested in minimising the total cost function

$$\Phi(m) = w_1 F(S^*(m)) + w_2 K m,$$

where $S^*(m)$ denotes an optimal schedule with m parallel machines to minimise an objective $F \in \{C_{\max}, \sum C_j\}$, while w_1 and w_2 represent positive weights associated with the contribution of the scheduling objective and the machine cost, respectively. The above objective function can be normalised and be rewritten as

$$\Phi(m) = F(S^*(m)) + \kappa m, \tag{11.17}$$

where the normalised cost of using each machine is denoted as $\kappa := \frac{w_2 K}{w_1}$.

In this section, we handle the problem with $F = C_{\max}$, for both the preemptive and non-preemptive cases. For the preemptive case, we present a linear time algorithm for finding the number of machines that minimises function (11.17). For the non-preemptive case, we analyse a worst-case behaviour of an approximation algorithm that accepts the number of machines that is optimal for the preemptive counterpart.

11.4.1 Preemption Allowed

We start with the preemptive version of the problem of minimising the total cost. We denote the total cost function by $\Phi_p(m)$, provided that the scheduling objective is the makespan; the subscript "p" is used to indicate preemption. Thus,

$$\Phi_p(m) = C_{\max}(S_p^*(m)) + \kappa m,$$

where $S_p^*(m)$ is an optimal preemptive schedule on m machines. Let m_p^* denote the optimal number of machines, i.e., $\Phi_p(m_p^*) \leq \Phi_p(m)$ for all values of $m \geq 1$. Recall that for the optimal makespan for problem $Pm\,|pmtn|\,C_{\max}$, the lower bounds (2.9) and (2.10) hold.

Define m_1 as the smallest number of machines for which the duration of the longest job is either larger than or equal to the average machine load, i.e.,

$$m_1 = \left\lceil \frac{P}{p_{\max}} \right\rceil. \tag{11.18}$$

Thus, we can consider function $\Phi_p(m)$ as

$$\Phi_p(m) = \begin{cases} \Gamma_1(m), & \text{for } m \geq m_1 \\ \Gamma_2(m), & \text{for } 1 \leq m < m_1, \end{cases}$$

where

$$\begin{aligned} \Gamma_1(m) &: = p_{\max} + \kappa m; \\ \Gamma_2(m) &: = \frac{P}{m} + \kappa m. \end{aligned} \tag{11.19}$$

Obviously, m_1 minimises $\Gamma_1(m)$, since adding another machine incurs extra cost without changing the makespan $p_{\max}$. Let m_2 be such that

$$\Gamma_2(m_2) = \min\{\Gamma_2(m) \,|\, 1 \leq m < m_1\}.$$

If we have $\Gamma_1(m_1) \leq \Gamma_2(m_2)$, then $m_p^* = m_1$, otherwise, $m_p^* = m_2$.

Notice that function $\Gamma_2(m)$ is exactly of the same shape as the total cost function of the EOQ model of inventory control, this topic being a part of any standard OR curriculum; see, e.g., Winston (1994). Function $\Gamma_2(m)$ is convex and its global minimum is achieved for $m = \mu$, found by the formula, similar to the famous "square root" EOQ formula:

$$\mu = \sqrt{\frac{P}{\kappa}}. \tag{11.20}$$

In what follows, we assume that the coefficient κ is such that $\mu < n$; otherwise function $\Gamma_2(m)$ reaches its minimum for $m = n$. Notice that $\frac{P}{\mu} = \kappa\mu$, so that the smallest value of $\Gamma_2(m)$ is equal to $2\kappa\mu$. Notice that for $\kappa = 1$, the value of $2\kappa\mu$ becomes $2\sqrt{P}$, which complies with a lower bound on the total cost used by Imreh and Noga (1999) as well as in other online studies cited in the introduction.

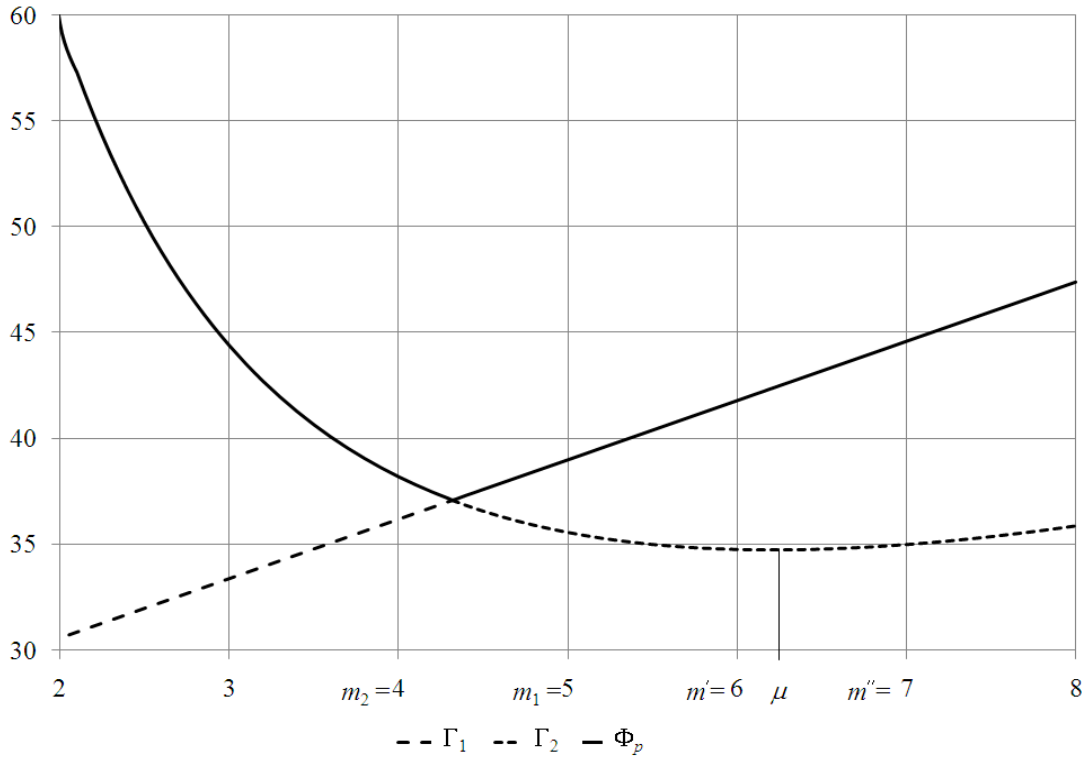


Figure 11.2: An example of graphs of the functions $\Gamma_1(m)$, $\Gamma_2(m)$ and $\Phi_p(m)$

Due to the convexity of $\Gamma_2(m)$, we observe that

$$\begin{aligned} \frac{P}{m} &< \kappa m, \ m < \mu; \\ \frac{P}{m} &> \kappa m, \ m > \mu. \end{aligned} \tag{11.21}$$

If $\mu \geq m_1$, then function $\Gamma_2(m)$ decreases for $m \in \{1, 2, \ldots, m_1 - 1\}$, so that $\Gamma_2(m_1 - 1)$ might be smaller than $\Gamma_1(m_1)$. For $\mu < m_1$, if μ happens to be integer, we take $m_2 = \mu$. Otherwise, due to the convexity of function $\Gamma_2(m)$, its integer minimum is delivered either by $m' = \lfloor \mu \rfloor$ or $m'' = \lceil \mu \rceil$, depending on which of these two values returns a lower value of function $\Gamma_2(m)$.

Figure 11.2 illustrates how function $\Phi_p(m)$ is formed. In the taken instance, $\kappa = 2.8$, $P = 108$ and $p_{\max} = 25$. In this case the graphs of $\Gamma_1(m)$ and $\Gamma_2(m)$ intersect at $m = 4.32$, while $\mu = 6.21059$. In this case, $m_2 < m_1 < m' < m''$. For other instances other relative orders of these four values are possible. Algorithm 1 below describes how to find the one that delivers the minimum of function $\Phi_p(m)$.

Algorithm 1

INPUT: Jobs of set $N = \{1, 2, \ldots, n\}$; normalised cost κ of using each machine.

OUTPUT: The cost-optimal number of machines m_p^*.

Step 1. Determine the values of P and $p_{\max}$. Compute m_1 by (11.18) and define $m_p^* := m_1$.

Step 2. Compute μ by (11.20). If $\lceil \mu \rceil < m_1$, go to Step 3; otherwise define $m_2 = m_1 - 1$, if $\Gamma_2(m_2) < \Gamma_1(m_1)$ redefine $m_p^* := m_2$. Go to Step 4.

Step 3. Compute $m' := \lfloor \mu \rfloor$ and $m'' := \lceil \mu \rceil$. Define $m_p^* := m'$ and, if $\Gamma_2(m'') < \Gamma_2(m')$, redefine $m_p^* := m''$.

Step 4. Output m_p^* and $\Phi_p(m_p^*)$. Stop.

The running time of the Algorithm 1 is $O(n)$, provided the square root operation takes constant time. The algorithm outputs the optimal number of machines m_p^* and the optimal total cost $\Phi_p(m_p^*)$. The corresponding optimal schedule $S_p^*(m_p^*)$ can be found by McNaughton's algorithm in $O(n)$ time. For the instance illustrated in Figure 11.2 Algorithm 1 outputs $m_p^* = m_2 = 4$ found in Step 2.

Thus, the following statement holds.

Theorem 11.6. *Algorithm 1 finds the optimal number of machines m_p^*, the optimal cost $\Phi_p(m_p^*)$ and the corresponding preemptive optimal schedule $S_p^*(m_p^*)$ in $O(n)$ time.*

11.4.2 No Preemption Allowed

Now we pass to the non-preemptive version of the problem. We only may look for an approximation algorithm, since problem $Pm \,||\, C_{\max}$ is NP-hard. We denote the total cost function by $\Phi_{np}(m)$; the subscript "np" is used to indicate that no preemption is allowed. Thus,

$$\Phi_{np}(m) = C_{\max}(S_{np}^*(m)) + \kappa m,$$

where $S_{np}^*(m)$ is an optimal non-preemptive schedule on m machines. Let m_{np}^* denote the optimal number of machines, i.e., $\Phi_{np}(m_{np}^*) \leq \Phi_{np}(m)$ for all values of $m \geq 1$.

As a part of our algorithm we need to be able to describe a procedure for finding a non-preemptive schedule with a given number of machines and compare the found makespan with the makespan of the optimal preemptive schedule with the same number of machines. This brings us to a discussion of the power of preemption in the context of scheduling on identical parallel machines.

Recall from Section 2.2.5, that for problem $Pm \,||\, C_{\max}$ the power of preemption, i.e., the maximum ratio $C_{\max}(S_{np}^*(m))/C_{\max}(S_p^*(m))$ across all instances of the problem at

hand is given by (2.11). In particular, Braun and Schmidt (2003) show that the bound (2.11) holds if an optimal non-preemptive schedule $S^*_{np}(m)$ is replaced by a heuristic schedule $S_{LPT}(m)$, which is found by an LPT list scheduling algorithm. Note that it requires $O(n \log n + nm)$ time to compute a heuristic schedule $S_{LPT}(m)$.

Below we present an algorithm that in $O(nm)$ time outputs a non-preemptive schedule that allows us to give better estimates of the power of preemption.

Algorithm 2

INPUT: Jobs of set $N = \{1, 2, \ldots, n\}$; m identical parallel machines.

OUTPUT: A heuristic schedule $S_{np}(m)$.

Step 1. Compute P, and find p_m, the m-th largest value among p_j, $j \in N$. Identify $m-1$ values of the processing times for which $p_j \geq p_m$ and renumber the jobs in such a way that

$$p_1 \geq p_2 \geq \cdots \geq p_m,$$

while the remaining jobs are taken in an arbitrary order. Determine the index g, $0 \leq g \leq m-1$, as the smallest index such that

$$p_{g+1} < \frac{P - \sum_{j=1}^{g} p_j}{m-g}$$

Step 2. For $1 \leq j \leq g$, taking the jobs in the order of the obtained numbering, assign job j to machine M_j. Complete the obtained partial schedule by assigning the remaining jobs to machines $M_{g+1}, \ldots, M_m$ by Algorithm LS. Call the resulting schedule $S_{np}(m)$ and stop.

