Technische Universität München

Lehrstuhl für Angewandte Geometrie und Diskrete Mathematik
(M9)

# Deterministic, Stochastic, and Robust Cost-Aware Scheduling

**Roman Rischke**

Vollständiger Abdruck der von der Fakultät für Mathematik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

| | | |
|---|---|---|
| **Vorsitzender:** | | Prof. Dr. Christian Liedtke |
| **Prüfer der Dissertation:** | 1. | Prof. Dr. Nicole Megow |
| | 2. | Prof. Dr. Marc Uetz<br>Universität Twente / Niederlande |

Die Dissertation wurde am 17.05.2016 bei der Technischen Universität München eingereicht und durch die Fakultät für Mathematik am 01.07.2016 angenommen.

# Acknowledgments

I am extremely grateful to many people who made this thesis possible through their invaluable advise and support. First and foremost, I thank my advisor Nicole Megow for sharing her passion for optimization under uncertainty and for introducing me to the research community. Her office door was literally always open for research and non-research matters and I enjoyed the numerous inspirational discussions with her. Nicole's great support and her trust were the basis for this thesis. Special thanks also go to Leen Stougie, whom I always saw as my second advisor. I especially thank him for the joyful weeks of intense joint research in Amsterdam and Berlin, for his unshakable optimism, and for all the proofreading. I would also like to express my gratitude to Marc Uetz for taking the second assessment of this thesis.

Most of the results in this thesis originate from the collaboration with Lin Chen, whom I always considered as a mentor. His goal-oriented and rigorous way of doing research together with his broad knowledge and his clear mind shaped my view of research a lot.

I also thank my other coauthors Fidaa Abed, Yann Disser, Martin Groß, Julie Meißner, Alexander Richter, and José Verschae for the great collaboration and for proofreading drafts of different parts of this thesis. Their detailed comments and suggestions helped to improve the presentation of this thesis and clarified many arguments. I am also grateful to Kevin Schewior for reading and commenting on parts of this thesis and for all the fun we had together.

I started my PhD project in the combinatorial optimization and graph algorithms group at TU Berlin and I very much enjoyed being a member of this family. I thank all members of COGA for the great time we spent together. I will never forget the numerous table soccer tournaments and the lively coffee and tea breaks. Special thanks go to Rolf H. Möhring and Martin Skutella, who formed this outstanding research environment.

I finished my PhD project in the applied geometry and discrete mathematics group at TU München. I thank all M9 members for the wonderful time we had.

This work was financially supported by the German Research Foundation (DFG) under contract ME 3825/1. Furthermore, I thank the research train-

ing group “Methods for Discrete Structures” in Berlin, the Berlin Mathematical School and the International School of Applied Mathematics in München for all the valuable soft skill seminars and workshops.

Without the constant support and encouragement of my family and friends this thesis would not have been possible. I owe special thanks to Daniela Luft for all her love and unconditional support. Special thanks also go to my parents, Evelyn and Steffen Rischke, my brother, Tobias Rischke, and the family of Daniela for all the encouraging words and support over the years.

I am extremely grateful to all of you.

München, May 2016 Roman Rischke

# Abstract

Scheduling concerns the temporal allocation of tasks to scarce resources with the objective of optimizing some performance measure subject to certain side constraints. It is involved for example when the execution of a computer program is assigned to the central processing unit (CPU) or when the take-offs and landings of airplanes are allocated to the runways of an airport. Classical scheduling problems do not address the fact that in practice we usually need to pay a certain cost when using scarce resources. Moreover, this cost may also vary over time, as for example electricity cost fluctuating over day time, or reservation cost for computational power in the cloud that may vary depending on the law of supply and demand. It is a challenging task to optimize the tradeoff between the resource provisioning cost and the scheduling quality. Furthermore, the allocation of the tasks to the scarce resources is usually carried out in the future. As the future is unpredictable, scheduling problems have an intrinsic data uncertainty in practice. Machines might unexpectedly break down or a job might have a much longer processing time than expected. Our scheduling models should take also this issue into account. We consider a natural generalization of classical machine scheduling problems in which occupying a time slot incurs certain cost that may vary over time and which must be paid in addition to the actual scheduling cost. We also propose a general two-stage model for cost-aware scheduling with uncertain input data. Furthermore, we investigate the problem of scheduling the maintenance of edges in a network with the objective of preserving connectivity between two distinguished vertices of the network. This is motivated by the servicing and replacement in transportation and telecommunication networks which requires a well-planned schedule to minimize the performance loss through temporary outages. This problem can be seen as a deterministic cost-aware scheduling problem in which we pay a certain penalty cost whenever the two distinguished vertices in our network are disconnected. Our contributions are optimal algorithms, results on the computational complexity and approximation algorithm for these general scheduling problems.

# Zusammenfassung

Scheduling beschäftigt sich mit der zeitlichen Zuordnung von Aufgaben zu knappen Ressourcen mit dem Ziel der Optimierung einer gegebenen Zielfunktion unter Einhaltung gegebener Restriktionen. Scheduling kommt zum Beispiel zur Anwendung wenn die Ausführung eines Computerprogramms dem Prozessor zugewiesen wird oder wenn Starts und Landungen von Flugzeugen den Landebahnen eines Flughafens zugeteilt werden. Klassische Scheduling-Probleme berücksichtigen jedoch nicht, dass in der Praxis für die Nutzung von knappen Ressourcen üblicherweise Kosten anfallen. Zusätzlich weisen diese Kosten oftmals eine zeitliche Variabilität auf, wie zum Beispiel über den Tag hinweg schwankende Stromkosten oder dem Gesetz von Angebot und Nachfrage folgende Reservierungskosten für Kapazitäten in der Cloud. Das Finden eines optimalen Kompromisses zwischen den Kosten für die Ressourcenbeschaffung und der Qualität des Schedules stellt eine herausfordernde Aufgabenstellung dar. Hinzu kommt, dass die Zuordnung der Aufgaben zu den knappen Ressourcen üblicherweise in der Zukunft ausgeführt wird. Da die Zukunft jedoch unvorhersehbar ist, weisen Scheduling-Probleme eine innewohnende Datenunsicherheit in der Praxis auf. Maschinen können unvorhersehbar zusammenbrechen oder eine Aufgabe kann länger in der Abarbeitung dauern als erwartet. Scheduling-Modelle sollten auch dieses Problem berücksichtigen. Wir betrachten eine natürliche Verallgemeinerung von klassischen Maschinen-Scheduling-Problemen, in der die Belegung eines Zeitslots zeitabhängige Kosten verursacht, die zusätzlich zu den eigentlichen Scheduling-Kosten anfallen. Wir schlagen auch ein allgemeines zwei-stufiges Modell für kostenbewusstes Maschinen-Scheduling mit Datenunsicherheit vor. Zudem untersuchen wir das Problem der Koordination von Wartungsaufgaben auf Kanten in einem Netzwerk mit dem Ziel der Aufrechterhaltung der Konnektivität zwischen zwei ausgewiesenen Netzwerkknoten. Dies lässt sich durch Wartungsarbeiten in Infrastruktur- und Telekommunikationsnetzwerken motivieren, die eine sorgfältige Planung erfordern, um Leistungseinbußen durch temporären Ausfall zu minimieren. Dieses Problem kann als deterministisches kostenbewusstes Scheduling-Problem aufgefasst werden, in welchem Strafkosten immer dann anfallen, wenn die zwei ausgewiesenen Netzwerkknoten nicht verbunden sind. Unser Beitrag sind optimale Algorithmen, Resultate hinsichtlich der Komplexität und Approximationsalgorithmen für diese allgemeinen Scheduling-Probleme.

# Contents

# 1

# Introduction

Scheduling addresses the temporal allocation of tasks to scarce resources with the objective of optimizing some performance measure subject to certain side constraints. It is involved for example when the execution of a computer program is assigned to the central processing unit (CPU) or when the take-offs and landings of airplanes are allocated to the runways of an airport. Objectives may be, among many others, the minimization of the time point when the last task is completed, also known as makespan minimization, or the minimization of the average weighted completion time. The former objective is often used in project scheduling and the latter one serves as a measure for the quality of service. Examples for possible side constraints are precedence relations between the tasks, restrictions on the time window where a task can be processed, or the requirement that the processing of a task must not be interrupted. An explicit description of the temporal allocation of the given tasks to the given resources is referred to as a *schedule*.

Scheduling has its roots in management science, operations research, theoretical computer science, and combinatorial optimization. In the early 20th century, Frederick W. Taylor [82] described 'principles of scientific management' to improve the efficiency of labor, in particular, by replacing rule-of-thumb methods by scientific methods for the management. With his work, Taylor established the basis for the research field that we call today *management science* which contains scheduling as a sub-discipline. Later, in World War II, many planning and logistical problems required scientific approaches from which the field of *operations research* evolved. Today, the boundaries between management science and operations research are fluent. Johnson [50], Jackson [48, 49] and Smith [78] started the rigorous study of scheduling problems in the 1950s. Already in these early publications, tasks are also called *jobs*, which is a generally accepted convention in the scheduling community. In the middle of the 20th century, the development of operating systems for computers promoted the scientific study of scheduling problems as a sub-discipline of theoretical computer science. In this context, the notion of *machine scheduling* has been established, because the scarce resources correspond to machines and computers, respectively. The focus of this thesis is

on machine scheduling problems.

Combinatorial optimization as a sub-discipline of discrete mathematics is also concerned with scheduling problems. In combinatorial optimization, we aim at finding an optimal object among a discrete set of feasible objects. Such an object could be a schedule, as considered in this thesis, but it could for example also be a path or a tour in a given network. Assigning every object a certain value gives rise to the question of finding an optimal object with respect to this value. For finding an optimal object, algorithms are designed. An *algorithm* for a certain problem takes as input an instance of the problem and returns after finitely many operations a solution for the given problem instance. The performance of an algorithm can be assessed in terms of the running time and the solution quality. Based on this assessment, which is usually with respect to the worst-case behavior of the algorithm, we ideally want to classify problems into ones that admit an efficient algorithm and those that are computationally hard. The word 'efficient' means here that the running time can be bounded by a polynomial in the input size. Cook [27] and Karp [52] laid the basis for this problem classification with the development of the complexity theory at the beginning of the 1970s. Since then, many scheduling problems were shown to be NP-hard [38,60]. This, in turn, rules out the existence of an optimal deterministic polynomial-time algorithm for such scheduling problems under widely believed assumptions.

As we probably cannot hope for an NP-hard problem to have an algorithm that returns for any instance of the problem an optimal solution in polynomial time, we need to relax at least one of the requirements. In this thesis, we shall relax the optimality requirement, that is, we focus on approximation algorithms. An *approximation algorithm* for a certain optimization problem returns in polynomial time for any problem instance a feasible solution that is within a guaranteed factor of the optimal value. Proving such a performance guarantee for an approximation algorithm requires a rigorous worst-case analysis. We remark that this approach differs from the study of heuristics, where the performance is usually assessed based on empirical observations. It is interesting to note that the first approximation algorithm in literature was given for a machine scheduling problem by Graham [40].

In practical applications of scheduling, our overall objective is usually not only affected by the traditional scheduling cost, but also governed by some other cost function such as cost for operating or renting machines. Moreover, these two cost functions may be in conflict. For example, the additional cost function may suggest delaying jobs in favor of cheaper cost, whereas the scheduling cost usually increases with the delay of jobs. Finding an optimal tradeoff between these two costs is then a challenging task. This motivates the topic of this thesis, namely *cost-aware scheduling problems*, which take this aspect into account. We now introduce the cost-aware scheduling problems that we study in this thesis.

## 1.1 Deterministic Cost-Aware Machine Scheduling

Classical scheduling problems do not address the fact that in practice we usually need to pay a certain cost when using scarce resources. Moreover, this cost may also vary over time, as for example labor cost that may vary depending on the day of the week, or the hour of the day [83], or electricity cost fluctuating over day time [57]. On the one hand, the latter have economically a huge impact on facilities with enormous power consumption such as large data centers. On the other hand, these fluctuations reflect the imbalance between generation and consumption of electricity on a daily or weekly basis. In fact, peak demand periods in an energy network are usually inefficient, because either energy is stored or additional power plants are kept ready for these rather short peak demand periods [14]. Both options require additional resource consumption, which we aim to minimize. Hence, cost-aware scheduling is economically profitable and supports an eco-aware usage and generation of energy.

Another motivation for cost-aware machine scheduling stems from cloud computing. Users of such services, for example, of Amazon EC2, are offered time-varying pricing schemes for processing jobs on a remote cloud server [1]. It is a challenging task for them to optimize the tradeoff between the resource provisioning cost and the scheduling quality.

With these motivations in mind, we consider a natural generalization of classical machine scheduling problems in which occupying a time slot incurs certain cost that may vary over time and which must be paid in addition to the actual scheduling cost. This framework has been proposed recently by Wan and Qi [83] and Kulkarni and Munagala [57]. Wan and Qi [83] consider cost-aware single machine scheduling problems with the constraint that the processing of a job must not be preempted. They show for some classical polynomial-time solvable scheduling problems that they become strongly NP-hard in the cost-aware setting. Kulkarni and Munagala [57] focused on more general preemptive single machine scheduling problems, in which the job set and the time-varying cost for using the machine are revealed over time, known as *online model*. Compared to that, deterministic models in which the input is clearly specified and not uncertain, are often also called *offline models*.