Finding p_m in Step 1 requires $O(n)$ time by the median technique. Renumbering the jobs needs $O(m \log m + n)$ time, determining g takes $O(m)$ time and Step 2 needs $O(nm)$ time. Thus, the overall running time of the algorithm is $O(m(n + \log m)) = O(nm)$, due to $n \geq m$.

Theorem 11.7. *For the problem of scheduling n jobs on m identical parallel machines Algorithm 2 finds a non-preemptive schedule $S_{np}(m)$ for which either $C_{\max}(S_{np}(m)) = p_1$ or*

$$C_{\max}(S_{np}(m)) \leq \left(2 - \frac{2}{m-g+1}\right) \frac{P - \sum_{j=1}^{g} p_j}{m-g}, \tag{11.22}$$

the latter bound being tight.

Proof: The makespan of the obtained schedule $S_{np}(m)$ is either equal to the largest load of machines $M_1, \ldots, M_g$ or to the largest load of machines $M_{g+1}, \ldots, M_m$. In the former case, $C_{\max}(S_{np}(m)) = p_1$, i.e., this schedule is optimal. Otherwise, let k be a job that terminates schedule $S_{np}(m)$. Assume that job k starts at time τ, so that

$$C_{\max}\left(S_{np}(m)\right) = \tau + p_k.$$

Recall that the greedy nature of Algorithm LS implies that all $m - g$ machines are permanently busy in the time interval $[0, \tau]$, so that

$$\tau \leq \frac{P - \sum_{j=1}^{g} p_j - p_k}{m - g}.$$

Thus, we deduce

$$C_{\max}\left(S_{np}(m)\right) \leq \frac{P - \sum_{j=1}^{g} p_j}{m - g} + \frac{m - g - 1}{m - g} p_k.$$

Furthermore, $p_k \leq p_m \leq \cdots \leq p_{g+1}$, i.e.,

$$p_k \leq \frac{\sum_{j=g+1}^{m} p_j + p_k}{m - g + 1} \leq \frac{P - \sum_{j=1}^{g} p_j}{m - g + 1}.$$

Finally, we derive

$$\begin{aligned} C_{\max}\left(S_{np}(m)\right) &\leq \frac{P - \sum_{j=1}^{g} p_j}{m - g} + \left(\frac{m - g - 1}{m - g}\right) \frac{P - \sum_{j=1}^{g} p_j}{m - g + 1} \\ &= \left(1 + \frac{m - g - 1}{m - g + 1}\right) \frac{P - \sum_{j=1}^{g} p_j}{m - g} = \left(2 - \frac{2}{m - g + 1}\right) \frac{P - \sum_{j=1}^{g} p_j}{m - g}, \end{aligned}$$

which proves (11.22).

To see that the bound (11.22) is tight, take a positive integer m and a non-negative integer g less than m. Consider an instance with m machines and $m + 1$ jobs, that contains jobs $1, \cdots, g$ with the processing time $m - g + 1$ each, and jobs $g + 1, \cdots, m+1$ with the processing time $m - g$ each, where m is a positive integer and g is a non-negative integer less than m. We have that $P = g\,(m - g + 1) + (m - g + 1)\,(m - g) = m\,(m - g + 1)$ and $p_{\max} = m - g + 1$, so that $C_{\max}(S_p^*(m)) = m - g + 1$.

In an optimal non-preemptive schedule $S_{np}^*(m)$ each job $j, 1 \leq j \leq g$, is assigned to an individual machine; without loss of generality, this can be machine M_j. The remaining $m - g + 1$ jobs are processed on machines $M_{g+1}, \cdots, M_m$ and each of these

machines will process exactly one job, except one machine that will process two jobs. Notice that this schedule is exactly schedule $S_{np}(m)$ found by Algorithm 2. It can be checked that

$$\frac{C_{\max}\left(S_{np}(m)\right)}{\frac{P-\sum_{j=1}^{g} p_j}{m-g}} = \frac{C_{\max}(S^*_{np}(m))}{C_{\max}(S^*_p(m))} = \left(2 - \frac{2}{m-g+1}\right).$$

Indeed, $C_{\max}(S^*_{np}(m)) = C_{\max}(S_{np}(m)) = 2m - 2g$ and

$$\frac{P - \sum_{j=1}^{g} p_j}{m-g} = m - g + 1 = C_{\max}(S^*_p(m)),$$

so that

$$\left(2 - \frac{2}{m-g+1}\right) \frac{P - \sum_{j=1}^{g} p_j}{m-g} = 2m - 2g = C_{\max}(S_{np}(m)).$$

□

For our purposes, we need the following statement that follows from Theorem 11.7.

Corollary 11.1. *For the problem of scheduling n jobs on m identical parallel machines Algorithm 2 finds a non-preemptive schedule $S_{np}(m)$ such that either*

$$C_{\max}(S_{np}(m)) \leq \left(2 - \frac{2}{m+1}\right) \frac{P}{m}, \tag{11.23}$$

if $p_{\max} = p_1 \leq \frac{P}{m}$ and $g = 0$, or

$$C_{\max}(S_{np}(m)) \leq \left(2 - \frac{2}{m-g+1}\right) p_g, \tag{11.24}$$

if $p_{\max} = p_1 > \frac{P}{m}$ and $g \geq 1$.

Proof: The bound (11.23) emerges if Theorem 11.7 is applied with $g = 0$. On the other hand, from Theorem 11.7 for $g \geq 1$, we get

$$C_{\max}(S_{np}(m)) \leq \left(2 - \frac{2}{m-g+1}\right) \frac{P - \sum_{j=1}^{g} p_j}{m-g}.$$

Since by definition of g the inequality

$$p_g > \frac{P - \sum_{j=1}^{g-1} p_j}{m-g+1},$$

holds, we deduce that

$$p_g\,(m-g) > P - \sum_{j=1}^{g} p_j,$$

which leads to (11.24). □

Since P/m and p_g are lower bounds on the value of $C_{\max}(S_p(m))$, the inequalities (11.23) and (11.24) give refined tight upper bounds on the power of preemption for identical machines.

Now we come back to designing an approximation algorithm for minimising the total cost function $\Phi_{np}\,(m)$. Our algorithm will take the candidate number of machines found by Algorithm 1 and then use Algorithm 2 to find a non-preemptive schedule with the chosen number of machines.

Algorithm 3

INPUT: Jobs of set $N = \{1, 2, \ldots, n\}$; normalised cost κ of using each machine.

OUTPUT: A cost-optimal heuristic solution for the problem of minimising the makespan of the non-preemptive case.

Step 1. Run Algorithm 1 to find the number of machines m_p^* that is optimal for the preemptive version of our problem. Define $m^H := m_p^*$.

Step 2. Run Algorithm 2 to find a non-preemptive schedule $S_{np}\left(m^H\right)$. Output m^H, $C_{\max}\left(S_{np}\left(m^H\right)\right)$ and $\Phi_{np}\left(m^H\right)$.

Step 1 of Algorithm 3 requires $O(n)$ time, and then Step 2 applied with the found m^H additionally takes $O(n)$ time. It appears that the worst-case performance of Algorithm 3 depends in which step of Algorithm 1 the value m_p^* is found; in other words, that depends on the sign of the difference $m_1 - \lceil \mu \rceil$. This is why our analysis of Algorithm 3 is done in two separate statements.

Theorem 11.8. *For $m^H = m_p^* \in \{m_1, m_2\}$ found in Step 2 of Algorithm 1 the bound*

$$\frac{\Phi_{np}(m^H)}{\Phi_{np}\left(m_{np}^*\right)} \leq 2 - \frac{2}{m_1} \tag{11.25}$$

holds, and this bound is tight.

Proof: Since the makespan of the best non-preemptive schedule with m machines is no smaller than the makespan for the best preemptive schedule with the same number of machines, it follows that $\Phi_{np}\left(m_{np}^*\right) \geq \Phi_p\left(m_{np}^*\right) \geq \Phi_p\left(m_p^*\right)$.

If $m_p^* = m_2 = m_1 - 1$ (see Figure 11.2 for an illustration), then $C_{\max}\left(S_p^*(m_2)\right) = P/m_2 > p_{\max}$ and we deduce

$$\frac{\Phi_{np}\left(m^H\right)}{\Phi_{np}\left(m_{np}^*\right)} \leq \frac{\Phi_{np}\left(m_2\right)}{\Phi_p\left(m_2\right)} = \frac{\Phi_{np}\left(m_2\right)}{\Gamma_2\left(m_2\right)} = \frac{C_{\max}\left(S_{np}^*(m_2)\right) + \kappa m_2}{\frac{P}{m_2} + \kappa m_2}.$$

Algorithm 2 applied to $m = m_2$ will find $g = 0$, so that (11.23) holds for schedule $S_{np}(m_2)$. Thus,

$$\frac{\Phi_{np}\left(m^H\right)}{\Phi_{np}\left(m_{np}^*\right)} \leq \frac{C_{\max}\left(S_{np}(m_2)\right) + \kappa m_2}{\frac{P}{m_2} + \kappa m_2} \leq \frac{\left(2 - \frac{2}{m_2+1}\right)\frac{P}{m_2} + \kappa m_2}{\frac{P}{m_2} + \kappa m_2}.$$

The last fraction decreases in κm_2, so that we can replace it by zero to achieve

$$\frac{\Phi_{np}\left(m^H\right)}{\Phi_{np}\left(m_{np}^*\right)} \leq \frac{\left(2 - \frac{2}{m_2+1}\right)\frac{P}{m_2}}{\frac{P}{m_2}} = 2 - \frac{2}{m_2+1} = 2 - \frac{2}{m_1},$$

as required.

If $m_p^* = m_1$ then $C_{\max}\left(S_p^*(m_1)\right) = p_{\max} \geq P/m_1$ and we deduce

$$\frac{\Phi_{np}\left(m^H\right)}{\Phi_{np}\left(m_{np}^*\right)} \leq \frac{\Phi_{np}\left(m_1\right)}{\Phi_p\left(m_1\right)} = \frac{\Phi_{np}\left(m_1\right)}{\Gamma_1\left(m_1\right)} = \frac{C_{\max}\left(S_{np}^*(m_1)\right) + \kappa m_1}{p_{\max} + \kappa m_1}.$$

Algorithm 2 applied to $m = m_1$ will find that $g \geq 1$, so that (11.24) holds for schedule $S_{np}(m_1)$. Therefore,

$$\frac{\Phi_{np}\left(m^H\right)}{\Phi_{np}\left(m_{np}^*\right)} \leq \frac{p_g\left(2 - \frac{2}{m_1-g+1}\right) + \kappa m_1}{p_{\max} + \kappa m_1}.$$

In the worst case, $g = 1$ and $p_g = p_{\max}$. Replacing κm_2 by zero as above, we obtain

$$\frac{\Phi_{np}\left(m^H\right)}{\Phi_{np}\left(m_{np}^*\right)} \leq \frac{p_{\max}\left(2 - \frac{2}{m_1}\right)}{p_{\max}} = 2 - \frac{2}{m_1}.$$

To see that the bound (11.25) is asymptotically tight, consider an instance with $m + 1$ jobs, that contains one job with the processing time m and m jobs with the processing time $m - 1$ each, where m is a positive integer. The value of the constant κ is less than 1. It follows that $m_1 = m$ and $\mu = \sqrt{m^2/\kappa} > m$, so that $m_2 = m_1 - 1 =$

$m-1$. Since $\Gamma_1(m_1) = m + \kappa m$ and $\Gamma_2(m_2) = \frac{m^2}{m-1} + \kappa\,(m-1)$, we deduce that $m_p^* = m_1 = m$ due to a small value of κ. Taking m as m^H we find the best non-preemptive schedule on m machines in which each machine processes one job, except one machine that processes two jobs of duration $m-1$ each, i.e., $C_{\max}(S_{np}^*(m)) = 2m-2$ and $\Phi_{np}(m) = 2m - 2 + \kappa m$. On the other hand, the total cost function reaches its minimum for $m_{np}^* = m+1$, so that in schedule $S_{np}^*(m+1)$ each machine processes exactly one job. Thus, $C_{\max}\left(S_{np}^*(m+1)\right) = m$ and $\Phi_{np}(m+1) = m + \kappa\,(m+1)$. Thus,

$$\frac{\Phi_{np}(m^H)}{\Phi_{np}(m_{np}^*)} = \frac{\Phi_{np}\,(m)}{\Phi_{np}\,(m+1)} = \frac{2m-2+\kappa m}{m+\kappa(m+1)}.$$

As κ approaches zero, the ratio goes to $2 - \frac{2}{m}$. This proves the theorem. □

The next statement establishes an improved performance of Algorithm 3, because for $m_1 > \lceil \mu \rceil$ the machine cost κm makes a larger contribution to the total cost function.

Theorem 11.9. *For $m^H = m_p^* \in \{m', m''\}$ found in Step 3 by Algorithm 1 the bound*

$$\frac{\Phi_{np}(m^H)}{\Phi_{np}\left(m_{np}^*\right)} \leq \frac{3}{2} - O\left(\frac{1}{m_p^*}\right) \tag{11.26}$$

holds, and this bound is tight.

Proof: We know that $\Phi_{np}(m_{np}^*) \geq \Phi_p(m_{np}^*) \geq \Phi_p(m_p^*)$, where either $m_p^* = m''$ or $m_p^* = m'$.

If $m_p^* = m''$, i.e., if $\Phi_p(m_p^*) = \Gamma_2\,(m'')$ we deduce from (11.23) with $m = m''$ that

$$\begin{aligned}\frac{\Phi_{np}\left(m^H\right)}{\Phi_{np}\left(m_{np}^*\right)} &\leq \frac{\Phi_{np}\,(m'')}{\Gamma_2\,(m'')} = \frac{C_{\max}\left(S_{np}^*(m'')\right) + \kappa m''}{\frac{P}{m''} + \kappa m''} \\ &\leq \frac{C_{\max}\left(S_p^*(m'')\right)\left(2 - \frac{2}{m''+1}\right) + \kappa m''}{\frac{P}{m''} + \kappa m''}.\end{aligned}$$

Notice that due to the choice of m'' the equality $C_{\max}\left(S_p^*(m'')\right) = \frac{P}{m''}$ holds, so that

$$\frac{\Phi_{np}\left(m^H\right)}{\Phi_{np}\left(m_{np}^*\right)} \leq \frac{\frac{P}{m''}\left(2 - \frac{2}{m''+1}\right) + \kappa m''}{\frac{P}{m''} + \kappa m''}.$$

The last derived expression is decreasing in $\kappa m''$. Since $m'' \geq \mu$, it follows from

(11.21) with $m = m''$ that

$$\begin{aligned}\frac{\Phi_{np}\left(m^H\right)}{\Phi_{np}\left(m^*_{np}\right)} &\leq \frac{\frac{P}{m''}\left(2-\frac{2}{m''+1}\right)+\frac{P}{m''}}{2\frac{P}{m''}} \\ &= \frac{3-\frac{2}{m''+1}}{2} = \frac{3}{2}-\frac{1}{m''+1} = \frac{3}{2}-O\left(\frac{1}{m^*_p}\right).\end{aligned}$$

Now, assume that $m^*_p = m'$, i.e., that $\Phi_p(m^*_p) = \Gamma_2(m')$. We deduce

$$\begin{aligned}\frac{\Phi_{np}\left(m^H\right)}{\Phi_{np}\left(m^*_{np}\right)} &\leq \frac{\Phi_{np}(m')}{\Gamma_2(m')} = \frac{\left(2-\frac{2}{m'+1}\right)\frac{P}{m'}+\kappa m'}{\frac{P}{m'}+\kappa m'} \\ &= \frac{\left(2-\frac{2}{m'+1}\right)\frac{P}{m'}+\kappa m'}{\frac{P}{m'}+\kappa m'}.\end{aligned}$$

Applying (11.21) with $m = m'' = m'+1$, we get

$$\frac{P}{m'+1} < \kappa\left(m'+1\right),$$

which implies

$$\frac{m'P}{\left(m'+1\right)^2} < \kappa m'.$$

Using this lower bound on the cost component, we obtain

$$\begin{aligned}\frac{\Phi_{np}\left(m^H\right)}{\Phi_{np}\left(m^*_{np}\right)} &\leq \frac{\left(2-\frac{2}{m'+1}\right)\frac{P}{m'}+\frac{m'P}{(m'+1)^2}}{\frac{P}{m'}+\frac{m'P}{(m'+1)^2}} \\ &= \frac{3}{2}-\frac{m'+\frac{3}{2}}{2\left(m'\right)^2+2m'+1} = \frac{3}{2}-O\left(\frac{1}{m^*_p}\right).\end{aligned}$$

To see that the bound (11.26) is tight, consider an instance with $m+1$ jobs, each with the processing time m, where m is a positive integer. In the objective function, set $\kappa = \frac{m+1}{m}$. It follows that $m_1 = m+1$ and $m_2 = \mu = m' = m'' = m$. Since $\Gamma_1(m_1) = m + \frac{(m+1)^2}{m} = 2m+2+\frac{1}{m}$ and $\Gamma_2(m_2) = 2(m+1)$, we deduce that $m^*_p = m_2 = m$. Taking m as m^H we find the best non-preemptive schedule on m machines in which each machine processes one job, except one machine that processes two jobs, i.e., $C_{\max}(S^*_{np}(m)) = 2m$ and $\Phi_{np}(m) = 2m + (m+1) = 3m+1$. On the other hand, the total cost function reaches its minimum for $m^*_{np} = m+1$, so that $C_{\max}\left(S^*_{np}(m+1)\right) = m$ and $\Phi_{np}(m+1) = m + \frac{(m+1)^2}{m}$. Thus,

$$\begin{aligned}\frac{\Phi_{np}(m^H)}{\Phi_{np}(m^*_{np})} &= \frac{\Phi_{np}(m)}{\Phi_{np}(m+1)} = \frac{3m+1}{m+\frac{(m+1)^2}{m}} \\ &= \frac{3m^2+m}{2m^2+2m+1} = \frac{3}{2} - \frac{m+\frac{3}{2}}{2m^2+2m+1} = \frac{3}{2} - O\left(\frac{1}{m}\right).\end{aligned}$$

This proves the theorem. □

We have conducted a series of computational experiments to evaluate the performance of Algorithm 3. For that purpose, we have generated 100 instances of the problem with $\kappa \in \{1, 5, 10\}$ and $n \in \{50, 100, 500\}$. For each combination of κ and n, we used six probability distributions to draw the processing times from:

Uni1, Uni2: uniform distribution over $[1, 40]$ and over $[1, 100]$, respectively;

Norm1, Norm2: normal distribution with a mean of 20 and a standard deviation of 5, a mean of 50 and a standard deviation of 10, respectively;

Exp1, Exp2: exponential distribution with a mean of 20 and with a mean of 50, respectively.

The use of different distributions is justified by the fact that theoretically the performance of the algorithm is affected by the optimal number of machines for the preemptive counterpart of the problem found in Step 1 of Algorithm 3, and that in turn depends on the presence of a long job in the instance.

Algorithm 3 has been programmed in MATLAB 7.12.0 on an Intel Core i7-2670QM CPU at 2.20 GHz and 6GB of memory. The CPU time required to run the whole set of experiments, i.e., a total of 5400 problems, is less than 12 sec.

For each combination of a probability distribution and the values of κ and n, the corresponding cell of Table 11.1 reports three pieces of information. In the top line, the string of the form $x_1/x_2/y_1/y_2$ shows that out of 100 instances, the case $m^*_p = m_1$ or $m^*_p = m_2$ has been observed x_1 and x_2 times, respectively, while the case $m^*_p = m'$ or $m^*_p = m''$ has been observed y_1 and y_2 times, respectively. In the bottom line, the string of the form z_1/z_2 shows the average value (z_1) and the maximum value (z_2) of the ratio $\Phi_{np}(m^H)/\Phi_p(m^*_p)$. Notice that in our computation we use $\Phi_p(m^*_p)$ as a lower bound on $\Phi_{np}(m^*_{np})$. For example, in the top left corner cell, we see that among 100 instances with 50 jobs and $\kappa = 1$ with the processing times drawn from distribution *Uni1*, m^*_p is never equal to m' or m'', while for 37 instances $m^*_p = m_1$ and for 63 instances $m^*_p = m_2$. Besides, the average value of $\Phi_{np}(m^H)/\Phi_p(m^*_p)$ is 1.1692, with a maximum of 1.2516.

Problem	Effect	Running Time
[Attach	separate	table here]

Table 11.1: Results of computational experiment for Algorithm 3

We think it is important to be aware how often a particular m_p^* has been observed. Recall that for $m_p^* \in \{m', m''\}$ Algorithm 3 has a better theoretically proved worst-case performance than for $m_p^* \in \{m_1, m_2\}$; see Theorems 11.9 and 11.8, respectively. Apart from the exponential distributions, a clear separation is observed, i.e., either $m_p^* \notin \{m_1, m_2\}$ or $m_p^* \notin \{m', m''\}$, with one exception in each column *Norm1* and *Norm2* for $\kappa = 1$.

For $m_p^* \notin \{m', m''\}$, the worst performance has been observed for *Norm2* distribution with $\kappa = 1$ and $n = 50$; the average and maximum values of $\Phi_{np}(m^H)/\Phi_p(m_p^*)$ are 1.2216 and 1.2885. Notice that these values are also the worst across all generated instances. For $m_p^* \notin \{m_1, m_2\}$ the worst performance has been observed for *Uni1* distribution with $\kappa = 1$ and $n = 100$; the average and maximum values of $\Phi_{np}(m^H)/\Phi_p(m_p^*)$ are 1.1385 and 1.1772. Thus, Algorithm 3 that calls Algorithm 2 with $m_p^* \in \{m', m''\}$ performs more accurately in practice, which complies with the theoretical estimates in Theorems 11.8 and 11.9.

If no separation occurs, i.e., for m_p^* each of the values m_1, m_2, m' and m'' is possible, the worst performance has been observed for *Norm2* distribution with $\kappa = 1$ and $n = 100$; the average and maximum values of $\Phi_{np}(m^H)/\Phi_p(m_p^*)$ are 1.1648 and 1.2026. Notice that in this case Algorithm 3 calls Algorithm 2 with $m_p^* \in \{m_1, m_2\}$ for 91 out of 100 instances.

We have used a design of experiments on software VISUALDOC 6.0 to determine that for all studied probability distributions for both performance measures (average and maximum ratio) κ is the most influential parameter. Both measures decrease as either κ or n increase; this complies well with the fact the higher values of both performance measures are observed in the top row of Table 11.1 ($\kappa = 1$, $n = 50$).

Finally notice that, for each combination of the design parameters, Algorithm 3 in practice delivers ratios less than theoretically established worst-case ratios.

11.5 Cost-Effective Choice of The Number of Machines: Total Flow Time

In this section, we consider the problem of finding the optimal number of machines that minimises the total cost function (11.17), provided that the scheduling objective $F = \sum C_j$.

Recall that for problem $Pm \,||\, \sum C_j$, the optimal total flow time is defined by formula (2.15), provided that the jobs are numbered in accordance with the LPT rule (2.7).

Define two sequences:

$$\begin{aligned} G(m) &= \sum_{j=1}^{n} p_j \left\lceil \frac{j}{m} \right\rceil, \ m = 1, 2, \cdots, n \\ T(m) &= \kappa m, \ m = 1, 2, \cdots, n \end{aligned}$$

where $G(m)$ is the total flow time of processing the jobs on m machines and $T(m)$ defines the normalised total cost of using m machines. Thus, for the problem under consideration, the objective function can be written as

$$\Phi(m) = G(m) + T(m).$$

Finding the value of m^* that minimises function $\Phi(m)$ can easily be done by direct enumeration: try all values of m from 1 to n, compute the components of the function and select the best value of m. Such an approach requires $O(n \log n)$ time for finding an LPT numbering of the jobs, and then $O(n)$ time for computing function Φ for each trial value of m, $1 \leq m \leq n$. Thus, such a brute-force algorithm takes $O(n^2)$ time.