Note that cost-awareness in scheduling allows to model scheduling problems in which the machines are unreliable, that is, the machines might break down for some time and in such periods the machines are not available for processing the jobs. We can set the cost in the periods of break-downs to a sufficiently large value such that there is no incentive to use these time slots.

We study the scheduling objectives of minimizing the makespan and the total sum of (weighted) completion times. For the makespan objective, we show that preemptive scheduling on unrelated parallel machines, which is one of the most

general scheduling models, admits a polynomial-time algorithm in the cost-aware setting. The word 'unrelated' means that each job's processing time is machine-dependent. However, the problem of minimizing the total (weighted) completion time is considerably harder, even on a single machine. We present a polynomial-time algorithm that computes for any given sequence of jobs an optimal schedule, that is, the optimal set of time slots to be used for scheduling the jobs according to the given sequence. This result is based on dynamic programming using structural properties of optimal solutions and a potential function argument. With this algorithm we can solve the unweighted problem optimally in polynomial time, since an optimal job sequence is efficiently computable for this problem. Furthermore, we argue that for any $\varepsilon > 0$ there is an approximation algorithm with performance guarantee $4 + \varepsilon$ for the strongly NP-hard problem with individual job weights. For the weighted version, we also give a polynomial-time approximation scheme (PTAS), which is the best possible approximation result one can hope for, given the complexity of this problem. The idea for this PTAS is based on a re-interpretation of the scheduling problem within the Gantt chart representation [36] introduced for scheduling on a machine of varying speed by Megow and Verschae [64].

All mentioned results hold for the scheduling model, where all jobs are available for processing at the same time point. We also analyze the influence of having job-individual release dates. The processing of a job must not start before the release date of a job. We show that there is again a polynomial-time algorithm for makespan minimization on a single machine in the presence of release dates. However, the problem with the total completion time objective, that is, all jobs have the same weight, turns out to be NP-hard. Hence, having release dates increase the problem complexity for the min-sum objective.

## 1.2 Cost-Aware Machine Scheduling under Data Uncertainty

Scheduling aims at finding good temporal allocations of given tasks to given resources, where this allocation is usually carried out in the future. As the future is unpredictable, scheduling problems have an intrinsic data uncertainty in practice. Moreover, measurement or estimation errors within the input data add to this data uncertainty. Machines might unexpectedly break down or a job might have a much longer processing time than expected. Our scheduling models should take this issue into account.

Let us reconsider the concrete application of cloud computing. Cloud computing providers offer their customers rapid access to computing resources via the Internet and use different pricing options such as *on-demand* and *reserved instances* [1]. In the reservation option, a user pays a priori a fixed amount to reserve resources in advance, whereas on-demand instances are charged on a

(for example hourly) pay-as-used basis. Users of cloud computing services face the challenging task of choosing the best combination of pricing options when provisioning resources [17], in particular, if instances of computing jobs underlie uncertainty.

We propose the following general model for *two-stage scheduling with reservation cost under uncertainty* that captures such a resource provisioning and scheduling problem in the cloud. The model relies on a scenario-based approach, where a *scenario* is a particular realization of the uncertain input parameters. In the first stage, we are given distributional information about scheduling scenarios, and in the second stage the actual scenario is revealed. The task is to construct a schedule for the realized scenario. Using a time unit of processing in the schedule incurs some fixed cost, independent of the used capacity (number of machines), but dependent on when the time unit is reserved: low if it is reserved in the first stage, not knowing the actual scenario, and high in the second stage, given full information. Such a cost structure applies, for example, when reserving a time unit on a server gives access to all processors on this server. In the *stochastic setting*, the overall goal is to minimize total expected payment (in both stages) plus scheduling cost. In the *robust setting*, the overall goal is to minimize the maximum, over all scenarios, of payment (in both stages) plus scheduling cost. That is, we do not make use of the given distributional information in the first stage, because we aim at finding a solution that minimizes the total cost in the worst-case.

Two-stage stochastic and two-stage robust optimization with recourse are two established methodologies for optimization under data uncertainty. Many combinatorial optimization problems such as Set Cover, Network Design, Maximum Weight Matching, Shortest Path, etc. have been studied in this framework in terms of approximation algorithms [29, 34, 53, 80]. For scheduling problems in this two-stage framework, however, there are only a few approximation results known [73].

We consider both stochastic and robust versions of scheduling preemptive jobs with release dates on unrelated parallel machines and study the two objectives of minimizing the total sum of weighted completion times and the makespan. The corresponding single-stage single-scenario versions of these problems are fundamental classical scheduling problems. We give approximation algorithms with constant performance guarantees for both objectives in the stochastic and the robust model. Our results for the stochastic setting hold in the most general random model, the so-called *black-box model*, in which we have efficient access to an oracle that provides samples according to the unknown probability distribution.

## 1.3 Scheduling Maintenance Jobs in Networks

Transportation and telecommunication networks are important backbones of modern infrastructure and have been a major focus of research in combinatorial

optimization and other areas. Research on such networks usually concentrates on optimizing their usage, for example by maximizing throughput or minimizing costs. In the majority of the studied optimization models it is assumed that the network is permanently available, and our choices only consist in deciding which parts of the network to use at each point in time.

Practical transportation and telecommunication networks, however, can generally not be used non-stop. Be it due to wear-and-tear, repairs, servicing, or modernizations of the network, there are times when parts of the network are unavailable. We study how to schedule and coordinate such maintenance in different parts of the network with the objective of preserving network connectivity.

While network problems and classical scheduling problems individually are fairly well understood, the combination of both areas that results from scheduling network maintenance has only recently received some attention [9, 10, 12, 35, 67] and is theoretically hardly understood. We study the Connectivity problem, which is a fundamental problem in this context. In this problem, we aim to schedule the maintenance of edges in a network so as to preserve connectivity between two designated vertices. Given a network and the maintenance jobs with processing times and feasible time windows, we need to decide on the temporal allocation of the maintenance jobs. While maintenance on an edge is performed, the edge is not available. We distinguish between MinConnectivity, where the time in which the network is disconnected has to be minimized, and MaxConnectivity, where the time in which it is connected has to be maximized.

The Connectivity problem can be seen as a cost-aware scheduling problem. If we need to pay a certain penalty cost whenever the two distinguished vertices in our network are disconnected, we aim at coordinating the maintenance so that the total penalty cost is minimized. The scheduling cost is neglected here. Similarly, if we get a certain reward for all time points where our maintenance schedule allows connectivity, we want to coordinate the maintenance so that the total reward is maximized.

Our contributions are optimal algorithms, results on the computational complexity and approximability for different variants of the problem and different network structures. We show that the problem with preemptive maintenance jobs can be solved optimally in polynomial time in arbitrary networks. However, any restriction on the job preemption makes the problem considerably harder. Limiting the preemption to integral points in time makes the problem NP-hard and even inapproximable in the minimization version. Fully disallowing preemption only increases the complexity further; here we give strong lower bounds on the approximability. Furthermore, we give tight bounds on the *power of preemption*, that is, the maximum ratio of the values of non-preemptive and preemptive optimal solutions. We show that it is non-constant, even on simple paths. Interestingly, in such a network setting the preemptive as well as the non-preemptive problem are known to be efficiently solvable, whereas we show that mixing both leads to an NP-hard problem.

## 1.4 Outline of Thesis

In Chapter 2, we introduce basic concepts of classical scheduling theory, complexity theory, and approximation algorithms. In Chapter 3, we consider deterministic cost-aware machine scheduling problems. Chapter 4 studies stochastic and robust cost-aware machine scheduling problems and Chapter 5 focuses on maintenance scheduling in networks. Each of the Chapters 3 to 5 starts with a brief outline, followed by a formal problem definition and an overview of related work and our contributions, and ends with concluding words and a list of open problems for future research. In all three chapters, we study computational complexity of the corresponding problems and devise and analyze optimal algorithms or approximation algorithms depending on the problem complexity. As the Chapters 3 to 5 are more or less self-contained, they can be read in any order.

**Bibliographic Remark.** Parts of this thesis are already published in peer-reviewed conference proceedings and and their submission to peer-reviewed journals is in preparation. Some parts of this thesis will therefore correspond to or be identical with the following publications:

[24] L. Chen, N. Megow, R. Rischke, L. Stougie, and J. Verschae. Optimal algorithms and a PTAS for cost-aware scheduling. In Proceedings of the 40th International Symposium on Mathematical Foundations of Computer Science (MFCS), volume 9235 of LNCS, pages 211–222. Springer, 2015.

[23] L. Chen, N. Megow, R. Rischke, L. Stougie, and J. Verschae. Optimal algorithms and a PTAS for cost-aware scheduling. Journal version of [24], in preparation.

[22] L. Chen, N. Megow, R. Rischke, and L. Stougie. Stochastic and robust scheduling in the cloud. In Proceedings of the 18th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX) and 19th International Workshop on Randomization and Computation (RANDOM), volume 40 of LIPIcs, pages 175–186. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[21] L. Chen, N. Megow, R. Rischke, and L. Stougie. Stochastic and robust scheduling in the cloud. Journal version of [22], in preparation.

[2] F. Abed, L. Chen, Y. Disser, M. Groß, N. Megow, J. Meißner, A. Richter, and R. Rischke. Scheduling maintenance jobs in networks. Submitted.

Chapter 3 is based on and in part identical with [23, 24], Chapter 4 is based on and in part identical with [21,22], and Chapter 5 is based on and in part identical with [2]. Parts of Chapter 5 may also appear in the PhD thesis of A. Richter.

# 2

# Preliminaries

In this chapter, we introduce basic concepts used in scheduling, complexity, and approximation theory. In addition to that, the reader should be familiar with the fundamentals of combinatorial optimization such as graph theory; see for example [56] for an introduction into it.

**Scheduling Theory.** We already know from Chapter 1 that scheduling concerns the temporal allocation of tasks to scarce resources with the objective of optimizing some performance measure subject to certain side constraints. Graham et al. [42] introduced 1979 a *three-field-notation* to formally classify machine scheduling problems. Since then machine scheduling problems are usually represented by a triple $\alpha \,|\, \beta \,|\, \gamma$, where the $\alpha$ specifies the machine environment, the $\beta$ describes job characteristics resp. side constraints, and the $\gamma$ defines the objective function. We will consider the following typical scheduling environments in this thesis.

| | $\alpha$-FIELD |
|---|---|
| 1 | There is a single machine for processing the jobs. |
| $P$ | There are $m$ identical parallel machines. A job $j$ has processing time $p_j$ on any of them. |
| $Q$ | There are $m$ related parallel machines with different speeds. Machine $i$ has speed $s_i$. The processing time of job $j$ on machine $i$ is $p_j/s_i$. |
| $R$ | There are $m$ unrelated parallel machines, that is, machine $i$ can process job $j$ at speed $s_{ij}$. The processing time of job $j$ on machine $i$ is then $p_{ij} = p_j/s_{ij}$. |

If $\alpha \in \{P, Q, R\}$ is followed by an $m$, it means that the number of machines is fixed and not part of the input. Even if we have parallel machines, we usually assume that a job cannot be processed in parallel to itself, that is, parallelization is not allowed.

| $\beta$-FIELD | |
|---|---|
| *pmtn* | The processing of a job may be preempted and resumed later on any of the given machines at no extra cost. When *pmtn* is included in the $\beta$-field preemption is allowed, otherwise forbidden. |
| $r_j$ | Each job $j$ has a release date $r_j$, which defines the earliest moment in time where the processing of jobs $j$ can be started. If not included in the $\beta$-field, all jobs are released at the same time point. |

If there are multiple entries in the $\beta$-field, they are comma separated.

| $\gamma$-FIELD | |
|---|---|
| $C_{\max}$ | The objective is to minimize $C_{\max} := \max_j C_j$, where $C_j$ is the completion time of job $j$. |
| $\sum(w_j)C_j$ | The objective is to minimize the total sum of the (weighted) completion times. |

We refer the interested reader to the textbooks [59, 69] for a more detailed introduction into scheduling theory.

**Complexity Theory.** Cook [27] and Karp [52] laid the basis for the complexity theory in the 1970s, which aims at classifying problems in terms of their computational complexity. The basic concepts were developed for *decision problems*, that is, for problems where the output is either 'yes' or 'no'. For instance, given the scheduling problem $P \,|\, pmtn \,|\, C_{\max}$ and a positive bound $B$, the task could be to decide whether there exists a feasible schedule with makespan $B$ or not. This example also serves to illustrate the relation between decision and optimization problems. If we are able to efficiently solve the optimization problem $P \,|\, pmtn \,|\, C_{\max}$, we can verify whether the optimal makespan $C^*_{\max} \leq B$, and thus we can also efficiently solve the corresponding decision problem. On the other hand, if the decision problem is efficiently solvable, then we can use binary search to solve the corresponding optimization problem in an efficient way, given that there is one.