Below we present an $O(n \log n)$ algorithm for finding m^* that uses only $\lceil \log_2 n \rceil$ trial values of m, due to our observation that the sequence $\Phi(m)$, $1 \leq m \leq n$, is actually V-shaped with respect to m, $1 \leq m \leq n$. Recall from Chapter 5, that a sequence $A(m)$ is called *V-shaped* if there exists an m_0, $1 \leq m_0 \leq n$, such that

$$A(1) \geq \cdots \geq A(m_0 - 1) \geq A(m_0) \leq A(m_0 + 1) \leq \cdots \leq A(n).$$

First, we show that sequence $G(m)$, $m = 1, 2, \cdots, n$, is convex. Recall that a sequence $A(m)$, $1 \leq m \leq n$, is called *convex* if

$$A(m) \leq \frac{1}{2} \left(A(m-1) + A(m+1) \right), \ 2 \leq m \leq n-1.$$

Our reasoning is based on Theorem 5.2 proved in Chapter 5, which states that a sequence $P(k) = \sum_{j=1}^{n} p_j g\left(\left\lceil \frac{j}{k} \right\rceil\right)$, $1 \leq k \leq n$, is convex if $p_1 \geq p_2 \geq \cdots \geq p_n$. For our purposes we need Theorem 5.2 in a weaker form, with $g \equiv 1$. Thus, the following statement holds.

Lemma 11.6. *Let $S^*(m)$ be a schedule that minimises the total flow time on m identical parallel machines, so that $G(m) = \sum_{j \in N} C_j\left(S^*(m)\right)$. Then the sequence of values $G(m)$, $1 \leq m \leq n$, is convex.*

Obviously, the sequence $T(m)$ is also convex since $T(m) = \frac{1}{2}\left(T(m-1) + T(m+\right.$

$1)) = \kappa m$ for each m, $2 \leq m \leq n-1$.

It is easy to verify that the sum of two convex sequences is convex and any convex sequence is V-shaped. Thus, Lemma 11.6 immediately implies

Theorem 11.10. *For the problem of minimising the total cost, the sequence* $\Phi(m)$, $1 \leq m \leq n$, *of values of the objective function is convex and* V*-shaped.*

Theorem 11.10 allows us to find an optimal schedule with an optimal number of machines m^* by performing binary search with respect to m. As a result at most $\lceil \log_2 n \rceil$ values of m need to be tested, so that the running time of this method becomes $O(n \log n)$.

11.6 Interpretations and Practical Implications

We conclude this chapter with a brief discussion of the issues related to the machine impact that may be of interest to managerial decision making.

The upper/lower bound on the machine impact is important for two reasons: (i) it gives the managers an estimate of what they can expect to gain in the best/worst case if extra machines are added, and (ii) what they can expect to lose in the worst/best case if a machine is not included. Below we give illustrations of how the machine impact can be used by managers in real life.

A lower bound on the machine impact guarantees an improvement of the performance of the schedule by a certain factor, if a known number of extra machines are used. Our analysis, however, shows that the only situation which results in a non-trivial lower bound is the problem $Pm|p_j = 1|\sum C_j$; see Theorem 11.4. For the other studied problems, a lower bound of 1 is returned. On the other hand, an upper bound provides non-trivial results and can be used more widely.

Consider a computing system that consists of two identical parallel units that daily performs more or less the same package of tasks with no preemption. The manager is interested in determining the number of parallel units to be additionally installed as part of the system to reduce the time of processing the daily package by 60%, i.e., to achieve $C_{\max}(S^*(\widehat{m})) \leq 0.4 C_{\max}(S^*(2))$ or $C_{\max}(S^*(2))/C_{\max}(S^*(\widehat{m})) \geq 2.5$. Then it follows from Theorem 11.1 that to achieve $\lceil \widehat{m}/2 \rceil \geq 2.5$ we need $\widehat{m} \geq 5$, i.e., at least three extra units should be used.

An illustrative example of a similar nature can be given regarding a manufacturing shop that uses a park of identical machine-tools for performing a particular techno-

logical operation, e.g., drilling. Suppose that originally three drilling machines are used to process an established collection of orders on a regular basis. The shop floor management wants to extend the park of machines with a purpose of reducing the average time for handling an order by 30%, i.e., to achieve $G(S^*(\widehat{m})) \leq 0.7G(S^*(3))$ or $G(S^*(3))/G(S^*(\widehat{m})) \geq 10/7$. Then it follows from Theorem 11.5 that to achieve $\widehat{m}/3 \geq 10/7$ we need $\widehat{m} \geq 5$, i.e., at least two extra machines should be installed. Given the cost of the new machines, the management may decide whether they want this change to be implemented. If the cost is their priority, the method of Section 11.5 can be used to find the most cost-effective decision.

Below we present an interpretation of the results obtained for the problem of minimising total flow time in terms of the problem of safe pickup of employees from offshore installations. The latter problem arises in the oil and gas sector, and its study has been initiated by Qian et al. (2011). In the most general settings, the problem can be formulated as follows. Given a set $N = \{1, 2, \ldots, n\}$ of installations, the pickup demands $p_j, j \in N$, i.e., the number of people to be taken from installation j, and the helicopter capacity Q, find a capacity-feasible flight schedule S that satisfies total pickup demand and minimises the *total risk,* measured as the total number of people exposed to take-offs and landings.

For the purpose of this illustration, we ignore the capacity constraints, and assume that the helicopter is large enough to take all people on board. This problem is closely linked with problem $Pm \,||\, \sum C_j$, in which the jobs are interpreted as installations, the processing times as pickup demands, the machines as flights of the helicopter, and minimising the total flow time is equivalent to minimising the total risk. Additionally, the *non-split* pickup scenario is assumed, under which no installation is visited more than once and the helicopter picks up all p_j passengers from an installation j.

Thus, $G(S^*(m))$ defined by (11.8) measures the total risk of an optimal flight schedule $S^*(m)$ that is comprised of m flights. Clearly, if the number of flights is decreased, the risk increases, i.e., $G(S^*(m+1)) > G(S^*(m))$. Thus, Theorem 11.5 implies that if the number of flights decreases from $m+1$ to m, then the total risk $G(S^*(m))$ of the resulting flight schedule can be up to $\frac{m+1}{m}$ times larger than that for the best schedule with $m+1$ flights. This is especially sensitive if m is small, which is typical in reality. Indeed, reducing the number of flights from 3 to 2 may increase the total risk up to 150%.

Theorem 11.10 leads to an $O(n \log n)$-time algorithm that determines the optimal number of flights that minimises the cost function $\Phi(m) = G(m) + \kappa m$, where κ represents the normalised cost of one flight.

11.7 Conclusion

In this chapter, we study the influence of adding extra machines for the classical scheduling problems on identical parallel machines, to minimise the makespan and to minimise the total flow time. We present tight bounds on the machine impact and give algorithms for making a cost-effective choice of the number of the machines. These results may have applications to various areas, including computing and manufacturing.

CHAPTER 12

Enhanced Models with Changing Processing Times

In this chapter, we extend our study on single machine scheduling with changing processing times to a parallel machine environment. We consider several parallel machine models that have not been studied previously. We start with a classical model in which the parallel machines are subject to capacity constraints with fixed processing times. We then extend this model, and study parallel machines which are subject to the combined effects as introduced in Chapter 9. We provide polynomial-time algorithms for the problem of minimising the total flow time for all the models considered. Most of the algorithmic ideas used to solve the problems are directly transferred from their single machine counterparts, studied in previous chapters.

The results of this chapter are published in our recent papers Rustogi and Strusevich (2012b) and Rustogi and Strusevich (2013b). Unlike the content provided in these papers, in this chapter, we provide a detailed account of scheduling problems with parallel machines and changing processing times.

12.1 Capacitated Parallel Machines

The jobs of a set $N = \{1, 2, \ldots, n\}$ have to be processed on m parallel machines $M_1, M_2, \ldots, M_m$, where $m \leq n$. To start with, we consider the case of uniform machines. We assume that the machines are renumbered and their speeds $s_i, 1 \leq i \leq m$, are rescaled in such a way that $s_1 \geq s_2 \geq \cdots \geq s_m = 1$. The objective is to minimise the sum of the completion times $\sum C_j$. There are restrictions regarding the number of jobs to be assigned to a machine, so that in any feasible schedule machine M_i processes no more than $q^{[i]}$ jobs, where $\sum_{i=1}^{m} q^{[i]} > n$. The processing times of jobs are assumed

to fixed at all times, i.e., effects such as deterioration or learning, are not involved. We denote the problem under consideration by $Qm \left| \sum_{i=1}^{m} q^{[i]} > n \right| \sum C_j$. The case that $\sum_{i=1}^{m} q^{[i]} = n$ is considered separately, in Section 12.2, since under that equality a more general problem can be solved, in which the jobs are subject to positional effects and rate-modifying activities.

Notice that if no capacity constraints are imposed, problem $Qm \,||\, \sum C_j$ is solvable in $O(n \log n)$ time due to Algorithm QmSum as outlined in Section 2.2.5. However, the status of problem $Qm \left| \sum_{i=1}^{m} q^{[i]} > n \right| \sum C_j$ to our best knowledge has remained open, even if $m = 2$ and the machines are identical. The closest problem studied earlier is $Pm \left| \sum_{i=1}^{m} q^{[i]} \geq n \right| \sum w_j C_j$ to minimise the sum of weighted completion times on capacitated identical machines. For this NP-hard problem, Woeginger (2005) gives a fully polynomial-time approximation scheme and a pseudo-polynomial dynamic programming algorithm for its solution. It should be noted that even for $m = 2$ and equal weights, the algorithm by Woeginger (2005) still requires pseudo-polynomial time.

Given a feasible schedule for problem $Qm \left| \sum_{i=1}^{m} q^{[i]} > n \right| \sum C_j$, the number of jobs assigned to machine M_i is denoted by $h^{[i]}$, where $h^{[i]} \leq q^{[i]}$,$1 \leq i \leq m$, and $\sum_{i=1}^{m} h^{[i]} = n$. On machine M_i the assigned jobs are processed in an order $\pi^{[i]}(1), \ldots, \pi^{[i]}(h^{[i]})$. The contribution of machine M_i towards the objective function is equal to

$$\begin{aligned} & \frac{p_{\pi^{[i]}(1)}}{s_i} + \left(\frac{p_{\pi^{[i]}(1)}}{s_i} + \frac{p_{\pi^{[i]}(2)}}{s_i} \right) + \cdots + \left(\frac{p_{\pi^{[i]}(1)}}{s_i} + \frac{p_{\pi^{[i]}(2)}}{s_i} + \cdots + \frac{p_{\pi^{[i]}(h^{[i]})}}{s_i} \right) \\ = & \sum_{r=1}^{h^{[i]}} \frac{\left(h^{[i]} - r + 1 \right) p_{\pi^{[i]}(r)}}{s_i}, \end{aligned}$$

and the overall objective function, the sum of completion times is written as (2.16).

Notice that the expression (2.16) is in the form of our generic objective function (4.6) as defined in Chapter 4. Here, the groups are machines, so that $k = m$, and we use i, rather than x, as an index variable, and $h^{[i]}$ rather than $n^{[x]}$ to denote the number of jobs in a group. The constant term $\Gamma(m)$ is zero, and the group-dependent weights are defined by $W^{[i]}(r) = \left(h^{[i]} - r + 1 \right) / s_i$, $1 \leq i \leq m$, where the values of $h^{[i]}$ are to be found out along with the optimal sequence of jobs.

To solve problem $Qm \left| \sum_{i=1}^{m} q^{[i]} > n \right| \sum C_j$, assume $h^{[i]} = q^{[i]}$, $1 \leq i \leq m$, and compute a set of all possible positional weights $W^{[i]}(r), 1 \leq r \leq q^{[i]}$, $1 \leq i \leq m$, which

can be organised as a matrix

$$\begin{pmatrix} q^{[1]}/s_1 & q^{[2]}/s_2 & \cdots & q^{[m]}/s_m \\ \vdots & \vdots & \cdots & \vdots \\ 2/s_1 & 2/s_2 & \vdots & 2/s_m \\ 1/s_1 & 1/s_2 & \cdots & 1/s_m \end{pmatrix}. \tag{12.1}$$

Each column in the above matrix represents all possible positional weights that can be associated with a particular machine, the first element of column i representing a weight associated with the first position of machine i, while the last element of column i, representing a weight associated with the last, i.e., the $q^{[i]}$- th position of machine i, $1 \leq i \leq m$. Notice that for each machine the smallest weights are associated with the last position on that machine.

The structure of the above matrix (12.1) is similar to the matrix (8.5) created for solving problem $1\left|p_j + a^{[x]}\tau, MP\right| C_{\max}$ in Section 8.3. The only significant difference is that each column of matrix (12.1) is truncated at the top because of the capacity constraint. It is possible to represent matrix (12.1) exactly as matrix (8.5) by assigning the value ∞, for the positional weights associated with the infeasible positions. For example, for problem $Q3\left|\sum_{i=1}^{m} q^{[i]} > n\right|\sum C_j$ with $q^{[1]} = 3$, $q^{[2]} = 2$ and $q^{[3]} = 4$, the resulting matrix will be as follows

$$\begin{pmatrix} \infty & \infty & 4/s_3 \\ 3/s_1 & \infty & 3/s_3 \\ 2/s_1 & 2/s_2 & 2/s_3 \\ 1/s_1 & 1/s_2 & 1/s_3 \end{pmatrix}.$$

For the purpose of minimising $\sum C_j$ we need to select the n smallest weights from the resulting matrix, taking consecutive weights from each column, and matching these weights to the jobs with the largest processing times. Theorem 6.1 proves the optimality of this approach. This can be done in $O(n \log n)$ time, exactly as described in Algorithm NSmallRev or Algorithm QSum, with the positional weights defined appropriately in Steps 1 and 2b.

Next, we consider the problem of minimising the total flow time on capacitated unrelated parallel machine. Recall that for unrelated parallel machines, the normal processing time of a job j if scheduled on a machine M_i is given by p_{ij}. If no machine capacity constraints are imposed, the resulting problem $Rm\,||\sum C_j$ is solvable in $O(n^3 m)$ time by reducing it to a rectangular assignment problem. In Section 2.2.5,

we provide details of such a rectangular assignment problem and show that it can be formulated as (2.17). If machine M_i cannot process more than $q^{[i]}$ jobs, the corresponding problem $Rm\left|\sum_{i=1}^{m} q^{[i]} > n\right|\sum C_j$ can still be represented as a rectangular LAP, by simply deleting those columns in (2.17) that correspond to infeasible positions.

The resulting LAP will have n rows and $\sum_{i=1}^{m} q^{[i]}$ columns and can be solved by using Algorithm BourLas (see Section 2.2.3) in $O\left(n^2 \sum_{i=1}^{m} q^{[i]}\right)$ time, which is at most $O\left(n^3 m\right)$. In an optimal solution for each machine M_i the equality $y_{j',(i,u')} = 1$ implies that $y_{j'',(i,u'')} = 1$ for all $u'', 1 \leq u'' < u'$, while the equality $y_{j',(i,u')} = 0$ implies that $y_{j'',(i,u'')} = 0$ for all $u'', u' < u'' \leq q^{[i]}$. In other words, if we view the sequence of the y-values associated with machine M_i going from the last position towards the first, then such a sequence starts with a subsequence of consecutive ones (except a situation in which a machine is not assigned any jobs at all) that might be followed by a subsequence of several consecutive zeros.

Thus, for all types of parallel machines, solving the capacitated version of the problem of minimising the total flow time requires the same time as needed for solving its uncapacitated counterpart. For the unrelated machines, we need a full form of the assignment problem, while for identical and uniform machines the matching procedure is applicable.

12.2 Problems with Changing Processing Times

In this section, we extend our results from Chapter 9 to parallel machines which are subject to changing processing times and rate-modifying activities.

Formally, the problem considered in this section is denoted by $\alpha\left|Combi, RMP\right|\sum C_j$, where $\alpha \in \{Pm, Qm.Rm\}$. The jobs of a set $N = \{1, 2, \ldots, n\}$ have to be processed on m parallel machines $M_1, M_2, \ldots, M_m$, where $m \leq n$. Each of the machines are subject to combined effect of the form (9.3). Additionally, the decision-maker is presented with a total of $K \geq 0$ possible rate-modifying activities, which can be either distinct or alike. For each RMP, it is exactly known how it affects the processing conditions of the machine, should the decision-maker decide to include it into a schedule. Further, the duration of each RMP is given by the general formula (9.4). In order to obtain an optimal solution to problem $\alpha\left|Combi, RMP\right|\sum C_j$, a decision-maker must make the following decisions:

Decision 1. The number, choice and sequence of RMPs on each machine.

Decision 2. The number of jobs to include in each of the created groups across all

machines.

Decision 3. For each job $j \in N$, determine the group and the position within that group, that it must be scheduled in.

Let us first concentrate on Decisions 1-2. We make these decisions by enumerating all options associated with them. Assume that a total of $k-1$ RMPs are chosen from the available K, so that $\sum_{i=1}^{m}(k_i-1)=k-1$. This means that the total number of groups created across all machines is equal to $m+k-1$. For a known k, $1 \leq k \leq K+1$, selecting $k-1$ of K available RMPs and taking all permutations over $k-1$ RMPs can be done in $\binom{K}{k-1}(k-1)!$ ways. Distributing $m+k-1$ groups over m machines can be done by generating all compositions of $m+k-1$ into exactly m positive summands; the number of these options is $\binom{m+k-2}{m-1}=\frac{(m+k-2)!}{(m-1)!(k-1)!}$. Thus, for a given k, the total number of options for Decision 1 is equal to $\binom{K}{k-1}\frac{(m+k-2)!}{(m-1)!}$. Next, generating all possible values for the number of jobs to include in each of the created groups, requires enumeration of integer compositions of n with $m+k-1$ parts; the number of these options is $\binom{n-1}{m+k-2}$ which can be approximated by $\frac{n^{m+k-2}}{(m+k-2)!}$. Thus, for a given k, the total number of options are equal to $\binom{K}{k-1}\frac{(k+m-2)!}{(m-1)!}\frac{n^{m+k-2}}{(m+k-2)!}=\binom{K}{k-1}\frac{n^{m+k-2}}{(m-1)!}$. Trying all possible values of k, $1 \leq k \leq K+1$, the total number of options to be enumerated for Decisions 1-2 can be estimated as $\sum_{k=1}^{K+1}\binom{K}{k-1}\frac{n^{m+k-2}}{(m-1)!}=O\left(n^{m+K-1}\right)$.

If Decisions 1-2 are taken in advance, denote the resulting problem as $\alpha\,|Combi, RMP\,(k-1)|\sum C_j$.

Let us now consider an instance of such a problem. For each machine M_i, $1 \leq i \leq m$, assume the following. A total of $q^{[i]}$ jobs are assigned to machine M_i, so that $\sum_{i=1}^{m} q^{[i]}=n$. Out of the available K RMPs, k_i-1 of them are chosen and scheduled on machine M_i in a given order, so that $q^{[i]}$ jobs are divided into k_i groups. Each group contains a total of $n^{[i,x]}$ jobs, where $\sum_{x=1}^{k_i} n^{[i,x]}=q^{[i]}$. The permutation of jobs on machine M_i is given by $\pi^{[i]}=\left(\pi^{[i,1]},\pi^{[i,2]},\ldots,\pi^{[i,k_i]}\right)$, where $\pi^{[i,x]}=\left(\pi^{[i,x]}(1),\pi^{[i,x]}(2),\ldots,\pi^{[i,x]}\left(n^{[i,x]}\right)\right)$, $1 \leq x \leq k_i$. The normal processing time of a job $j=\pi^{[i,x]}(r)$, is given by $p_{i,\pi^{[i,x]}(r)}$. The actual processing time of a job $j=\pi^{[i,x]}(r)$, scheduled in position r, $1 \leq r \leq n^{[i,x]}$, of the x-th group, $1 \leq x \leq k_i$, of machine M_i is given by the combined effect (9.3). The duration of the RMP scheduled after the x-th group of machine M_i is given by (9.4).

Notice that a group x, $1 \leq x \leq k_i$, on a machine M_i is denoted by the index $[i,x]$. Using such a double index notation allows us to distinguish between groups on different machines. It also allows us to consider machine-dependent effects, so that each machine is allowed to behave (time taken to process jobs, deterioration/learning rates, response to RMPs, etc.) in a unique way.

Under the conditions defined above, the total flow time on machine M_i can be computed similarly to the total flow time obtained for an instance of the single machine problem $1\left|Combi, RMP\left(k-1\right)\right|\sum C_j$, considered in Section 9.3.2. It follows that the total flow time for machine M_i is given by

$$\sum C_j\left(M_i\right)=\sum_{x=1}^{k_i}\sum_{r=1}^{n^{[i,x]}}W^{[i,x]}\left(r\right)p_{i,\pi^{[i,x]}(r)}+\Gamma\left(k_i\right),$$

where for a fixed i, $1\leq i\leq m$, the positional factors $W^{[i,x]}\left(r\right)$, $1\leq r\leq n^{[i,x]}$, $1\leq x\leq k_i$, are written similarly to (9.17) and the constant term $\Gamma\left(k_i\right)$ is similarly to (9.18). Thus, the total flow time for m machines can be written as

$$\sum C_j=\sum_{i=1}^{m}\sum_{x=1}^{k_i}\sum_{r=1}^{n^{[i,x]}}W^{[i,x]}\left(r\right)p_{i,\pi^{[i,x]}(r)}+\sum_{i=1}^{m}\Gamma\left(k_i\right). \tag{12.2}$$

For problem $Qm\left|Combi, RMP\left(k-1\right)\right|\sum C_j$, the objective function (12.2) can be seen as a linear assignment problem with a product matrix (2.3), with one array consisting of the values $W^{[i,x]}\left(r\right)/s_i$, and another array consisting of the values $p_{\pi^{[i,x]}(r)}$, for all $1\leq r\leq n^{[i,x]}$, $1\leq x\leq k_i$, $1\leq i\leq m$. The latter problem can be solved in $O\left(n\log n\right)$ time by Algorithm Match; see Section 2.2.3. Recall from Section 9.3.2, that the running time needed to compute all positional weights $W^{[x]}\left(r\right)$, $1\leq r\leq n^{[x]}$, $1\leq x\leq k$, for an instance of problem $1\left|Combi, RMP(k-1)\right|\sum C_j$ is equal to $T\left(W\right)=O\left(n^2\right)$. The same running time will be required to compute all values $W^{[i,x]}\left(r\right)/s_i$. Thus, all together, problem $Qm\left|Combi, RMP\left(k-1\right)\right|\sum C_j$ can be solved in $O\left(n^2\right)$ time, and problem $Qm\left|Combi, RMP\right|\sum C_j$ can be solved in $O\left(n^{m+K+1}\right)$ time.

Notice that some reduced versions of problem $Qm\left|Combi, RMP\left(k-1\right)\right|\sum C_j$ can be solved faster. Recall from Section 9.5, that for an instance of problems $1\left|Posi, RMP\left(k-1\right)\right|\sum C_j$ and $1\left|Time, RMP\left(k-1\right)\right|\sum C_j$, all positional weights can be computed in $T\left(W\right)=O\left(n\right)$ time each. It follows that the problem $Qm\left|Posi, RMP\left(k-1\right)\right|\sum C_j$, with a pure positional effect of the form (6.1), and problem $Qm\left|Time, RMP\left(k-1\right)\right|\sum C_j$, with a pure time-dependent effect of the form (9.21) can be solved in $O\left(n\log n\right)$ time each. Their full versions can be solved in $O\left(n^{m+K}\log n\right)$ time each.

Next, for problem $Rm\left|Combi, RMP\left(k-1\right)\right|\sum C_j$, the objective function (12.2) can be seen as a linear assignment problem in the full form (2.2), with the cost function given as $c_{j,(i,x,r)}=p_{ij}W^{[i,x]}\left(r\right)$, for all $1\leq r\leq n^{[i,x]}$, $1\leq x\leq k_i$, $1\leq i\leq m$. Here the rows of the cost matrix are associated with the jobs, while the columns with the

triples (i, x, r), i.e., (machine, group on the machine, position in the group). The latter problem can be solved in $O(n^3)$ time by the Hungarian Algorithm; see Section 2.2.3. Thus, all together, problem $Rm\,|Combi, RMP\,(k-1)|\sum C_j$ can be solved in $O(n^3)$ time and problem $Rm\,|Combi, RMP|\sum C_j$ can be solved in $O\left(n^{m+K+2}\right)$ time. Notice that problem $Rm\,|Posi\text{-JD}, RMP|\sum C_j$, with a pure job-dependent positional effect (7.1) can also be solved in $O\left(n^{m+K+2}\right)$ time by means of a full form LAP; see Section 9.5 for the single machine version of this problem.

The main results of this section are summarised in Table 12.1.