To present an instance $I$ of a certain decision problem $\pi$ to an algorithm for $\pi$, we need to encode the instance $I$ using a reasonable encoding scheme. The number of bits required to encode the instance $I$ is called the *size of the input instance* $I$ or short *input size*. Numerical values are usually encoded in a binary fashion.

**Definition 2.1 (Polynomial-Time Algorithm)** An algorithm for a problem is called a *polynomial-time algorithm* if the running time is bounded by a polynomial in the size of the input.

Note that this definition abstracts from the actual model of computer and the actual encoding scheme; see [38, Section 2.1] for a detailed discussion on that. Based on this definition, we can classify decision problems as follows.

The class P (polynomial-time) contains all decision problems that admit a deterministic polynomial-time algorithm, that is, problems in P are efficiently decidable. For example, the decision problem of $P\,|\,pmtn\,|\,C_{\max}$ belongs to that class [63]. A decision problem $\pi$ belongs to the class NP (non-deterministic polynomial-time), if there is a deterministic polynomial-time algorithm $A$ such that for any instance $I$ of $\pi$ the following holds: there exists a certificate $c(I)$ of polynomial size such that $A(I, c(I))$ outputs yes if and only if $I$ is a YES-instance. The decision problem of $P\,|\,pmtn\,|\,C_{\max}$ obviously also belongs to the class NP. In fact, we observe that $\mathrm{P} \subseteq \mathrm{NP}$. However, the question whether $\mathrm{P} \subset \mathrm{NP}$ or $\mathrm{P} = \mathrm{NP}$ is the most important open question in complexity theory.

There are good reasons to believe that $\mathrm{P} \subset \mathrm{NP}$. These reasons are based on the concept of *NP-completeness*.

**Definition 2.2 (Polynomial-Time Reduction)** A decision problem $\pi_1$ is *polynomial-time reducible* to another decision problem $\pi_2$, denoted by $\pi_1 \preceq \pi_2$, if there exists a polynomial-time algorithm $A$ that takes as input an instance of $\pi_1$ and outputs an instance of $\pi_2$ such that $A(I) \in \pi_2$ is a YES-instance if and only if $I \in \pi_1$ is a YES-instance.

It is rather easy to show that the binary relation '$\preceq$' is transitive, that is, if $\pi_1 \preceq \pi_2$ and $\pi_2 \preceq \pi_3$ then $\pi_1 \preceq \pi_3$.

**Definition 2.3 (NP-Completeness)** A decision problem $\pi$ is said to be *NP-complete* if $\pi \in \mathrm{NP}$ and $\pi' \preceq \pi$ for every $\pi' \in \mathrm{NP}$.

In this thesis, we will consider several NP-complete decision problems. If there is a deterministic polynomial-time algorithm for an NP-complete decision problem $\pi$, then $\mathrm{P} = \mathrm{NP}$. Despite intensive research over the past decades, no such algorithm has beed found, which gives reason to believe that $\mathrm{P} \neq \mathrm{NP}$.

Furthermore, we distinguish problems within the class of NP-complete problems as follows.

**Definition 2.4 (Pseudo-Polynomial-Time Algorithm)** An algorithm for a problem $\pi$ is called *pseudo-polynomial-time algorithm* if the running time is bounded by a polynomial in the size of the input when unary encoding is used.

This leads us to the following definition.

**Definition 2.5 (Strong and Weak NP-Completeness)** A decision problem $\pi$ is called *strongly NP-complete* if it is NP-complete even when a unary encoding scheme is used. However, a decision problem $\pi$ is said to be *weakly NP-complete* if it admits a pseudo-polynomial-time algorithm.

For example, the decision problem for $P\,|\,|\,C_{\max}$ is known to be strongly NP-hard, whereas the decision problem for $Pm\,|\,|\,C_{\max}$ is known to be weakly NP-hard [37, 38].

As a last concept, we introduce *NP-hardness* to also classify optimization problems.

**Definition 2.6 (NP-Hardness)** An optimization problem $\pi$ is called *NP-hard* if the corresponding decision problem is NP-complete.

The notions of strong and weak NP-hardness follow according to the above notions of strong and weak NP-completeness in Definition 2.5.

We refer the interested reader to [38, 56] for a more detailed introduction into complexity theory.

**Approximation Theory.** As we probably cannot hope for an NP-hard problem to have an algorithm that returns for any instance of the problem an optimal solution in polynomial time, we need to relax at least one of the requirements. In this thesis, we shall relax the optimality requirement, that is, we focus on approximation algorithms.

**Definition 2.7 (Approximation Algorithm)** An *$\alpha$-approximation algorithm* for an optimization problem $\pi$ is a polynomial-time algorithm that returns for any instance of $\pi$ a feasible solution whose objective function value is within a factor of $\alpha$ of the optimal value.

The factor $\alpha$ is called *performance guarantee* of the approximation algorithm, where we follow the convention that $\alpha \geq 1$. That is, an $\alpha$-approximation algorithm $A$ returns for any instance $I$ of a minimization problem a feasible solution of value $A(I) \leq \alpha \cdot OPT(I)$, whereas for a maximization problem we have $A(I) \geq 1/\alpha \cdot OPT(I)$, where $OPT(I)$ is the optimal value for instance $I$.

We are interested in best possible approximation algorithms and for this we introduce the notion of polynomial-time approximation schemes.

**Definition 2.8 (PTAS)** A *polynomial-time approximation scheme* (PTAS) is a family of algorithms $\{A_\varepsilon\}$ such that for any $\varepsilon > 0$ the algorithm $A_\varepsilon$ is a $(1+\varepsilon)$-approximation algorithm.

Note that according to this definition the running time may depend arbitrarily on $1/\varepsilon$. This is taken care of in the next definition.

**Definition 2.9 (FPTAS)** A *fully polynomial-time approximation scheme* (FPTAS) is a PTAS such that the running time of $A_\varepsilon$ is bounded by a polynomial in the input size and $1/\varepsilon$ for any $\varepsilon > 0$.

It is a well-known fact that a strongly NP-hard problem does not admit an FPTAS, unless P = NP [38]. For example, the strongly NP-hard scheduling problem $P\,|\,|\,C_{\max}$ admits a PTAS, whereas for the weakly NP-hard version $Pm\,|\,|\,C_{\max}$ an FPTAS is known [41, 45].

We refer the interested reader to the textbook [75] for a comprehensive discussion of techniques for designing and analyzing approximation algorithms.

# 3

# Deterministic Cost-Aware Machine Scheduling

In this chapter, we address a natural generalization of classical scheduling problems in which occupying a time slot incurs certain cost that may vary over time and which must be paid in addition to the actual scheduling cost. This framework has been proposed recently by Wan and Qi [83] and Kulkarni and Munagala [57]. Based on the problem definition in Section 3.1, we give an overview of related work and our contribution in Section 3.2. Then, in Section 3.3, we show that unrelated machine scheduling so as to minimize the makespan is polynomial-time solvable when taking the cost-awareness into account. In Section 3.4, we present a polynomial-time algorithm that computes for any given sequence of jobs an optimal set of time slots to be used for scheduling the jobs according to the given sequence. Based on this, we obtain approximation results for minimizing the total (weighted) completion time on a single machine. The weighted problem variant is again focus of Section 3.5, for which we give a best possible approximation result. Thereafter, in Section 3.6, we analyze the influence of release dates on the problem complexity. At the end of this chapter, we discuss open problems.

## 3.1 Problem Definition

We first describe the underlying classical scheduling problems. We are given a set of jobs $J := \{1, \dots, n\}$ where every job $j \in J$ has given a processing time $p_j \in \mathbb{N}$ and possibly a weight $w_j \in \mathbb{Q}_{\geq 0}$. The task is to find a preemptive schedule on a single machine such that the total (weighted) completion time, $\sum_{j \in J} w_j C_j$, is minimized. Here $C_j$ denotes the completion time of job $j$. In the standard scheduling notation, this problem is denoted as $1 \,|\, pmtn \,|\, \sum(w_j)C_j$. We also consider makespan minimization on unrelated machines, typically denoted as $R \,|\, pmtn \,|\, C_{\max}$. Here we are given a set of machines $M$, and each job $j \in J$ has an individual processing time $p_{ij} \in \mathbb{N}$ for running on machine $i \in M$. The task is to find a preemptive schedule that minimizes the makespan, that is, the completion time of the latest job.

In this chapter, we consider a generalization of these scheduling problems within a time-varying reservation cost model. We are given a cost function $e : \mathbb{R}_{\geq 0} \to \mathbb{Q}_{\geq 0}$, where $e(t)$ denotes the reservation cost for processing job(s) at time $t$. We assume that $e$ is piecewise constant with given breakpoints at integral time points. More formally, we assume that time is discretized into unit-size time slots, and the time horizon is partitioned into given intervals $I_k = [s_k, d_k)$ with $s_k, d_k \in \mathbb{N}$, $k = 1, \ldots, K$, within which unit-size time slots have the same *unit reservation cost* $e_k \in \mathbb{Q}_{\geq 0}$. To ensure feasibility, let $d_K \geq \sum_{j \in J} \min_{i \in M} p_{ij}$.

Given a schedule $\mathcal{S}$, let $y(t)$ be a binary variable indicating if any processing is assigned to time slot $[t, t+1)$. The *reservation cost* in $\mathcal{S}$ is $E(\mathcal{S}) = \sum_t e(t)y(t)$. That means, for any time unit that is used in $\mathcal{S}$ we pay the full unit reservation cost, even if the unit is only partially used. We also emphasize that in case of multiple machines, a reserved time slot can be used by all machines. This models applications in which reserving a time unit on a server gives access to all processors on this server.

The overall objective now is to find a schedule that minimizes the scheduling objective, $C_{\max}$ resp. $\sum_{j \in J} w_j C_j$, *plus* the reservation cost $E$. We refer to the resulting problems as $R \,|\, pmtn \,|\, C_{\max} + E$ and $1 \,|\, pmtn \,|\, \sum w_j C_j + E$. We remark that the results in this chapter also hold if we minimize a convex combination of the scheduling and reservation cost.

## 3.2 Related Work and Contributions

**Related Work.** Scheduling with time-varying reservation cost (aka variable time slot cost) has been studied explicitly by Wan and Qi [83], Kulkarni and Munagala [57] and Zhao, Qi, and Li [85]. In the seminal paper, Wan and Qi [83] consider several *non-preemptive* single machine problems, which are polynomial-time solvable in the classical setting, such as minimizing the total completion time, lateness, and total tardiness, or maximizing the weighted number of on-time jobs. These problems are shown to be strongly NP-hard when taking reservation cost into account, while efficient algorithms exist for restricted reservation cost functions. In particular, the problem $1 \,|\, | \sum C_j + E$ is strongly NP-hard, and it is efficiently solvable when the reservation cost is increasing or convex non-increasing [83].

Zhao, Qi, and Li [85] address an open question in [83], namely the complexity of non-preemptive single machine scheduling so as to minimize the total weighted completion time plus the reservation cost, where the reservation cost function is non-increasing over time. They show that for an arbitrary non-increasing reservation cost functions the problem is strongly NP-hard, whereas special non-increasing functions for the reservation cost admit efficient algorithms.

Kulkarni and Munagala [57] focus on *online* flow-time minimization using resource augmentation. Their main result is a scalable algorithm that obtains a

constant performance guarantee when the machine speed is increased by a constant factor and there are only two distinct unit reservation costs. They also show that, in this online setting, for arbitrary many distinct unit reservation costs there is no constant speedup-factor that allows for a constant approximate solution.

In this chapter, we study the simpler, but not yet well understood, offline problem without release dates. For this problem, Kulkarni and Munagala [57] announce the following results: a pseudo-polynomial $(4+\varepsilon)$-approximation for $1\,|\,pmtn\,|\,\sum w_jC_j + E$, which gives an optimal solution in case that all weights are equal, and a constant approximation in quasi-polynomial time for a constant number of distinct reservation costs or when using a machine that is processing jobs faster by a constant factor.

The general concept of taking into consideration additional (time-dependent) cost for resource utilization when scheduling has been implemented differently in other models. We mention the area of energy-aware scheduling, where the energy consumption is taken into account (see [4] for an overview), or scheduling with generalized non-decreasing (completion-) time dependent cost functions, such as minimizing $\sum_j w_j f(C_j)$, e.g. [33, 46, 64], or even more general job-individual cost functions $\sum_j f_j(C_j)$, e.g. [7, 25, 47, 66]. Our model differs fundamentally since our cost function may decrease with time. In fact, delaying the processing in favor of cheaper time slots may decrease the overall cost. This is not the case in the above-mentioned models. Thus, in our framework we have the additional dimension in decision-making of selecting the time slots that shall be reserved.

Nevertheless, there is some similarity between our model and scheduling on a machine of varying speed. Notice that the latter problem (with $\sum_j w_jC_j$ as objective function) can be reformulated as minimizing $\sum_j w_j f(C_j)$ on a single machine with constant speed. Interestingly, the independently studied problem of scheduling with non-availability periods, see e.g. the survey [59], is a special case of both, the time-varying speed and the time-varying reservation cost model. Indeed, machine non/availability can be expressed either by 0/1-speed or equivalently by $\infty/0$ unit reservation cost. Results shown in this context imply that our problem $1\,|\,pmtn\,|\,\sum w_jC_j + E$ is strongly NP-hard, even if there are only two distinct unit reservation costs [84].

**Our contribution.** We present new optimal algorithms and best-possible approximation results for a generalization of standard scheduling problems to a framework with time-varying reservation cost.

Firstly, we give an optimal polynomial-time algorithm for the problem $R\,|\,pmtn\,|\,C_{\max} + E$ (Section 3.3). We design a procedure that selects the optimal time slots to be reserved, given that we knew the optimal *number* of time slots. This optimal number can be determined by solving the scheduling problem *without* reservation cost, which can be done optimally in polynomial time by solving a linear program [58].

Our main results concern single-machine scheduling to minimize the to-

tal (weighted) completion time (Section 3.4). We present an algorithm that computes for a given ordered set of jobs an optimal choice of time slots to be used for scheduling (Section 3.4). We derive this by first showing structural properties of an optimal schedule, which we then exploit together with a properly chosen potential function in a dynamic program yielding polynomial running time. Based on this algorithm, we show that the unweighted problem $1\,|\,pmtn\,|\,\sum C_j + E$ can be solved in polynomial time and that there is a $(4+\varepsilon)$-approximation algorithm for the weighted version $1\,|\,pmtn\,|\,\sum w_j C_j + E$. A pseudo-polynomial $(4+\varepsilon)$-approximation was given in [57]. While pseudo-polynomial time algorithms are rather easy to derive, it is remarkable that our DP's running time is polynomial in the input size, in particular, independent of $d_K$.

Finally, we design for the strongly NP-hard weighted problem variant (Section 3.5) a polynomial-time algorithm that computes for any fixed $\varepsilon$ a $(1+\varepsilon)$-approximate schedule for $1\,|\,pmtn\,|\,\sum w_j C_j + E$, i.e., a PTAS. Unless P = NP, our algorithm is best possible, since the problem is strongly NP-hard even if there are only two different reservation costs [84].

Our approach is inspired by a recent PTAS for scheduling on a machine of varying speed [64] and it uses some of its properties. As discussed above, there is no formal mathematical relation known between these two seemingly related problems which allows to directly apply the result from [64]. The key is a dual view on scheduling: instead of directly constructing a schedule in the time-dimension, we first construct a dual scheduling solution in the weight-dimension which has a one-to-one correspondence to a true schedule. We design an exponential-time dynamic programming algorithm which can be trimmed to polynomial time using techniques known for scheduling with varying speed [64].

For both the makespan and the min-sum problem, job preemption is crucial for obtaining worst-case bounds. For non-preemptive scheduling, a straightforward reduction from 2-Partition shows that no approximation is possible, unless P = NP, even if there are only two different reservation costs, 0 and $\infty$.

We remark that in general it is not clear that a schedule can be encoded polynomially in the input. However, for our completion-time based minimization objective, it is easy to observe that if an algorithm reserves $p$ unit-size time slots in an interval of equal cost, then it reserves the first $p$ slots within this interval, which simplifies the structure and the output of an optimal solution.

We also analyze the influence of having job-individual release dates on the problem complexity in Section 3.6. The processing of a job must not start before the release date of a job. We show that there is again a polynomial-time algorithm for makespan minimization on a single machine in the presence of release dates (Section 3.6.1) and on unrelated parallel machines with a slightly relaxed cost model (Section 3.6.2). However, the problem with the total completion time objective, that is, all jobs have the same weight, turns out to be NP-hard (Section 3.6.3). Hence, having release dates increases the complexity for the min-sum objective.

## 3.3 Minimizing the Makespan on Unrelated Machines

The standard scheduling problem without reservation $R\,|\,pmtn\,|\,C_{\max}$ can be solved optimally in polynomial time by solving a linear program as was shown by Lawler and Labetoulle [58]. We show that the problem complexity does not increase significantly when taking into account time-varying reservation cost.

Consider the preemptive makespan minimization problem with reservation cost. Recall that we can use every machine in a reserved time slot and pay only once. Thus, it is sufficient to find an optimal reservation decision for solving this problem, because we can use the polynomial-time algorithm in [58] to find the optimal schedule within these slots.

**Observation 3.1** Given the set of time slots reserved in an optimal solution, we can compute an optimal schedule in polynomial time.

Given an instance of our problem, let $Z$ be the optimal makespan of the relaxed problem *without* reservation cost. Notice that $Z$ is not necessarily integral. To determine an optimal reservation decision, we use the following observation.

**Observation 3.2** Given an optimal makespan $C^*_{\max}$ for $R\,|\,pmtn\,|\,C_{\max}+E$, an optimal schedule reserves the $\lceil Z\rceil$ cheapest slots before $\lceil C^*_{\max}\rceil$.

Note that we must pay full reservation cost for a used time slot, no matter how much it is utilized, and so does an optimal solution. In particular, this holds for the last reserved slot. Hence, it remains to compute an optimal value $C^* := \lceil C^*_{\max}\rceil$ which we do by the following procedure.

We compute for every interval $I_k = [s_k, d_k)$, $k = 1, \ldots, K$, an optimal point in time for $C^*$ assuming that $C^* \in I_k$. Hereby we restrict to relevant intervals $I_k$ which allow for a feasible schedule, i.e., $s_k \geq \lceil Z\rceil$. For a relevant interval $I_k$, we let $C^* = s_k$ and reserve the $\lceil Z\rceil$ cheapest time slots before $C^*$, which is optimal by Observation 3.2. Notice that any reserved time slot of cost $e$ such that $e > e_k + 1$ can be replaced by a time slot from $I_k$ leading to a solution of less total cost. Thus, if there is no such time slot, then $s_k$ is the best choice for $C^*$ in $I_k$. Suppose there is such a time slot that could be replaced. Let $R \subseteq \{1, \ldots, k-1\}$ be the index set of intervals that contain at least one reserved slot. We define $I_\ell$ to be the interval with $e_\ell = \max_{h\in R} e_h$ and denote by $r_h$ the number of reserved time slots in $I_h$. Replace $\min\{r_\ell, d_k - s_k - r_k\}$ reserved slots from $I_\ell$ by slots from $I_k$ and update $R$, $I_\ell$ and $r_k$. This continues until $e_\ell \leq e_k + 1$ or the interval $I_k$ is completely reserved, i.e., $r_k = d_k - s_k$. This operation takes at most $O(K)$ computer operations per interval to compute the best $C^*$-value in that interval. It yields the following theorem.

**Theorem 3.3** The scheduling problem $R\,|\,pmtn\,|\,C_{\max}+E$ can be solved in polynomial time in the order of $O(K^2)$ plus the running time for solving $R\,|\,pmtn\,|\,C_{\max}$ without reservation cost [58].

## 3.4 Minimizing Total Weighted Completion Time on a Single Machine

In this section, we consider the problem $1\,|\,pmtn\,|\,\sum(w_j)C_j + E$. We design an algorithm that computes, for a given (not necessarily optimal) scheduling sequence $\sigma$, an optimal reservation decision for $\sigma$. We firstly identify structural properties of an optimal schedule, which we then exploit in a dynamic program. Based on this algorithm, we show that the unweighted problem $1\,|\,pmtn\,|\,\sum C_j + E$ can be solved optimally in polynomial time and that there is a $(4+\varepsilon)$-approximation algorithm for the weighted problem $1\,|\,pmtn\,|\,\sum w_j C_j + E$.

In principle, an optimal schedule may preempt jobs at fractional time points. However, since time slots can only be reserved entirely, any reasonable schedule uses the reserved slots entirely as long as there are unprocessed jobs. The following lemma shows that this is also true if we omit the requirement that time slots must be reserved entirely. (For the makespan problem considered in Section 3.3 this is not true.)

**Lemma 3.4** There is an optimal schedule $\mathcal{S}^*$ in which all reserved time slots are entirely reserved and jobs are preempted only at integral points in time.

***Proof.*** Consider an optimal schedule $\mathcal{S}$ and let $C$ be its total cost (scheduling and reservation cost). W.l.o.g. we may assume that $\mathcal{S}$ processes jobs one by one in a fixed priority order. That means, a job may be preempted but only for keeping the machine idle and not for processing another job. Suppose that there is a time slot $[t_0, t_0+1)$ that is only partially reserved, say from $t_0$ to $t_0+\lambda$ with $\lambda < 1$. We consider the first such slot in $\mathcal{S}$. The integrality of the processing times $p_j$ implies that there must exist another slot $[t_1, t_1+1)$ that is also fractionally reserved, say from $t_1$ to $t_1+\mu$ with $\mu < 1$. We consider the first such slot $[t_1, t_1+1)$ after $[t_0, t_0+1)$. Let $e(t)$ be the reservation cost for slot $[t, t+1)$ if we completely reserved it. We first observe that $e(t_1) < e(t_0)$, otherwise $\mathcal{S}$ is not optimal. We will show that the number of fractionally reserved slots can be reduced without increasing the total cost. Iteratively applying this argument until all slots are completely reserved shows the statement.

We introduce the notion of *$\delta$-right-shift* and *$\delta$-left-shift*. Performing a *$\delta$-right-shift* on the partial schedule between $t_0$ and $t_1+1$ means that we decrease the current amount of reservation for $[t_0, t_0+1)$ by $\delta$ while we increase the current amount of reservation for $[t_1, t_1+1)$ by $\delta$ and move the partial schedule as a whole to the right, that is, we reschedule workload without changing the processing order of jobs and by using only reserved slots. Similarly, a *$\delta$-left-shift* on the partial schedule between $t_0$ and $t_1+1$ means that we increase the current amount of reservation for $[t_0, t_0+1)$ by $\delta$ while we decrease the current amount of reservation for $[t_1, t_1+1)$ by $\delta$ and move the partial schedule as a whole to the left.

Let $J^c := \{j \in J : C_j(\mathcal{S}) \in (t_0, t_1+1)\}$. Consider any job $j \in J^c$. Observe that $C_j(\mathcal{S}) = t+1-\lambda$ for some integral time point $t \in (t_0, t_1+1)$, because $\mathcal{S}$

processes jobs in a fixed priority order, the processing times are integral, and $[t_0, t_0 + 1)$ is the first fractionally reserved slot. Therefore, a $\delta$-right-shift with $0 < \delta \leq \min\{\lambda, 1 - \mu\}$ of the partial schedule between $t_0$ and $t_1 + 1$ increases the completion time of every job $j \in J^c$ by $\delta$ and reduces the reservation cost by $\delta\,(e(t_1) - e(t_0))$. In total, the cost of $\mathcal{S}$ after performing a $\delta$-right-shift is

$$C + \delta \cdot \left( \sum_{j \in J^c} w_j - e(t_0) + e(t_1) \right).$$

On the other hand, a $\delta$-left-shift with $0 < \delta \leq \min\{\mu, 1 - \lambda\}$ of the partial schedule between $t_0$ and $t_1 + 1$ increases the reservation cost by $\delta\,(e(t_0) - e(t_1))$ and decreases the completion time of every job $j \in J^c$ by $\delta$ if $\delta < 1 - \lambda$. If, however, $\delta = (1 - \lambda)$, then the completion time of a job $j \in J^c$ might decrease by more than $1 - \lambda$, as the reserved slots in $[t_0, t_1 + 1)$ are not necessarily consecutive. In total, the cost of $\mathcal{S}$ after performing a $\delta$-left-shift is at most

$$C - \delta \cdot \left( \sum_{j \in J^c} w_j - e(t_0) + e(t_1) \right).$$

Hence, if $\sum_{j \in J^c} w_j - e(t_0) + e(t_1) > 0$, we perform a $(1 - \lambda)$-left-shift. Otherwise, we check whether a $(1 - \lambda)$-left-shift leads to less total cost than a $\min\{\lambda, 1 - \mu\}$-right-shift. If yes, then we perform the left-shift, otherwise the right-shift. This reduces the number of fractionally reserved slots by not increasing the total cost of $\mathcal{S}$. □

In the following, we assume that we are given a (not necessarily optimal) sequence of jobs, $\sigma = (1, \ldots, n)$, in which the jobs must be processed. We want to characterize an optimal schedule $\mathcal{S}^*$ for $\sigma$, that is, in particular the optimal choice of time slots for scheduling $\sigma$. We first split $\mathcal{S}^*$ into smaller sub-schedules, for which we introduce the concept of a *split point*.

**Definition 3.5 (Split Point)** Consider an optimal schedule $\mathcal{S}^*$ and the set of potential split points $\mathcal{P} := \bigcup_{k=1}^{K} \{s_k, s_k + 1\} \cup \{d_K\}$. Let $S_j$ and $C_j$ denote the start time and completion time of job $j$, respectively. We call a time point $t \in \mathcal{P}$ a split point for $\mathcal{S}^*$ if all jobs that start before $t$ also finish their processing not later than $t$, i.e., if $\{j \in J : S_j < t\} = \{j \in J : C_j \leq t\}$.