Problem	Effect	Running Time
$Qm\,\|Combi, RMP\|\sum C_j$	(9.3)-(9.4)	$O\left(n^{m+K+1}\right)$
$Qm\,\|Posi, RMP\|\sum C_j$	(6.1)-(9.4)	$O\left(n^{m+K}\log n\right)$
$Qm\,\|Time, RMP\|\sum C_j$	(9.21)-(9.4)	$O\left(n^{m+K}\log n\right)$
$Rm\,\|Combi, RMP\|\sum C_j$	(9.3)-(9.4)	$O\left(n^{m+K+2}\right)$
$Rm\,\|Posi\text{-JD}, RMP\|\sum C_j$	(7.1)-(9.4)	$O\left(n^{m+K+2}\right)$
$Rm\,\|Time, RMP\|\sum C_j$	(9.21)-(9.4)	$O\left(n^{m+K+2}\right)$

Table 12.1: Computational complexities of different versions of problem $\alpha\,|Combi, RMP|\sum C_j$.

Notice that together with all the different versions of this problem that we consider in section, this study encompasses almost all existing results on parallel machine scheduling with changing processing times. For example, a special case of problem $Rm\,|Posi\text{-JD}, RMP|\sum C_j$ is considered in the recent paper by Wang, Wang and Liu (2011). They consider models without deterioration, or group dependence and perform only one RMP per machine. Setting $K = m$, in our model leads us to their results. However, similar to a flaw noticed in earlier papers (see, e.g., Mosheiov (2001b)), they have overestimated the number of LAPs to be solved and hence report a running time of $O(n^{2m+3})$, as against $O(n^{2m+2})$ that we obtain. Even the problems $Pm\,|p_j r^a, a<0|\sum C_j$ and $Qm\,|p_j r^{a_j}, a<0|\sum C_j$ considered by Mosheiov (2001b) and Mosheiov and Sidney (2003), respectively, can be seen as special cases of our problems $Qm\,|Posi, RMP|\sum C_j$ and $Rm\,|Posi\text{-JD}, RMP|\sum C_j$, for $K = 0$. Another special case of problem $Rm\,|Posi\text{-JD}, RMP|\sum C_j$ is considered by Ji and Cheng (2010), who obtain the same running time as us for a much less general model; see Section 9.5 for a review of their model for a single machine environment.

12.3 Conclusion

We broadly consider two different problems in this chapter. The first, related to scheduling on parallel machines with capacity constraints, fills an existing gap in scheduling research on parallel machines. We show that the problem of minimising the total flow time, on capacitated uniform and unrelated machines can be solved by modifying classical scheduling algorithms.

The next problem, related to scheduling on parallel machines with changing processing times and rate-modifying activities, further generalises the class of problems included in this thesis. Appropriate values of $m \geq 1$ and $K \geq 0$ can be used to obtain single machine models or models with or without rate-modifying activities.

CHAPTER 13

Summary and Conclusion

The prime focus of this research is to study the effect of rate-modifying activities on scheduling models with changing processing times. The main idea has been to extend and generalise this area, from both the modelling and the algorithmic prospective, to identify the suitable algorithmic tools for solving the relevant problems, and to enhance the existing models so that they would still remain solvable by the available tool-kit. Below we summarise the most significant outcomes of this thesis and propose some possibilities for future work.

13.1 Main Contributions

13.1.1 Modelling Contributions

Rate-Modifying Activities

- We introduce the idea that we can include rate-modifying activities of a different nature into a schedule. These activities can have different effects on the machine conditions and can have different duration parameters as well. We give the decision-maker an option to choose to which RMPs they want to include and in which order.
- To handle such RMPs, we introduce the concept of group-dependent processing conditions, in which each group of a schedule can have a different effect on the jobs.
- These developments give birth to a very wide range of practically relevant models for scheduling problems with changing processing times.

Positional and Time-Dependent Effects

- To model positional effects, we systematically use a general function $g(r)$, instead of using specific functions like polynomial or exponential. We show that the resulting problems are no harder to solve than the problems with specific functions.
- For position-dependent scheduling models without rate-modifying activities, we show that there is no need to assume a monotone order in the sequence $g(r), 1 \leq r \leq n$. The resulting problem is no harder to solve than its monotone counterpart. This allows us to consider models in which a deterioration effect is combined with a learning effect, or even models in which an arbitrary positional effect is considered.
- For position-dependent scheduling models with rate-modifying activities, we introduce group-dependent positional factors of the form $g^{[x]}(r)$ or $g_j^{[x]}(r)$.
- For time-dependent scheduling models with rate-modifying activities, we introduce a group-dependent model, which allows the duration of previous groups to affect the processing time of the current job.
- We study models in which group-dependent positional effects are combined with group-dependent time-dependent effects. This allows us to come up a very powerful model which is capable of handling almost every known job-independent effect. Problems of minimising the makespan and the total flow time under such a model can be solved in polynomial time.

Cumulative Effects

- We study the effect of a single rate-modifying activity in a model with cumulative deterioration effects. This is the first model of its kind for scheduling with cumulative effects. The resulting problem is found to be NP-hard.

13.1.2 Analytic Contributions

- We prove that the sequence $P(k) = \sum_{j=1}^{n} p_j g(\lceil j/k \rceil)$, $1 \leq k \leq n$, is convex provided that $p_1 \geq p_2 \geq \cdots \geq p_n$; see Chapter 5. This result enables us to solve various problems by exploring only logarithmic number of options that would be linear in the non-convex. The result has potential applications to other areas of Operational Research, including inventory control and queueing.

- Our study of the extra machine impact factor for identical parallel machines formally does not belong to scheduling with changing time, but it shares the same ideological point of combining scheduling and logistics decisions with a purpose of improving the overall performance of the processing system. Besides, the use of the convex sequences has appeared to be useful for that study, especially for the model with a total flow time objective.

13.1.3 Algorithmic Contributions

- Taking a general modelling framework as a basis, we have developed a common framework for designing the algorithms that use similar general principles for handling various types of the generalised models.

- In the prior research, there has been a general understanding of the role of the linear assignment problem as the main tool for solving problems with positional effects, especially if these effects are job-dependent. For job-independent effect, the search for an appropriate solution approach has been limited to simple priority rules (SPT, LPT and their versions). In this work, we show that for problems with a job-independent effect, the main tool is the simplified linear assignment problem with a product matrix. For some cases, solving the resulting problems is equivalent to a sorting the jobs by a priority rules. However, we have found that on multiple occasions, missed by our predecessors, the problems, even for the enhanced models, can be still solved in $O(n \log n)$ by a matching algorithm.

- For scheduling with rate-modifying activities, one of the important decisions to be made is to determine the optimal number of jobs to schedule in each group. Our predecessors used a somewhat weak, group balance principle to answer this question. This method is limited in its application and only works for very simple models. In this thesis, we develop Algorithm NSmall, which searches for the n smallest values from a set of all possible positional weights, and assigns the jobs in the corresponding positions. This approach is fairly robust and allows us to study enhanced models with group-dependent effects and start-time dependent MP durations.

- We develop Algorithm NSmall2, to solve problems that satisfy certain specific conditions, defined by *K-domi*. We prove that under these conditions, it is possible to search for the n smallest positional weights by comparing only $2n$ other values. This enables us to achieve a faster running time than Algorithm NSmall.

- We develop Algorithm BestLAP, to solve job-dependent problems that satisfy *K-domi*. We borrow the principle idea behind the speed up of Algorithm NSmall2 and modify the classical algorithm by Bourgeois and Lassale (1971), so that it can permit faster running times for our problem. We prove that the number of columns to use in a rectangular assignment problem can be significantly reduced, if the problem under consideration satisfies *K-domi*. This is non-trivial result, and potentially has application in other areas as well.

- To solve problems $1\left|Cumu, MP\left[0\right]\right|C_{\max}$ and $1\left|Cumu, MP\left[\alpha\right]\right|C_{\max}$ we use existing FPTASs available for the Subset-sum problem and the Half-product problem, respectively. However, due to the presence of additive constants in our objective functions, a straightforward application is not permitted. We derive essential conditions for which these algorithms deliver an ε- approximate solution for our objective functions.

13.2 Future Work

We feel that we have nearly exhausted the possibilities for much future work on problems with job-independent effects and rate-modifying activities, that permit polynomial-time algorithms. The models and algorithms we develop for such effects have been stretched to their limits. On the other hand, there are other closely related problems, which are still open and invite further research. Below we list out few of the possible directions for future research:

- To study the problem of minimising the makespan on parallel machine models with changing processing times and rate-modifying activities. This problem is NP-hard and it will be interesting to see if an FPTAS can be developed.

- To extend the results of Chapters 8 and 9 for job-dependent effects. We feel that a solution might be possible by a full form LAP.

- To prove the V- shapeness of the sequence $C_{\max}\left(S^{*}\left(k\right)\right), 1 \leq k \leq K+1$, for problem $1\left|p_j g_j\left(r\right)\text{-det}, MP\left[0\right]\right|C_{\max}$. If the sequence $C_{\max}\left(S^{*}\left(k\right)\right),\ 1 \leq k \leq K+1$, is proved to be convex, we will be able to solve the problem $1\left|p_j g_j\left(r\right)\text{-det}, MP\left[0\right]\right|C_{\max}$ in $O\left(n^3 \log K\right)$ time, instead of the current running time of $O\left(n^3 K\right)$.

- To study the problems $1\left|Cumu, MP\left[0\right]\right|C_{\max}$ and $1\left|Cumu, MP\left[\alpha\right]\right|C_{\max}$ in the presence of multiple rate-modifying activities. Our preliminary work in this di-

rection has suggested that in the case of multiple RMPs, it become extremely hard to control the behaviour of the additive constants in the objective functions.

REFERENCES

Alidaee, B., & Rosa, D. (1995). A note on the V-shaped property in one-machine scheduling. *Journal of the Operational Research Society, 46*, 128–132.

Alturki, U.M., Mittenthal, J., & Raghavachari, M. (1996). A dominant subset of V-shaped sequences for a class of single machine sequencing problems. *European Journal of Operational Research, 88*, 345–347.

Azar, Y., Epstein, L., & van Stee, R. (2000). Resource augmentation in load balancing. In: M. M. Halldorsson (Ed.), *Proceedings of SWAT '00, Lecture Notes in Computer Science, 1851,* 375–382.

Bachman, A., & Janiak, A. (2004). Scheduling jobs with position-dependent processing times. *Journal of the Operational Research Society, 55*, 257–264.

Badics, T., & Boros, E. (1998). Minimisation of half-products. *Mathematics of Operations Research, 33*, 649–660.

Bagchi, U.B. (1989). Simultaneous minimisation of mean and variation of flow-time and waiting time in single machine systems. *Operations Research, 37*, 118–125.

Bar-Noy, A., Bhatia, R., Naor, J.S., & Schiber, B. (2002). Minimising service and operation costs of periodic scheduling. *Mathematics of Operations Research, 27*, 518–544.

Biskup, D. (1999). Single-machine scheduling with learning considerations. *European Journal of Operational Research, 115*, 173–178.

Biskup, D. (2008). A state-of-the-art review on scheduling with learning effects. *European Journal of Operational Research, 188*, 315–329.

Błażewicz, J., Ecker, K.H., Pesch, E., Schmidt, G., & Węglarz, J. (2010). *Scheduling Computer and Manufacturing Processes.* Berlin: Springer.

Bourgeois, F., & Lassale, J.C. (1971). An extension of the Munkres algorithm for the assignment problem to rectangular matrices. *Communications of the ACM, 14*, 802–804.

Braun, O., & Schmidt, G. (2003). Parallel processor scheduling with limited number of preemptions. *SIAM Journal on Computing, 32*, 671–680.

Brehob, M., Torng, E., & Uthaisombut, P. (2000). Applying extra-resource analysis to load balancing. *Journal of Scheduling, 3*, 273–288.

Browne, S., & Yechiali, U. (1990). Scheduling deteriorating jobs on a single processor. *Operations Research, 38*, 495–498.

Brucker, P. (2007). *Scheduling Algorithms*. Guildford, Surrey: Springer.

Bruno, L.J., Coffman, E.G., & Sethi, R. (1974). Scheduling independent tasks to reduce mean finishing time. *Communications of the ACM, 17*, 382–387.

Burkard, R.E., Klinz, B., & Rudolf, R. (1996). Perspectives of Monge properties in optimization. *Discrete Applied Mathematics, 70*, 95–161.