Given an optimal schedule $\mathcal{S}^*$, let $0 = \tau_1 < \tau_2 < \cdots < \tau_\ell = d_K$ be the *maximal* sequence of split points of $\mathcal{S}^*$, i.e. the sequence containing all split points of $\mathcal{S}^*$. We denote the interval between two consecutive split points $\tau_x$ and $\tau_{x+1}$ as *region* $R_x^{\mathcal{S}^*} := [\tau_x, \tau_{x+1})$, for $x = 1, \ldots, \ell - 1$.

Consider now any region $R_x^{\mathcal{S}^*}$ for an optimal schedule $\mathcal{S}^*$ with $x \in \{1, \ldots, \ell - 1\}$ and let $J_x^{\mathcal{S}^*} := \left\{ j \in J : S_j \in R_x^{\mathcal{S}^*} \right\}$. Note that $J_x^{\mathcal{S}^*}$ might be empty.

Among all optimal schedules we shall consider an optimal solution $\mathcal{S}^*$ that minimizes the value $\sum_{t=0}^{d_K-1} t \cdot y(t)$, where $y(t)$ is a binary variable that indicates if time slot $[t, t+1)$ is reserved or not.

We observe that any job $j$ completing at the beginning of a cost interval, $C_j = s_k \in R_x^{\mathcal{S}^*}$ or $C_j = s_k + 1 \in R_x^{\mathcal{S}^*}$, would make $s_k$ resp. $s_k + 1$ a split point. Thus, no such job can exist.

**Observation 3.6** There is no job $j \in J_x^{\mathcal{S}^*}$ with $C_j \in R_x^{\mathcal{S}^*} \cap \mathcal{P}$.

We say that an interval $I_k$ is *partially reserved* if at least one slot in $I_k$ is reserved, but not all.

**Lemma 3.7** There exists an optimal schedule $\mathcal{S}^*$ in which at most one interval is partially reserved in $R_x^{\mathcal{S}^*}$.

***Proof.*** By contradiction, suppose that there is more than one partially reserved interval in $R_x^{\mathcal{S}^*}$. Consider any two such intervals $I_h$ and $I_{h'}$ with $h < h'$, and all intermediate intervals reserved entirely or not at all. Let $[t_h, t_h + 1)$ and $[t_{h'}, t_{h'} + 1)$ be the last reserved time slot in $I_h$ and $I_{h'}$, respectively. If we reserve $[t_{h'} + 1, t_{h'} + 2)$ instead of $[t_h, t_h + 1)$, then the difference in cost is $\delta_1 := e_{h'} - e_h + \sum_{j \in J'} w_j$ with $J' := \left\{ j \in J : C_j \in \bigcup_{k=h+1}^{h'} I_k \right\}$, because all jobs in $J'$ are delayed by exactly one time unit. This is true since by Observation 3.6 no job finishes at $d_k = s_{k+1}$ for any $k$. If we reserve $[t_h + 1, t_h + 2)$ instead of $[t_{h'}, t_{h'} + 1)$, then the difference in cost is $\delta_2 := e_h - e_{h'} - \sum_{j \in J'} w_j$, again using Observation 3.6 to assert that no job finishes at $s_k + 1$ for any $h + 1 \leq k \leq h'$. Since $\delta_1 = -\delta_2$ and $\mathcal{S}^*$ is an optimal schedule, it must hold that $\delta_1 = \delta_2 = 0$. This, however, implies that there is another optimal schedule with earlier used time slots which contradicts our assumption that $\mathcal{S}^*$ minimizes the value $\sum_{t=0}^{d_K-1} t \cdot y(t)$. □

We are now ready to bound the unit reservation cost spent for jobs in $J_x^{\mathcal{S}^*}$. Let $e_{\max}^j$ be the maximum unit reservation cost spent for job $j$ in $\mathcal{S}^*$. Furthermore, let $\Delta_x := \max_{j \in J_x^{\mathcal{S}^*}} (e_{\max}^j + \sum_{j' < j} w_{j'})$ and let $j_x$ be the last job (according to sequence $\sigma$) that achieves $\Delta_x$. Suppose, there are $b \geq 0$ jobs before and $a \geq 0$ jobs after job $j_x$ in $J_x^{\mathcal{S}^*}$. The following lemma gives for every job $j \in J_x^{\mathcal{S}^*} \setminus \{j_x\}$ an upper bound on the unit reservation cost spent in the interval $[S_j, C_j)$.

**Lemma 3.8** Consider an optimal schedule $\mathcal{S}^*$. For any job $j \in J_x^{\mathcal{S}^*} \setminus \{j_x\}$ a slot $[t, t+1) \in [S_j, C_j)$ is reserved if and only if the cost of $[t, t+1)$ satisfies the upper bound given in the table below.

| $j_x - b$ | $\dots$ | $j_x - 1$ | $j_x + 1$ | $\dots$ | $j_x + a$ |
|---|---|---|---|---|---|
| $\leq e_{\max}^{j_x} + \sum_{j'=j_x-b}^{j_x-1} w_{j'}$ | $\dots$ | $\leq e_{\max}^{j_x} + w_{j_x-1}$ | $< e_{\max}^{j_x} - w_{j_x}$ | $\dots$ | $< e_{\max}^{j_x} - \sum_{j'=j_x}^{j_x+a-1} w_{j'}$ |

***Proof.*** Consider any job $j := j_x - \ell$ with $0 < \ell \leq b$. Suppose there is a job $j$ for which a slot is reserved with cost $e^j_{\max} > e^{j_x}_{\max} + \sum_{j'=j}^{j_x-1} w_{j'}$. Then $e^j_{\max} + \sum_{j'<j} w_{j'} > e^{j_x}_{\max} + \sum_{j'<j_x} w_{j'}$, which is a contradiction to the definition of job $j_x$. Thus, $e^j_{\max} \leq e^{j_x}_{\max} + \sum_{j'=j}^{j_x-1} w_{j'}$.

Now suppose that there is a slot $[t, t+1) \in [S_j, C_j)$ with cost $e(t) \leq e^{j_x}_{\max} + \sum_{j'=j}^{j_x-1} w_{j'}$ that is not reserved. There must be a slot $[t', t'+1) \in [S_{j_x}, C_{j_x})$ with cost exactly $e^{j_x}_{\max}$. If we reserve slot $[t, t+1)$ instead of $[t', t'+1)$, then the difference in cost is non-positive, because the completion times of at least $\ell$ jobs ($j = j_x - \ell, \ldots, j_x - 1$ and maybe also $j_x$) decrease by one. This contradicts either the optimality of $\mathcal{S}^*$ or our assumption that $\mathcal{S}^*$ minimizes $\sum_{t=0}^{d_K-1} t \cdot y(t)$.

The proof of the statement for any job $j_x + \ell$ with $0 < \ell \leq a$ follows a similar argument, but now using the fact that for every job $j := j_x + \ell$ we have $e^j_{\max} < e^{j_x}_{\max} - \sum_{j'=j_x}^{j-1} w_{j'}$, because $j_x$ was the last job with $e^j_{\max} + \sum_{j'<j} w_{j'} = \Delta_x$. □

To construct an optimal sub-schedule, we need the following two lemmata.

**Lemma 3.9** Let $[t', t'+1) \in [S_{j_x}, C_{j_x})$ be the last time slot with cost $e^{j_x}_{\max}$ that is used by job $j_x$. If there is a partially reserved interval $I_k$ in $R_x^{\mathcal{S}^*}$, then either (i) $I_k$ is not the last interval of $R_x^{\mathcal{S}^*}$ and $I_k$ contains $[t', t'+1)$ as its last reserved time slot or (ii) $I_k$ is the last interval of $R_x^{\mathcal{S}^*}$.

***Proof.*** There cannot be a partially reserved interval in $R_x^{\mathcal{S}^*}$ before $S_{j_x}$, as otherwise this would contradict Lemma 3.8 or the maximality of our sequence of split points. If there exists a partially reserved interval $I_k$ entirely after $C_{j_x}$ in $R_x^{\mathcal{S}^*}$, then it can only be the last interval in $R_x^{\mathcal{S}^*}$. Otherwise, the last job processed in $I_k$ creates a contradiction to Lemma 3.8 or to the maximality of our sequence of split points.

Now suppose there is a partially reserved interval $I_k$ that intersects with $[S_{j_x}, C_{j_x})$. Clearly (i) may occur, in which case, by Lemma 3.7, (ii) will not occur. Thus, suppose (i) does not occur. Then $I_k$ is the last interval of $R_x^{\mathcal{S}^*}$ or $I_k$ does not contain $[t', t'+1)$ as its last reserved time slot. If $I_k$ is the last interval of $R_x^{\mathcal{S}^*}$, we are done. Thus, suppose $I_k$ does not contain $[t', t'+1)$ as its last reserved time slot. Then $I_k$ clearly has $e_k \leq e^{j_x}_{\max}$. Therefore, $I_k$ has to be after the interval that contains $[t', t'+1)$, otherwise $\mathcal{S}^*$ cannot be optimal. By definition of $[t', t'+1)$, it must be that $e_k < e^{j_x}_{\max}$. Hence, $C_{j_x} \in I_k$ since otherwise reserving an additional slot in $I_k$ instead of $[t', t'+1)$ decreases the reservation cost without affecting the completion time of $j_x$, which contradicts the optimality of $\mathcal{S}^*$. Since $[t', t'+1)$ is, by assumption, not the last reserved slot in $I_k$, there must be some other job than $j_x$ starting in $I_k$ after $C_{j_x}$, otherwise $I_k$ must be the last interval of $R_x^{\mathcal{S}^*}$. Consider any such job. By Lemma 3.8, all remaining time slots of $I_k$ need to be reserved until completion. Thus, either all slots of $I_k$ will be reserved, contradicting that $I_k$ was only partially reserved, or all jobs starting in $I_k$ complete in $I_k$, whence $I_k$ is the last interval of $R_x^{\mathcal{S}^*}$. □

**Lemma 3.10** Let $[t', t'+1) \in [S_{j_x}, C_{j_x})$ be the last time slot with cost $e^{j_x}_{\max}$ that is used by job $j_x$. There exists an optimal solution $\mathcal{S}^*$ such that if there is a partially reserved interval $I_k$ in $R_x^{\mathcal{S}^*}$ and it is the last one in $R_x^{\mathcal{S}^*}$, then there is no slot $[t, t+1) \in [S_{j_x}, C_{j_x})$ with cost at most $e^{j_x}_{\max}$ that is not reserved.

***Proof.*** Let $j_x + \ell$ with $\ell \in \{0, \dots, a\}$ be the last job processed in the partially reserved interval $I_k$. Suppose there is a time slot $[t, t+1) \in [S_{j_x}, C_{j_x})$ with cost at most $e^{j_x}_{\max}$ that is not reserved. If we reserve $[t, t+1)$ instead of the last reserved slot in $I_k$, then the difference in cost is $\delta_1 := e(t) - e_k - \sum_{j=j_x}^{j_x+\ell} w_j$. If we reserve one additional slot in $I_k$ instead of $[t', t'+1)$, then the difference in cost is $\delta_2 := e_k - e^{j_x}_{\max} + \sum_{j=j_x}^{j_x+\ell} w_j$. We consider an optimal schedule $\mathcal{S}^*$, thus $\delta_1 \geq 0$ and $\delta_2 \geq 0$ which implies that $\delta_1 + \delta_2 = e(t) - e^{j_x}_{\max} \geq 0$. This is a contradiction if $e(t) < e^{j_x}_{\max}$. If $e(t) = e^{j_x}_{\max}$, then $\delta_1 = -\delta_2 = 0$, because we consider an optimal schedule $\mathcal{S}^*$. This, however, contradicts our assumption that $\mathcal{S}^*$ minimizes the value $\sum_{t=0}^{d_K-1} t \cdot y(t)$. □

We now show how to construct an optimal partial schedule for a given ordered job set in a given region in polynomial time.

**Lemma 3.11** Given a region $R_x$ and an ordered job set $J_x$, we can construct in polynomial time an optimal schedule for $J_x$ within the region $R_x$, which does not contain any other split point than $\tau_x$ and $\tau_{x+1}$, the boundaries of $R_x$.

***Proof.*** Given $R_x$ and $J_x$, we guess the optimal combination $(j_x, e^{j_x}_{\max})$, i.e., we enumerate over all $nK$ combinations and choose eventually the best solution.

We firstly assume that a partially reserved interval exists and it is the last one in $R_x$ (case (ii) in Lemma 3.9). Based on the characterization in Lemma 3.8 we find in polynomial time the slots to be reserved for the jobs $j_x - b, \dots, j_x - 1$. This defines $C_{j_x-b}, \dots, C_{j_x-1}$. Then starting job $j_x$ at time $C_{j_x-1}$, we check intervals in the order given and reserve as much as needed of each next interval $I_h$ if and only if $e_h \leq e^{j_x}_{\max}$, until a total of $p_{j_x}$ time slots have been reserved for processing $j_x$. Lemma 3.10 justifies to do that. This yields a completion time $C_{j_x}$. Starting at $C_{j_x}$, we use again Lemma 3.8 to find in polynomial time the slots to be reserved for processing the jobs $j_x + 1, \dots, j_x + a$. This gives $C_{j_x+1}, \dots, C_{j_x+a}$.

Now we assume that there is no partially reserved interval or we are in case (i) of Lemma 3.9. Similar to the case above, we find in polynomial time the slots that $\mathcal{S}^*$ reserves for the jobs $j_x - b, \dots, j_x - 1$ based on Lemma 3.8. This defines $C_{j_x-b}, \dots, C_{j_x-1}$. To find the slots to be reserved for the jobs $j_x + 1, \dots, j_x + a$, in this case, we start at the end of $R_x$ and go backwards in time. We can start at the end of $R_x$ because in this case the last interval of $R_x$ is fully reserved. This gives $C_{j_x+1}, \dots, C_{j_x+a}$. Job $j_x$ is thus to be scheduled in $[C_{j_x-1}, S_{j_x+1})$. In order to find the right slots for $j_x$ we solve a makespan problem in the interval $[C_{j_x-1}, S_{j_x+1})$, which can be done in polynomial time (Theorem 3.3) and gives a solution that cannot be worse than what an optimal schedule $\mathcal{S}^*$ does.

If anywhere in both cases the reserved intervals cannot be made sufficient for processing the job(s) for which they are intended, or if scheduling the jobs in the reserved intervals creates any intermediate split point, then this $(j_x, e_{\max}^{j_x})$-combination is rejected. Hence, we have computed the optimal schedules over all $nK$ combinations of $(j_x, e_{\max}^{j_x})$ and over both cases of Lemma 3.9 concerning the position of the partially reserved interval. We choose the schedule with minimum total cost and return it with its value. This completes the proof. □

Now we are ready to prove our main theorem.

**Theorem 3.12** Given an instance of $1\,|\,pmtn\,|\,\sum w_jC_j + E$ and an arbitrary processing sequence of jobs $\sigma$, we can compute an optimal reservation decision for $\sigma$ in polynomial time.

***Proof.*** We give a dynamic program. We define a state for every possible potential split point $t \in \mathcal{P}$. By definition, there are $2K + 1$ of them. A state also includes the set of jobs processed until time $t$. Given the sequence $\sigma$, this job set can be uniquely identified by the index of the last job, say $j$, that finished by time $t$. By relabeling the job set $J$, we can assume w.l.o.g. that $\sigma = (1, \ldots, n)$.

For each state $(j, t)$ we compute and store recursively the optimal scheduling cost plus reservation cost $Z(j, t)$ by

$$Z(j,t) = \min\Big\{ Z(j',t') + z\big(\{j'+1,\ldots,j\},[t',t)\big) : t',t \in \mathcal{P}, t' < t, j', j \in J, j' < j \Big\},$$

where $z\big(\{j'+1,\ldots,j\},[t',t)\big)$ denotes the value of an optimal partial schedule for job set $\{j'+1, j'+2, \ldots, j\}$ in the region $[t', t)$, or $\infty$ if no such schedule exists. This value can be computed in polynomial time, by Lemma 3.11. Hence, we compute $Z(j, t)$ for all $O(nK)$ states in polynomial time, which concludes the proof. □

The following observation follows from a standard interchange argument.

**Observation 3.13** An optimal schedule $\mathcal{S}^*$ for the problem $1\,|\,pmtn\,|\,\sum C_j + E$ processes jobs according to the Shortest Processing Time First (SPT) policy.

Combining this observation with Theorem 3.12 gives the following corollary.

**Corollary 3.14** There is a polynomial-time algorithm for $1\,|\,pmtn\,|\,\sum C_j + E$.

For the weighted problem $1\,|\,pmtn\,|\,\sum w_jC_j + E$, there is no sequence that is universally optimal for all reservation decisions [33]. However, in the context of scheduling on an unreliable machine there has been shown a polynomial-time algorithm that computes a universal $(4 + \varepsilon)$-approximation [33]. More precisely, the algorithm constructs a sequence of jobs which approximates the scheduling cost for any reservation decision with a factor at most $4 + \varepsilon$.

Consider an instance of problem $1\,|\,pmtn\,|\,\sum w_jC_j + E$ and compute such a universally $(4+\varepsilon)$-approximate sequence. Applying Theorem 3.12 to $\sigma$, we obtain a schedule $\mathcal{S}$ with an optimal reservation decision for $\sigma$. Let $\mathcal{S}'$ denote the schedule which we obtain by changing the reservation decision of $\mathcal{S}$ to the reservation in an optimal schedule $\mathcal{S}^*$ (but keeping the scheduling sequence $\sigma$). The schedule $\mathcal{S}'$ has cost no less than the original cost of $\mathcal{S}$. Furthermore, given the reservation decision in the optimal solution $\mathcal{S}^*$, the sequence $\sigma$ approximates the scheduling cost of $\mathcal{S}^*$ within a factor of $4+\varepsilon$. This gives the following result.

**Corollary 3.15** There exists a $(4+\varepsilon)$-approximation algorithm for the problem $1\,|\,pmtn\,|\,\sum w_jC_j + E$.

## 3.5 A PTAS for Minimizing Total Weighted Completion Time

The main result of this section is an approximation scheme for minimizing the total weighted completion time with time-varying reservation cost.

**Theorem 3.16** There exists a polynomial-time approximation scheme (PTAS) for $1\,|\,pmtn\,|\,\sum_j w_jC_j + E$.

In the remainder of this section we describe some preliminaries, present a dynamic programming (DP) algorithm with exponential running time, and then we argue that the running time can be reduced to polynomial time. As noted in the introduction, our approach is inspired by a PTAS for scheduling on a machine of varying speed [64], but a direct application does not seem possible.

### 3.5.1 Preliminaries and Scheduling in the Weight-Dimension

We describe a schedule $\mathcal{S}$ not in terms of completion times $C_j(\mathcal{S})$, but in terms of the remaining weight function $W^{\mathcal{S}}(t)$ which, for a given schedule $\mathcal{S}$, is defined as the total weight of all jobs not completed by time $t$. Based on the remaining weight function we can express the cost for any schedule $\mathcal{S}$ as

$$\int_0^\infty W^{\mathcal{S}}(t)\,\mathrm{d}t = \sum_{j\in J} w_jC_j(\mathcal{S})\,.$$

This has a natural interpretation in the standard 2D-Gantt chart, which was originally introduced in [32].

For a given reservation decision, we follow the idea of [64] and implicitly describe the completion time of a job $j$ by the value of the function $W$ at the time that $j$ completes. This value is referred to as the *starting weight* $S_j^w$ of job $j$.

In analogy to the time-dimension, the value $C_j^w := S_j^w + w_j$ is called *completion weight* of job $j$. When we specify a schedule in terms of the remaining weight function, then we call it a *weight-schedule*, otherwise a *time-schedule*. Other terminologies, such as feasibility and idle time, also translate from the time-dimension to the weight-dimension. A weight-schedule is called *feasible* if no two jobs overlap and the machine is called *idle in weight-dimension* if there exists a point $w$ in the weight-dimension with $w \notin \left[S_j^w, C_j^w\right]$ for all jobs $j \in J$.

A weight-schedule together with a reservation decision can be translated into a time-schedule by ordering the jobs in decreasing order of completion weights and scheduling them in this order in the time-dimension in the reserved time slots. For a given reservation decision, consider a weight-schedule $\mathcal{S}$ with completion weights $C_1^w > \cdots > C_n^w > C_{n+1}^w := 0$ and the corresponding completion times $0 =: C_0 < C_1 < \cdots < C_n$ for the jobs $j = 1, \ldots, n$. We define the *(scheduling) cost of a weight schedule* $\mathcal{S}$ as $\sum_{j=1}^{n} \left(C_{j+1}^w - C_j^w\right) C_j$. This value equals $\sum_{j=1}^{n} \pi_j^{\mathcal{S}} C_j^w$, where $\pi_j^{\mathcal{S}} := C_j - S_j$, if and only if there is no idle weight. If there is idle weight, then the cost of a weight-schedule can only be greater, and we can safely remove idle weight without increasing the scheduling cost [64].

### 3.5.2 A Dynamic Programming Algorithm

Let $\varepsilon > 0$. Firstly, we scale the input parameters so that all job weights $w_j$, $j = 1, \ldots, n$, and all unit reservation costs $e_k$, $k = 1, \ldots, K$, are non-negative integers. Then, we apply standard geometric rounding to the weights to gain more structure on the input, i.e, we round the weights of all jobs up to the next integer power of $(1+\varepsilon)$, by losing at most a factor $(1+\varepsilon)$ in the objective value. Furthermore, we discretize the weight-space into intervals of exponentially increasing size: we define intervals $WI_u := \left[(1+\varepsilon)^{u-1}, (1+\varepsilon)^u\right)$ for $u = 1, \ldots, \nu$ with $\nu := \lceil \log_{1+\varepsilon} \sum_{j \in J} w_j \rceil$.

Consider a subset of jobs $J' \subseteq J$ and a partial weight-schedule of $J'$. In the dynamic program, the set $J'$ represents the set of jobs at the beginning of a corresponding weight-schedule, i.e., if $j \in J'$ and $k \in J \setminus J'$, then $C_j^w < C_k^w$. However, the jobs in $J'$ are scheduled at the end in a corresponding time-schedule. As discussed at the beginning of this section, a partial weight-schedule $\mathcal{S}$ for the jobs in $J \setminus J'$ together with a reservation decision for these jobs can be translated into a time-schedule.

Let $\mathcal{F}_u := \{J_u \subseteq J : \sum_{j \in J_u} w_j \leq (1+\varepsilon)^u\}$ for $u = 1, \ldots, \nu$. The set $\mathcal{F}_u$ contains all the possible job sets $J_u$ that can be scheduled in $WI_u$ or before. Additionally, we define $\mathcal{F}_0$ to be the set that contains only the set of all zero-weight jobs $J_0 := \{j \in J : w_j = 0\}$. The following observation allow to restrict to simplified completion weights.

**Observation 3.17** Consider an optimal weight-schedule in which the set of jobs with completion weight in $WI_u$, $u \in \{1, \ldots, \nu\}$, is exactly $J_u \setminus J_{u-1}$ for some $J_u \in \mathcal{F}_u$ and $J_{u-1} \in \mathcal{F}_{u-1}$. By losing at most a factor $(1+\varepsilon)$ in the objective

value, we can assume that for all $u \in \{1, \dots, \nu\}$ the completion weight of the jobs in $J_u \setminus J_{u-1}$ is exactly $(1+\varepsilon)^u$.

The following observation follows from a simple interchange argument.

**Observation 3.18** There is an optimal time-schedule in which $J_0$ is scheduled completely after all jobs in $J \setminus J_0$.

The dynamic program recursively constructs states $Z = [J_u, b, avg]$ and computes for every state a time point $t(Z)$ with the following meaning. A state $Z = [J_u, b, avg]$ with time point $t(Z)$ expresses that that there is a feasible partial schedule $\mathcal{S}$ for the jobs in $J \setminus J_u$ with $J_u \in \mathcal{F}_u$ up to the time point $t(Z)$ for which the total reservation cost is at most $b$ and for which the *average scheduling cost*, i.e.,

$$\frac{1}{t(Z)} \cdot \int_0^{t(Z)} W^{\mathcal{S}}(t)\,\mathrm{d}t,$$

is at most $avg$. Note that $avg \cdot t(Z)$ is an upper bound on the total scheduling cost of $\mathcal{S}$ and that the average scheduling cost is non-increasing in time, because the remaining weight function $W^{\mathcal{S}}(t)$ is non-increasing in time. In the *iteration for* $u$, we only consider states $[J_u, b, avg]$ with $J_u \in \mathcal{F}_u$. The states in the iteration for $u$ are created based on the states from the iteration for $u+1$. Initially, we only have the state $Z_\nu = [J, 0, 0]$ with $t(Z_\nu) := 0$, we start the dynamic program with $u = \nu - 1$, iteratively reduce $u$ by one, and stop the process after the iteration for $u = 0$. In the iteration for $u$, the states together with their time points are constructed in the following way. Consider candidate sets $J_{u+1} \in \mathcal{F}_{u+1}$ and $J_u \in \mathcal{F}_u$, a partial weight-schedule $\mathcal{S}$ of $J \setminus J_u$, in which the set of jobs with completion weight in $WI_{u+1}$ is exactly $J_{u+1} \setminus J_u$, two budgets $b_1, b_2$ with $b_1 \leq b_2$, and two bounds on the average scheduling cost $avg_1, avg_2$. Let $Z_1 = [J_{u+1}, b_1, avg_1]$ and $Z_2 = [J_u, b_2, avg_2]$ be the corresponding states. We know that there is a feasible partial schedule for the job set $J \setminus J_{u+1}$ up to time $t(Z_1)$ having average scheduling cost at most $avg_1$ and reservation cost at most $b_1$. By augmenting this schedule, we want to compute a minimum time point $t(Z_1, Z_2)$ that we associate with the link between $Z_1$ and $Z_2$ so that there is a feasible partial schedule for $J \setminus J_u$ that processes the jobs from $J_{u+1} \setminus J_u$ in the interval $[t(Z_1), t(Z_1, Z_2))$, has average scheduling cost at most $avg_2$, and reservation cost at most $b_2$. That is, $t(Z_1, Z_2)$ is the minimum makespan if we start with $Z_1$ and want to arrive at $Z_2$. For the computation of $t(Z_1, Z_2)$, we use the following subroutine.