Burkard, R.E., Deineko, V.G., van Dal, R., van der Veen, J.A.A., & Woeginger, G.J. (1998). Well-solvable special cases of the traveling salesman problem: a survey. *SIAM Review, 40*, 496–546.

Burkard, R., Dell'Amico, M., & Martello, S. (2009). *Assignment Problems.* Philadelphia: SIAM.

Chekuri, C., Goel, A., Khanna, S., & Kumar, A. (2004). Multi-processor scheduling to minimise flow time with ε-resource augmentation. In: *Proceedings of STOC'04,* 363–372, http://doi.acm.org/10.1145/1007352.1007411.

Chen, B. (2004). Parallel machine scheduling for early completion. In: Leung, J.Y.T. (ed). *Handbook of Scheduling: Algorithms, Models and Performance Analysis*. London: Chapman & Hall/CRC, pp 9-175–9-184.

Chen, B. (2012). The power of additional machines under LPT. Unpublished manuscript, University of Warwick, U.K.

Chen, B., Potts, C.N., & Woeginger, G.J. (1998). A review of machine scheduling: Complexity, algorithms and approximability. *Handbook of Combinatorial Optimisation, 3*, 21–169.

Chen, J.-S. (2008). Optimisation models for the tool change scheduling problem. *Omega, 36*, 888–894.

Chen, W.-J. (2009). Minimising number of tardy jobs on a single machine subject to periodic maintenance. *Omega, 37*, 591–599.

Cheng, T.C.E., Ding, Q., & Lin, B.M.T. (2004). A concise survey of scheduling with time-dependent processing times. *European Journal of Operational Research, 152*, 1–13.

Conway, R.W., Maxwell, W.L., & Miller, L.W. (1967). *Theory of Scheduling.* Reading (MA): Addison Wesley.

Dósa, G., & Tan, Z. (2010). New upper and lower bounds for online scheduling with machine cost. *Discrete Optimisation, 7*, 125–135.

Došlić, T. (2009). Log-convexity of combinatorial sequences from their convexity, *Journal of Mathematical Inequalities, 3*, 437–442.

Erel, E., & Ghosh, J.B. (2008). FPTAS for half-products minimisation with scheduling applications. *Discrete Applied Mathematics*, 156, 3046–3056.

Federgruen, A., & Mosheiov, G. (1997). Single machine scheduling problems with general breakdowns, earliness and tardiness costs. *Operations Research, 45*, 66–71.

Flajolet, P., & Sedgewick, R. (2009). *Analytic Combinatorics.* Cambridge: Cambridge University Press, pp 39–46.

Gawiejnowicz, S. (1996). A note on scheduling on a single processor with speed dependent on a number of executed jobs. *Information Processing Letters, 57*, 297–300.

Gawiejnowicz, S. (2008). *Time-Dependent Scheduling.* Berlin: Springer.

Garey, M.R., Johnson, D.S. (1978). Strong NP-completness results: motivation, examples and implications, *Journal of the ACM, 25*, 499–508.

Garey, M.R., Johnson, D.S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness.* San Francisco: Freeman.

Gopalakrishnan, M., Ahire, S.L., & Miller, D.M. (1997). Maximising the effectiveness of a preventive maintenance system: an adaptive modeling approach. *Management Science, 43*, 827–840.

Gordon, V.S., Potts, C.N., Strusevich, V.A., & Whitehead, J.D. (2008). Single machine scheduling models with deterioration and learning: Handling precedence constraints via priority generation. *Journal of Scheduling, 11*, 357–370.

Gordon, V.S., Proth, J.M., & Strusevich, V.A. (2004). Scheduling with due date assignment. In: Leung, J.Y.T. (ed). *Handbook of Scheduling: Algorithms, Models and Performance Analysis*. London: Chapman & Hall/CRC, pp 21-1–21-22.

Gordon, V.S., & Strusevich, V.A. (2009). Single machine scheduling and due date assignment with positionally dependent processing time. *European Journal of Operational Research, 198*, 57–62.

Gordon, V.S., & Tarasevich, A.A. (2009). A note: Common due date assignment for a single machine scheduling with the rate-modifying activity. *Computers and Operations Research, 36*, 325–328.

Graham, R.L. (1966). Bounds for certain multiprocessing anomalies. *Bell System Technical Journal, 45*, 1563–1581.

Graham, R.L. (1967). Bounds on multiprocessing timing anomalies. *SIAM Journal of Applied Mathematics, 17*, 263–269.

Graham, R.L., Knuth, D.E., & Patashnik, O. (1989). *Concrete Mathematics*. New York: Addison-Wesley.

Graham, R.L., Lawler, E.L., Lenstra, J.K., & Rinnooy Kan, A.H.G. (1979). Optimisation and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics, 5*, 287–326.

Grigoriev, A., van de Klundert, J., & Spieksma, F.C.R. (2006). Modelling and solving periodic maintenance problem. *European Journal of Operational Research, 172*, 783–797.

Hardy, G.H., Littlewood, J.E., & Polya, G. (1934). *Inequalities*. London: Cambridge University Press.

Hartmann, S., & Briskorn, D. (2010). A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research, 207*, 1–14.

He, Y., & Cai., S. (2002). Semi-online scheduling with machine cost. *Journal of Computer Science and Technology, 17*, 781–787.

Hochbaum, D.S., & Hong, S.-P. (1995). About strongly polynomial time algorithms for quadratic optimisation over submodular constraints. *Mathematical Programming, 69*, 269–309.

Horn, W.A. (1973). Minimising average flow time with parallel machines. *Operations Research, 21*, 846–847.

Imreh, C. (2009). Online scheduling with general machine cost functions. *Discrete Applied Mathematics, 157*, 2070–2077.

Imreh, C., & Noga, J. (1999). Scheduling with machine cost. *Lecture Notes in Computer Science, 1671*, 168–176.

Jackson, J.R. (1956). An extension of Johnson's results on job lot sceduling. *Naval Research Logistics Quarterly, 3*, 201–203.

Janiak, A., Krysiak, T., & Trela, R. (2011). Scheduling problems with learning and ageing effects: A Survey. *Decision Making in Manufacturing Services, 5*, 19–36.

Ji, M., & Cheng, T.C.E. (2010). Scheduling with job-dependent learning effects and multiple rate-modifying activities. *Information Processing Letters, 110*, 460–463.

Ji, M., He, Y., & Cheng, T.C.E. (2007). Single-machine scheduling with periodic maintenance to minimise makespan, *Computers and Operations Research, 34*, 1764–1770.

Jiang, Y.-W., & He., Y. (2005). Preemptive online algorithms for scheduling with machine cost. *Acta Informatica, 41*, 315–340.

Jiang, Y.-W., & He., Y. (2006). Semi-online algorithms for scheduling with machine costs. *Journal of Computer Science and Technology, 21*, 984–988.

Johnson, S.M. (1954). Optimal two- and three-stage productions schedules with setup times included. *Naval Research Logistics Quarterly, 1*, 61–68.

Jurisch, B., Kubiak, W., & Józefowska, J. (1997). Algorithms for minclique scheduling problems. *Discrete Applied Mathematics, 72*, 115–139.

Kabadi, S.N. (2007). Polynomially solvable cases of the TSP. In: Gutin, G., & Punnen, A.P. (eds). *The Traveling Salesman Problem and Its Variations*. Berlin: Springer, pp 489–583.

Kalyanasundaram, B., & Pruhs, K. (2000). Speed is as powerful as clairvoyance. *Journal of the ACM, 47,* 617–643.

Karp, R.M. (1972). Reducibility among combinatorial problems. In: Miller, R.E., & Thatcher. J.W. (eds.). Complexity of Computer Computations. New York: Plenum Press, pp 85–103.

Katoh, N., & Ibaraki, T. (1998). Resource allocation problems. In: Du, D.-Z., & Pardalos, P.M. (eds). *Handbook of Combinatorial Optimisation*. Dordrecht: Kluwer, Vol. 2, pp 159–260.

Kellerer, H., Mansini, R., Pferschy, U., & Speranza, M.G. (2003). An efficient fully polynomial approximation scheme for the Subset-Sum Problem. *Journal of Computer and System Sciences, 66*, 349–370.

Kellerer, H., Pferschy, U., & Pisinger, D. (2004). *Knapsack Problems*. Berlin: Springer.

Kellerer, H., Rustogi, K., & Strusevich, V. A. (2012c). Approximation schemes for scheduling on a single machine subject to cumulative deterioration and maintenance. *Journal of Scheduling,* DOI:10.1007/s10951-012-0287-8.

Kellerer, H., & Strusevich, V.A. (2010a). Fully polynomial approximation schemes for a symmetric quadratic knapsack problem and its scheduling applications. *Algorithmica, 57*, 769–795.

Kellerer, H., & Strusevich, V.A. (2010b). Minimising total weighted earliness-tardiness on a single machine around a small common due date: An FPTAS using quadratic knapsack. *International Journal of Foundations of Computer Science, 21*, 357–383.

Kellerer, H., & Strusevich, V.A. (2012). The symmetric quadratic knapsack problem: approximation and scheduling applications. *4OR, 10*, 111–161.

Koulamas, C. (2010). A note on single-machine scheduling with job-dependent learning effects. *European Journal of Operational Research, 207*, 1142–1143.

Koulamas, C., & Kyparisis, G.J. (2007). Single-machine and two-machine flowshop scheduling with general learning functions. *European Journal of Operational Research, 178*, 402–407.

Kovalyov, M.Y., & Kubiak, W. (1998). A fully polynomial approximation scheme for minimising makespan of deteriorating jobs. *Journal of Heuristics, 3*, 287–297.

Kubiak, W. (1995). New results on the completion time variance minimisation. *Discrete Applied Mathematics, 58*, 157–168.

Kubzin, M.A., & Strusevich, V.A. (2005). Two-machine flow shop no-wait scheduling with machine maintenance. *4OR, 3*, 303–313.

Kubzin, M.A., & Strusevich, V.A. (2006). Planning machine maintenance in two-machine shop scheduling. *Operations Research, 54*, 789–800.

Kuhn, H.W. (1955). The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly, 2*, 83–97.

Kuo, W.-H., & Yang, D.-L. (2006a). Minimising the makespan in a single machine scheduling problem with a time-based learning effect. *Information Processing Letters, 97*, 64–67.

Kuo, W.-H., & Yang, D.-L. (2006b). Minimising the total completion time in a single-machine scheduling problem with a time-dependent learning effect. *European Journal of Operational Research, 174*, 1184-1190.

Kuo, W.-H., & Yang, D.-L. (2007). Single machine scheduling with past-sequence-dependent setup times and learning effects. *Information Processing Letters, 102*, 22–26.

Kuo, W.-H., & Yang, D.-L. (2008a). Minimising the makespan in a single-machine scheduling problem with the cyclic process of an aging effect. *Journal of the Operational Research Society, 59*, 416–420.

Kuo, W.-H., & Yang, D.-L. (2008b). Parallel-machine scheduling with time dependent processing times. *Theoretical Computer Science, 393*, 204–210.

Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., & Shmoys, D.B. (1993). Sequencing and scheduling: algorithms and complexity. In: Graves, S., Rinnooy Kan, A.H.G., & Zipkin, P. (eds). *Handbooks in Operations Research and Management Science: Logistics of Production and Inventory.* North Holland: Elsevier, Vol. 4, pp 445–522.

Lee, C.-Y. (2004). Machine scheduling with availability constraints. In: Leung, J.Y.T. (ed). *Handbook of Scheduling: Algorithms, Models and Performance Analysis.* London: Chapman & Hall/CRC, pp 22-1–22-13.

Lee, C.-Y., & Leon, V.J. (2001). Machine scheduling with a rate-modifying activity. *European Journal of Operational Research, 128*, 119–128.

Lee, C.-Y., & Strusevich, V.A. (2005). Two-machine shop scheduling with an uncapacitated interstage transporter. *IIE Transactions, 37*, 725–736.

Lee, W.-C., & Wu, C.-C. (2009). A note on single-machine group scheduling problems with position-based learning effect. *Applied Mathematical Modelling, 33*, 2159–2163.

Leung, J.Y.T. (2004). *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, Chapters 1–3. London: Chapman & Hall/CRC, pp 1-1–3-18.

Leyvand, Y., Shabtay, D., & Steiner, G. (2010). A unified approach for scheduling with convex resource consumption functions using positional penalties. *European Journal of Operational Research, 206*, 301–312.