Using Observation 3.17, we approximate the area under the remaining weight function $W^{\mathcal{S}}(t)$ for the jobs in $J_{u+1} \setminus J_u$ by $(1+\varepsilon)^{u+1} \cdot (t(Z_1, Z_2) - t(Z_1))$, where $t(Z_1, Z_2)$ is the time point that we want to compute. Approximating this area gives us the flexibility to schedule the jobs in $J_{u+1} \setminus J_u$ in any order. However, we need that $avg_2 \cdot t(Z_1, Z_2)$ is an upper bound on the integral of the remaining

weight function by time $t(Z_1, Z_2)$. That is, we want that

$$avg_2 \cdot t(Z_1, Z_2) \geq (1+\varepsilon)^{u+1} \cdot t(Z_1, Z_2) + t(Z_1) \cdot (avg_1 - (1+\varepsilon)^{u+1}).$$

Both the left-hand side and the right-hand side of this inequality are linear functions in $t(Z_1, Z_2)$. So, we can compute a smallest time point $t^{LB}$ such that the right-hand side is greater or equal to the left-hand side for all $t(Z_1, Z_2) \geq t^{LB}$. If there is no such $t^{LB}$, then we set $t(Z_1, Z_2)$ to infinity and stop the subroutine. Otherwise, we know that our average scheduling cost at $t^{LB}$ or later is at most $avg_2$. Let $E(p, [t_1, t_2))$ denote the total cost of the $p$ cheapest slots in the interval $[t_1, t_2)$. We compute the smallest time point $t(Z_1, Z_2) \geq t^{LB}$ so that the set of jobs $J_{u+1} \setminus J_u$ can be feasibly scheduled in $[t(Z_1), t(Z_1, Z_2))$ having reservation cost not more than $b_2 - b_1$. That is, we set

$$t(Z_1, Z_2) = \min \left\{ t \geq \max\{t(Z_1), t^{LB}\} : E(p(J_{u+1} \setminus J_u), [t(Z_1), t)) \leq b_2 - b_1 \right\}.$$

The time point $t(Z_1, Z_2)$ can be computed in polynomial time by applying binary search to the interval $[\max\{t(Z_1), t^{LB}\}, d_K)$, since $E(p, [t_1, t_2))$ is a monotone function in $t_2$.

Given all possible states $[J_{u+1}, b_1, avg_1]$ from the iteration for $u+1$, the dynamic program enumerates for all these states all possible links to states $[J_u, b_2, avg_2]$ from the iteration for $u$ fulfilling the above requirement on the candidate sets $J_{u+1}$ and $J_u$, on the budgets $b_1$ and $b_2$, and on the average scheduling costs $avg_1$ and $avg_2$. For any such possible link $(Z_1, Z_2)$ between states from the iteration for $u+1$ and $u$, we apply the above subroutine and associate the time point $t(Z_1, Z_2)$ with this link. Thus, the dynamic program associates several possible time points with a state $Z_2 = [J_u, b_2, avg_2]$ from the iteration for $u$. However, we only keep the link with the smallest associated time point $t(Z_1, Z_2)$ (ties are broken arbitrarily) and this defines the time point $t(Z_2)$ that we associate with the state $Z_2$. That is, for a state $Z_2$ from the iteration for $u$ we define $t(Z_2) := \min\{t(Z_1, Z_2) \mid Z_1 \text{ is a state from the iteration for } u+1\}$.

Let $E_{\max}$ be an upper bound on the total reservation cost in an optimal solution, e.g., the total cost of the first $p(J)$ finite-cost time slots. The dynamic program does not enumerate all possible budgets but only a polynomial number of them, namely budgets with integer powers of $(1+\eta_1)$ with $\eta_1 > 0$ determined later (see proof of Lemma 3.19). That is, for the budget on the reservation cost, the dynamic program enumerates all values in

$$B := \{0, 1, (1+\eta_1), (1+\eta_1)^2, \ldots, (1+\eta_1)^{\omega_1}\} \text{ with } \omega_1 = \lceil \log_{1+\eta_1} E_{\max} \rceil.$$

Similarly, we observe that $(1+\varepsilon)^\nu$ is an upper bound on the average scheduling cost. The dynamic program does also only enumerate a polynomial number of possible average scheduling costs, namely integer powers of $(1+\eta_2)$ with $\eta_2 > 0$ also determined later (see proof of Lemma 3.19). This means, for the average scheduling cost, the dynamic program enumerates all values in

$$AVG := \{0, 1, (1+\eta_2), (1+\eta_2)^2, \ldots, (1+\eta_2)^{\omega_2}\} \text{ with } \omega_2 = \lceil \nu \log_{1+\eta_2}(1+\varepsilon) \rceil.$$

The dynamic program stops after the iteration for $u = 0$. Now, only the set of zero-weight jobs is not scheduled yet. For any state $Z = [J_0, b, avg]$ constructed in the iteration for $u = 0$, we append the zero-weight jobs starting at time $t(Z)$ and reserving the cheapest slots, which is justified by Observation 3.18. We add the additional reservation cost to $b$. After this, we return the state $Z = [J_0, b, avg]$ and its corresponding schedule, which can be computed by backtracking and following the established links, with minimum total cost $b + avg \cdot t(Z)$. With this, we obtain the following result.

**Lemma 3.19** The dynamic program computes a $(1 + O(\varepsilon))$-approximate solution.

***Proof.*** Consider an arbitrary iteration $i$ of the dynamic program and suppose, we consider states $Z = [J_u, b, avg]$ with $J_u \in \mathcal{F}_u$, $b \in B$, and $avg \in AVG$ for which we construct the time points $t(Z)$. Let $Z_1^* = [J_{u+1}^*, b_1^*, avg_1^*]$ and $Z_2^* = [J_u^*, b_2^*, avg_2^*]$ with $J_{u+1}^* \in \mathcal{F}_{u+1}$ and $J_u^* \in \mathcal{F}_u$ be the states that represent an optimal solution $S^*$ for which the set of jobs with completion weight in $WI_{u+1}$ is exactly $J_{u+1}^* \setminus J_u^*$. By Observation 3.17, we assume that also in $S^*$ the area under the remaining weight function $W^{\mathcal{S}^*}(t)$ for the jobs in $J_{u+1}^* \setminus J_u^*$ is approximated by $(1+\varepsilon)^{u+1} \cdot (t(Z_2^*) - t(Z_1^*))$. We now show the following. The dynamic program constructs in iteration $i$ a state $Z = [J_u, b, avg]$ with $J_u \in \mathcal{F}_u$, $b \in B$, and $avg \in AVG$ such that

(i) $J_u = J_u^*$,

(ii) $b \leq (1 + \eta_1)^i \cdot b_2^*$,

(iii) $avg \leq (1 + \eta_2)^i \cdot avg_2^*$, and

(iv) $t(Z) \leq t(Z_2^*)$.

We prove this statement by induction on $i$. Consider the first iteration of the dynamic program, in which we consider states with job sets from $\mathcal{F}_{\nu-1}$. Let $Z^* = [J_{\nu-1}^*, b^*, avg^*]$ be the state that corresponds to the optimal solution $S^*$. The dynamic program also considers the job set $J_{\nu-1}^*$. Suppose, we reserve the same slots that $S^*$ reserves for the jobs in $J \setminus J_{\nu-1}^*$ in the interval $[0, t(Z^*))$. Let $b$ be the resulting reservation cost after rounding $b^*$ up to the next value in $B$. With this, we know that $b \leq (1+\eta_1) \cdot b^*$. Furthermore, by our assumption, we know that the average scheduling cost of $S^*$ up to time $t(Z^*)$ is $(1+\varepsilon)^\nu$. Let $avg$ be $(1+\varepsilon)^\nu$ rounded up to the next value in $AVG$. Then we know that $avg \leq (1+\eta_2) \cdot avg^*$. The dynamic program also considers the state $Z = [J_{\nu-1}^*, b, avg]$. However, the dynamic program computes the *minimum* time point $t(Z_\nu, Z) \geq t^{LB}$ so that the set of jobs $J \setminus J_{\nu-1}^*$ can be feasibly scheduled in $[0, t(Z_\nu, Z))$ having reservation cost not more than $b$. This implies that $t(Z_\nu, Z) \leq t(Z^*)$, which implies that $t(Z) \leq t(Z^*)$. Note that $t^{LB} = 0$ for the specified values in $Z$.

Suppose, the statement is true for the iterations $1, 2, \ldots, i$. We prove that it is also true for iteration $i+1$, in which we consider job sets from $\mathcal{F}_u$. Again, let $Z_1^* = [J_{u+1}^*, b_1^*, avg_1^*]$ and $Z_2^* = [J_u^*, b_2^*, avg_2^*]$ with $J_{u+1}^* \in \mathcal{F}_{u+1}$ and $J_u^* \in \mathcal{F}_u$ be the states that represent $S^*$. By our hypothesis, we know that the dynamic program constructs a state $Z_1 = [J_{u+1}, b_1, avg_1]$ with

(i) $J_{u+1} = J_{u+1}^*$,

(ii) $b_1 \leq (1+\eta_1)^i \cdot b_1^*$,

(iii) $avg_1 \leq (1+\eta_2)^i \cdot avg_1^*$, and

(iv) $t(Z_1) \leq t(Z_1^*)$.

We augment this schedule in the following way. Suppose, we reserve the same slots that $S^*$ reserves for the jobs in $J_{u+1}^* \setminus J_u^*$ in the interval $[t(Z_1^*), t(Z_2^*))$. Let $b_2$ be the resulting total reservation cost after rounding up to the next value in $B$. Thus, there is a feasible schedule for $J \setminus J_u^*$ having reservation cost of at most

$$\begin{aligned} b_2 &\leq (1+\eta_1) \cdot (b_1 + b_2^* - b_1^*) \\ &\leq (1+\eta_1)^{i+1} \cdot (b_1^* + b_2^* - b_1^*) \\ &= (1+\eta_1)^{i+1} \cdot b_2^*. \end{aligned}$$

The new average scheduling cost after rounding to the next value in $AVG$ is

$$\begin{aligned} avg_2 &\leq (1+\eta_2) \cdot \frac{avg_1 \cdot t(Z_1) + (1+\varepsilon)^{u+1} \cdot (t(Z_2^*) - t(Z_1))}{t(Z_2^*)} \\ &\leq (1+\eta_2)^{i+1} \cdot \frac{avg_1^* \cdot t(Z_1) + (1+\varepsilon)^{u+1} \cdot (t(Z_2^*) - t(Z_1))}{t(Z_2^*)} \\ &\leq (1+\eta_2)^{i+1} \cdot \frac{avg_1^* \cdot t(Z_1^*) + (1+\varepsilon)^{u+1} \cdot (t(Z_2^*) - t(Z_1^*))}{t(Z_2^*)} \\ &= (1+\eta_2)^{i+1} \cdot avg_2^*. \end{aligned}$$

The third inequality follows from the fact that $avg_1^* \geq (1+\varepsilon)^{u+1}$. The dynamic program also considers the link between the state $Z_1$ and $Z_2 := [J_u^*, b_2, avg_2]$. We first observe that $t^{LB} \leq t(Z_2^*)$, since

$$avg_2 \cdot t(Z_2^*) \geq avg_1 \cdot t(Z_1) + (1+\varepsilon)^{u+1} \cdot (t(Z_2^*) - t(Z_1))$$

by construction of $avg_2$. Furthermore, we observe that $b_2 - b_1 \geq b_2^* - b_1^*$ by construction of $b_2$. These two facts together with $t(Z_1) \leq t(Z_1^*)$ imply that $t(Z_1, Z_2) \leq t(Z_2^*)$, which implies that $t(Z_2) \leq t(Z_2^*)$.

To complete the proof, we need to specify the parameters $\eta_1$ and $\eta_2$. We want that $(1+\eta_i)^\nu \leq (1+\varepsilon)$ for $i = 1, 2$. We claim that for a given $\nu \geq 1$ there exists an $\bar{\eta} > 0$ such that for all $\eta \in (0, \bar{\eta}]$ we have $(1+\eta)^\nu \leq 1 + 2\nu\eta$.