Liao, C.-J., & Chen, W.-J. (2003). Single-machine scheduling with periodic maintenance and nonresumable jobs. *Computers and Operations Research, 30*, 1335–1347.

Lodree Jr., E.J., & Geiger, C.D. (2010). A note on the optimal sequence position for a rate-modifying activity under simple linear deterioration. *European Journal of Operational Research, 201*, 644–648.

Ma, Y., Chu, C., & Zuo, C. (2010). A survey of scheduling with deterministic machine availability constraints. *Computers and Industrial Engineering, 58*, 199–211.

Marshall, A.W., & Olkin, I. (1979). *Inequalities: The Theory of Majorisation and Its Applications*. New York: Academic Press.

McNaughton, R. (1959). Scheduling with deadlines and loss functions. *Management Science, 6*, 1–12.

Mercer, A.M. (2005). Polynomials and convex sequence inequalities. *Journal of Inequalities in Pure and Applied Mathematics, 6*, 1–4.

Mittenhal, J., Raghavachari, M., & Rana, A.I. (1995). V-shapes and Lambda-spared properties for optimal single-machine schedules for a class of nonseparable penalty-functions. *European Journal of Operational Research, 86*, 262–269.

Mosheiov, G. (1991). V-shaped policies for scheduling deteriorating jobs. *Operations Research, 39*, 979–991.

Mosheiov, G. (1994). Scheduling jobs under simple linear deterioration. *Computers and Operations Research, 21*, 653–659.

Mosheiov, G. (2001a). Scheduling problems with a learning effect. *European Journal of Operational Research, 132*, 687–693.

Mosheiov, G. (2001b). Parallel machine scheduling with a learning effect. *Journal of the Operational Research Society,52*, 1165–1169.

Mosheiov, G. (2005). A note on scheduling deteriorating jobs. *Mathematical and Computer Modelling, 41*, 883–886.

Mosheiov, G. (2008). Minimising total absolute deviation of job completion times: extensions to position-dependent processing times and parallel identical machines. *Journal of the Operational Research Society, 59,* 1422–1424.

Mosheiov, G., & Oron, D. (2006). Due-date assignment and maintenance activity scheduling problem. *Mathematical and Computer Modelling, 44*, 1053–1057.

Mosheiov, G., & Sarig, A. (2009). Scheduling a maintenance activity and due-window assignment on a single machine. *Computers and Operations Research, 36*, 2541–2545.

Mosheiov, G., & Sidney, J.B. (2003). Scheduling with general job-dependent learning curves. *European Journal of Operational Research, 147*, 665–670.

Mosheiov, G., & Sidney, J.B. (2010). Scheduling a deteriorating maintenance activity on a single machine. *Journal of the Operational Research Society, 61*, 882–887.

Ng, C.T., Cheng, T.C.E., Bachman, A., & Janiak. A. (2002). Three scheduling problems with deteriorating jobs to minimize the total completion time. *Information Processing Letter, 81*, 327–333.

Nyblom, M.A. (2002). Some curious sequences involving floor and ceiling functions. *American Mathematical Monthly, 109*, 559–564.

Nyman, D., & Levitt, J. (2001). *Maintenance Planning, Scheduling and Coordination.* New York: Industrial Press.

Okołowski, D., & Gawiejnowicz, S. (2010). Exact and heuristic algorithms for parallel-machine scheduling with DeJong's learning effect. *Computers and Industrial Engineering, 59*, 272–279.

Palmer, D. (1999). *Maintenance Planning and Scheduling Handbook.* New York: McGraw Hill.

Panwalkar, S.S., Smith, M.L., & Seidmann, A. (1982). Common due date assignment to minimise total penalty for the one machine scheduling problem. *Operations Research, 30*, 391–399.

Qi, X., Chen, T., & Tu, F. (1999). Scheduling the maintenance on a single machine. *Journal of the Operational Research Society, 50*, 1071–1078.

Qian, F., Strusevich, V.A., Gribkovskaia, I., Halskau, Ø. (2011). Minimisation of passenger takeoff and landing risk in offshore helicopter transportation: Models, approaches and analysis. Report SORG-07-2011, University of Greenwich, London, UK.

Qian, J., & Steiner, G. (2012). Fast algorithms for scheduling with learning effects and time-dependent processing times on a single machine. *European Journal of Operational Research, 225*, 547–551.

Różycki, R., & Węglarz, J. (2012). Power-aware scheduling of preemptable jobs on identical parallel processors to meet deadlines. *European Journal of Operational Research, 218*, 68–75.

Rustogi, K., & Strusevich, V.A. (2011). Convex and V-shaped sequences of sums of functions that depend on ceiling functions. *Journal of Integer Sequences, 14*, Article 11.1.5, http://www.cs.uwaterloo.ca/journals/JIS/VOL14/Strusevich/strusevich2.html.

Rustogi, K., & Strusevich, V. A. (2012a). Single machine scheduling with general positional deterioration and rate-modifying maintenance. *Omega, 40*, 791–804.

Rustogi, K., & Strusevich, V. A. (2012b). Simple matching vs linear assignment in scheduling models with positional effects: A critical review. *European Journal of Operational Research, 222*, 393–407.

Rustogi, K., & Strusevich, V. A. (2013a). Parallel machine scheduling: Impact of adding extra machines. Accepted for publication in *Operations Research.*

Rustogi, K., & Strusevich, V. A. (2013b). Combining time and position dependent effects on a single machine subject to rate-modifying activities. Accepted for publication in *Omega.*

Rustogi, K., & Strusevich, V. A. (2013c). Single machine scheduling with time-dependent deterioration and rate-modifying maintenance. Submitted to *Journal of the Operational Research Society.*

Shabtay, D., & Steiner, G. (2007). A survey of scheduling with controllable processing times. *Discrete Applied Mathematics, 155*, 1643–1666.

Smith, W.E. (1956). Various optimisers for single state production. *Naval Research Logistics Quarterly, 3*, 66–69.

Toader, G. (1996). On Chebychev's inequality for sequences. *Discrete Mathematics, 161*, 317–322.

Wang, J.-B. (2006). A note on scheduling problems with learning effects and deteriorating jobs, *International Journal of Systems Science, 37*, 827–832.

Wang, J.-J., Wang, J.-B., & Liu, F. (2011). Parallel machines scheduling with a deteriorating maintenance activity. *Journal of the Operational Research Society, 62*, 1898–1902.

Węglarz, J., Józefowska, J., Mika, M., & Waligóra, G. (2011). Project scheduling with finite or infinite number of activity processing modes - A survey. *European Journal of Operational Research, 208*, 177–205.

Winston, W.L. (2003). *Operations Research: Applications and Algorithms.* Duxbury Press.

Woeginger, G.J. (2005). A comment on scheduling two parallel machines with capacity constraint. *Discrete Optimisation, 2*, 269–272.

Wright, T.P. (1936). Factors affecting the cost of airplanes. *Journal of Aeronautical Sciences, 3*, 122–128.

Wu, S., & Debnath, L. (2007). Inequalities for convex sequences and their application. *Computers and Mathematics with Applications, 54*, 525–534.

Wu, C.-C., & Lee, W.-C. (2008). Single-machine scheduling problems with a learning effect. *Applied Mathematical Modeling, 32*, 1191–1197.

Wu, C.-C., Yin, Y., & Cheng, S.-R. (2011). Some single-machine scheduling problems with a truncation learning effect. *Computers and Industrial Engineering, 60*, 790–795.

Yang, S.-J. (2010). Single-machine scheduling problems with both start-time dependent learning and position dependent aging effects under deteriorating maintenance consideration. *Applied Mathematics and Computation, 217*, 3321–3329.

Yang, S.-J. (2012). Single-machine scheduling problems simultaneously with deterioration and learning effects under deteriorating multi-maintenance activities consideration. *Computers and Industrial Engineering, 62*, 271–275.

Yang, D.-L., & Kuo, W.-H. (2009). Single-machine scheduling with both deterioration and learning effects. *Annals of Operational Research, 172*, 315–327.

Yang, S.-J., & Yang, D.-L. (2010a). Minimising the makespan single-machine scheduling with aging effects and variable maintenance activities. *Omega, 38*, 528–533.

Yang, S.-J. & Yang, D.-L. (2010b). Minimising the total completion time in single-machine scheduling with ageing/deteriorating effects and deteriorating maintenance activities. *Computers and Mathematics with Applications, 60*, 2161–2169.

Yin, Y., Xu, D., Sun, K., & Li, H. (2009). Some scheduling problems with general position-dependent and time-dependent learning effects. *Information Sciences, 179,* 2416–2425.

Yin, Y., & Xu, D. (2011). Some single-machine scheduling problems with general effects of learning and deterioration. *Computers and Mathematics with Applications, 61*, 100–108.

Zhao, C.-L., & Tang, H.-Y. (2010). Single machine scheduling with general job-dependent aging effect and maintenance activities to minimise makespan. *Applied Mathematical Modelling, 34*, 837–841.

	Uni1 $[1, 40]$	**Uni2** $[1, 100]$	**Norm1** $\mu = 20,\ \sigma = 5$	**Norm2** $\mu = 40,\ \sigma = 10$	**Exp1** $\mu = 20$	**Exp2** $\mu = 40$
$\kappa = 1,$	37/63/0/0	72/28/0/0	8/50/15/27	56/44/0/0	85/15/0/0	92/8/0/0
$n = 50$	1.1692 / 1.2516	1.2024 / 1.2891	1.1403 / 1.1846	1.2216 / 1.2885	1.0909 / 1.2099	1.1080 / 1.2717
$\kappa = 1,$	0/0/51/49	46/54/0/0	0/0/48/52	12/79/3/6	81/19/0/0	95/5/0/0
$n = 100$	1.1385 / 1.1772	1.1905 / 1.2460	1.1376 / 1.1693	1.1648 / 1.2026	1.1171 / 1.1935	1.1162 / 1.2190
$\kappa = 1,$	0/0/53/47	0/0/52/48	0/0/48/52	0/0/54/46	47/53/0/0	81/19/0/0
$n = 500$	1.0949 / 1.1094	1.1336 / 1.1529	1.0645 / 1.0744	1.1077 / 1.1210	1.1049 / 1.1393	1.1345 / 1.1775
$\kappa = 5,$	0/0/48/52	0/0/49/51	0/0/65/35	0/0/49/51	32/50/9/9	64/36/0/0
$n = 50$	1.0930 / 1.1441	1.1243 / 1.1793	1.0764 / 1.1060	1.1366 / 1.1692	1.0807 / 1.1536	1.0864 / 1.1770
$\kappa = 5,$	0/0/55/45	0/0/51/49	0/0/65/35	0/0/31/69	13/31/28/28	63/37/0/0
$n = 100$	1.0779 / 1.1139	1.1178 / 1.1598	1.0554 / 1.0781	1.0952 / 1.1390	1.0785 / 1.1263	1.1033 / 1.1828
$\kappa = 5,$	0/0/51/49	0/0/55/45	0/0/67/33	0/0/45/55	0/0/42/58	4/33/34/29
$n = 500$	1.0439 / 1.0556	1.0695 / 1.0815	1.0300 / 1.0395	1.0434 / 1.0568	1.0601 / 1.0763	1.0817 / 1.1015
$\kappa = 10,$	0/0/54/46	0/0/57/43	0/0/79/21	0/0/36/64	7/30/38/25	38/56/3/3
$n = 50$	1.0672 / 1.1063	1.1043 / 1.1563	1.0504 / 1.0782	1.0878 / 1.1307	1.0642 / 1.1171	1.0835 / 1.1609
$\kappa = 10,$	0/0/49/51	0/0/49/51	0/0/75/25	0/0/52/48	4/7/47/42	23/57/9/11
$n = 100$	1.0561 / 1.0900	1.0872 / 1.1353	1.0404 / 1.0598	1.0644 / 1.0888	1.0612 / 1.1059	1.0835 / 1.1240
$\kappa = 10,$	0/0/53/47	0/0/57/43	0/0/56/44	0/0/70/30	0/0/60/40	1/1/43/55
$n = 500$	1.0307 / 1.0388	1.0494 / 1.0628	1.0210 / 1.0289	1.0309 / 1.0387	1.0457 / 1.0679	1.0650 / 1.0902

Table 11.1: Results of computational experiment for Algorithm 3