Consider the function $f(\eta) := (1+\eta)^\nu - 1 - 2\nu\eta$. We have that $f(0) = 0$ and $f'(\eta) < 0$ for $\eta \in [0, 2^{1/(\nu-1)} - 1)$. This shows the claim. Hence, we choose $\eta_i = \min\{\frac{\varepsilon}{2\nu}, 2^{1/(\nu-1)} - 1\}$ for $i = 1, 2$. This shows the statement of the lemma and that the size of $B$ as well as the size of $AVG$ are bounded by a polynomial in the size of the input. □

We remark that the given DP works for more general reservation cost functions $e : \mathbb{N} \to \mathbb{Q}_{\geq 0}$ than considered here in this thesis. As argued in the proof, it is sufficient for the DP that there is a function $E(p, [t_1, t_2))$ that outputs in polynomial time for a given time interval $[t_1, t_2)$ and a given $p \in \mathbb{Z}_{\geq 0}$ the total cost of the $p$ cheapest slots in $[t_1, t_2)$.

We also remark that the running time of the presented DP is exponential, because the size of the sets $\mathcal{F}_u$ are exponential in the size of the input. However, in the next section we show that we can trim the sets $\mathcal{F}_u$ down to ones of polynomial size at an arbitrarily small loss in the performance guarantee.

### 3.5.3 Trimming the State Space

The set $\mathcal{F}_u$, containing all possible job sets $J_u$, is of exponential size, and so is the DP state space. In the context of scheduling with variable machine speed, it has been shown in [64] how to reduce the set $\mathcal{F}_u$ for a similar DP (without reservation decision, though) to a set $\tilde{\mathcal{F}}_u$ of polynomial size at only a small loss in the objective value. In general, such a procedure is not necessarily applicable to our setting because of the different objective involving additional reservation cost and the different decision space. However, the compactification in [64] holds *independently of the speed of the machine* and, thus, independently of the reservation decision of the DP (interpret non/reservation as speed 0/1). Hence, we can apply it to our cost-aware scheduling framework and obtain a PTAS. We now describe the building blocks for this trimming procedure and argue why we can apply it in order to obtain the set $\tilde{\mathcal{F}}_u$ for our problem.

**Light Jobs.** The first building block for the trimming procedure is a classification of the jobs based on their weights.

**Definition 3.20** Given a weight schedule and a job $j \in J$ with starting weight $S_j^w \in WI_u$, we call job $j$ *light* if $w_j \leq \varepsilon^2 |WI_u|$, otherwise $j$ is called *heavy*.

This classification enables us to structure near-optimal solutions.

**Lemma 3.21 ([64])** At a loss of a factor of $1 + O(\varepsilon)$ in the scheduling cost, we can assume the following. For a given interval $WI_u$, consider any pair of light jobs $j, k$. If both jobs start in $WI_u$ or later and $p_k/w_k \leq p_j/w_j$, then $C_j^w \leq C_k^w$.

We remark, that Lemma 3.21 holds independently of the speed of the machine, as pointed out in [64]. Translated to our problem, this means that at a loss

of a factor of $1 + O(\varepsilon)$ in the scheduling cost we can assume that light jobs are scheduled according to *reverse Smith's rule* in the weight-dimension. Most importantly, this statement holds regardless of our actual reservation decision.

**Localization.** We now localize jobs in the weight-dimension to gain more structure. That is, we determine for every job $j \in J$ two values $r_j^w$ and $d_j^w$ such that, independently of our actual reservation decision, $j$ is scheduled completely within $\left[r_j^w, d_j^w\right)$ in some $(1+O(\varepsilon))$-approximate weight-schedule (in terms of the scheduling cost). We call $r_j^w$ and $d_j^w$ the *release-weight* and the *deadline-weight* of job $j$, respectively.

**Lemma 3.22 ([64])** We can compute in polynomial time values $r_j^w$ and $d_j^w$ for each $j \in J$ such that: (i) there exists a $(1 + O(\varepsilon))$-approximate weight-schedule (in terms of the scheduling cost) that processes each job $j$ within $[r_j^w, d_j^w)$, (ii) there exists a constant $s \in O(\log(1/\varepsilon)/\varepsilon)$ such that $d_j^w \leq r_j^w \cdot (1+\varepsilon)^s$, (iii) $r_j^w$ and $d_j^w$ are integer powers of $(1+\varepsilon)$, and (iv) the values $r_j^w$ an $d_j^w$ are independent of the speed of the machine.

This lemma enables us to localize all jobs in $J$ in polynomial time and independent of our actual reservation decision, as guaranteed by property (iv).

**Compact Search Space.** Based on the localization of jobs in weight space, we can cut the number of different possibilities for a candidate set $J_u$ in iteration for $u$ of our DP down to a polynomial number. That is, we replace the set $\mathcal{F}_u$ by a polynomially sized set $\tilde{\mathcal{F}}_u$. Instead of describing all sets $S \in \tilde{\mathcal{F}}_u$ explicitly, we give all possible complements $R = J \setminus S$ and collect them in a set $\mathcal{D}_u$, where a set $R \in \mathcal{D}_u$ represents a possible set of jobs having completion weights in $WI_{u+1}$ or later. Obviously, a set $R \in \mathcal{D}_u$ must contain all jobs $j \in J$ having a release weight $r_j^w \geq (1+\varepsilon)^u$. Furthermore, we know that $d_j^w \geq (1+\varepsilon)^{u+1}$ is necessary for job $j$ to be in a set $R \in \mathcal{D}_u$. Following property (ii) in Lemma 3.22, we thus only need to decide about the jobs having a release weight $r_j^w = (1+\varepsilon)^i$ with $i \in \{u+1-s, \ldots, u-1\}$. An enumeration over basically all possible job sets for each $i \in \{u+1-s, \ldots, u-1\}$ gives the following desired result.

**Lemma 3.23 ([64])** For each $u$, we can construct in polynomial time a set $\tilde{\mathcal{F}}_u$ that satisfies the following: (i) there exists a $(1 + O(\varepsilon))$-approximate weight-schedule (in terms of the scheduling cost) in which the set of jobs with completion weight at most $(1+\varepsilon)^u$ belongs to $\tilde{\mathcal{F}}_u$, (ii) the set $\tilde{\mathcal{F}}_u$ has cardinality at most $2^{O\left(\log^3(1/\varepsilon)/\varepsilon^2\right)}$, and (iii) the set $\tilde{\mathcal{F}}_u$ is completely independent of the speed of the machine.

Again, Property (iii) implies that we can construct the set $\tilde{\mathcal{F}}_u$ independently of our reservation decision.

To complete the proof of Theorem 3.16 it remains to argue on the running time of the DP. The DP has $\nu$ iterations, where in each iteration for at most $2^{O\left(\log^3(1/\varepsilon)/\varepsilon^2\right)} \cdot |B| \cdot |AVG|$ previous states at most $2^{O\left(\log^3(1/\varepsilon)/\varepsilon^2\right)} \cdot |B| \cdot |AVG|$ many links to new states are considered. Therefore, the running time complexity of our DP is $\nu \cdot (2^{O\left(\log^3(1/\varepsilon)/\varepsilon^2\right)} \cdot |B| \cdot |AVG|)^2$, which is bounded by a polynomial in the size of the input.

## 3.6 Consequences of Having Release Dates

In this section, we analyze the influence of release dates on the problem complexity. First, we consider the makespan objective on a single machine in Section 3.6.1 and on unrelated parallel machines with a slightly relaxed cost model in Section 3.6.2. Second, we focus on the total completion time objective in Section 3.6.3.

We remark that it is a priori not clear how to encode a feasible schedule using polynomial space when having release dates. However, we can do the following. First, we want that there is no interval $I_k := [s_k, d_k)$ and no job $j$ with $r_j \in (s_k, d_k)$. If, however, there is such an interval-job combination $(k, j)$, we repartition $I_k$ into $I_k^1 := [s_k, r_j)$ and $I_k^2 := [r_j, d_k)$ until the desired condition is fulfilled. This gives a new partitioning of the time horizon into intervals $I_k$ with $k = 1, \ldots, K'$, where $K'$ is polynomial in the size of the input. Then it is not hard to see that if an optimal schedule reserves $p$ units in any interval $I_k$, then it reserves the first $p$ slots in $I_k$. This observation allows us to encode a schedule using polynomial space.

### 3.6.1 Minimizing the Makespan on a Single Machine

The scheduling problem $1 \,|\, pmtn, r_j \,|\, C_{\max}$ without reservation can be solved optimally in polynomial time by scheduling the jobs as early as possible in order of non-decreasing release dates [6]. We show that the more general problem with time-varying reservation cost is also polynomial-time solvable. In the following, we first give a combinatorial algorithm and then an LP-based algorithm.

Consider an arbitrary instance of the scheduling problem $1 \,|\, pmtn, r_j \,|\, C_{\max} + E$.

**Observation 3.24** There is an optimal solution that schedules the jobs in order of non-decreasing release dates.

This observation can be shown by a standard interchange argument and is the basis for our combinatorial algorithm. Let us assume for the moment that we know an optimal makespan $C^*_{\max}$. We now show that, given the makespan $C^*_{\max}$, we can compute an optimal schedule $\mathcal{S}^*$. In a first step, we sort the jobs in order of non-decreasing release dates and schedule them in this order as early as

possible. We assume that jobs are numbered so that $r_1 \leq r_2 \leq \ldots \leq r_n$. This gives a schedule $\mathcal{S}$ with an earliest possible starting time $S_j$ (w.r.t. the given ordering in $\mathcal{S}$) for each $j \in J$ and we note that $C_j = S_j + p_j$ for all $j \in J$. In a second step, we find the cheapest slots so as to realize the scheduling order of $\mathcal{S}$ in $[0, C^*_{\max})$. For $j = n, \ldots, 1$ we do the following: in addition to the $\sum_{j+1}^{n} p_j$ already reserved slots, we reserve in $[S_j, C^*_{\max})$ the $p_j$ cheapest free slots. After computing the cheapest slots, we reassign the jobs to the reserved slots by keeping the scheduling order of $\mathcal{S}$.

**Lemma 3.25** Given an optimal makespan for an instance of the problem $1 \,|\, pmtn, r_j \,|\, C_{\max} + E$, we can compute an optimal schedule in polynomial time.

***Proof.*** We show that the above-described algorithm computes an optimal schedule. By Observation 3.24, it is sufficient to show that we reserve the $p(J)$ cheapest slots so as to feasibly schedule the jobs. We show this by induction on the number of jobs. If there is only one job, this is trivially fulfilled. Suppose it is true for $n-1$ jobs. When having $n$ jobs, we use the induction hypothesis for the jobs $2, \ldots, n$ and the fact that in addition to the already reserved slots we reserve the $p_1$ cheapest free slots in a feasible way. □

Note that we partitioned the time horizon into polynomially many intervals $I_k = [s_k, d_k)$ with $s_k, d_k \in \mathbb{N}$, $k = 1, \ldots, K$, within which unit-size time slots have the same unit reservation cost $e_k$. Additionally, we can assume that there is no interval $I_k$ such that $r_j \in (s_k, d_k)$ for some job $j \in J$; see beginning of this section for a discussion on that.

In order to find $C^*_{\max}$ in polynomial time we compute for every interval $I_k$, $k = 1, \ldots, K$, an optimal makespan $C^k_{\max}$ assuming that $C^*_{\max} \in I_k$ and then we return a makespan that leads to a solution with minimum total cost. For this we use the same procedure as given in Section 3.3 for the problem without release dates, which is as follows. Assuming that we guessed the right interval $I_k$, we start with $C^k_{\max} = s_k$ and apply the above procedure to compute an optimal schedule for this makespan. Then we replace the currently most expensive slots with slots from $I_k$ as long as the most expensive slots have unit cost more than $e_k + 1$. With this, it takes at most $K$ computer operations to find $C^k_{\max}$. This shows the following statement.

**Theorem 3.26** The scheduling problem $1 \,|\, pmtn, r_j \,|\, C_{\max} + E$ can be solved in polynomial time.

We now give an alternative algorithm for computing an optimal schedule $\mathcal{S}^*$, given an optimal makespan $C^*_{\max}$ (see Lemma 3.25), by showing that this problem can be formulated as a flow problem that can be solved in polynomial time. Let $I_\ell$ be the interval with $C^*_{\max} \in I_\ell$. We create four types of nodes, a super-source $s$ with supply $p(J)$, a super-sink $t$ with demand $p(J)$, a node $j$ for every job $j \in J$

and a node $k$ for every interval $I_k$, $k = 1, \dots, K$. We connect every interval-node $k$ to $s$ by an arc $(s, k)$ with capacity $|I_k|$ and cost $e_k$ for $k = 1, \dots, \ell - 1$, capacity $C^*_{\max} - s_\ell$ and cost $e_\ell$ for $k = \ell$, and zero capacity and cost $e_k$ for $k > \ell$. Additionally, we connect interval-node $k$ to job-node $j$ by an arc $(k, j)$ with infinite capacity and zero cost if $r_j \leq s_k$. At last, we connect every job-node $j$ to the super-sink $t$ by an arc $(j, t)$ having capacity $p_j$ and zero cost. It is easy to show that a solution to the constructed flow problem can be translated into a feasible schedule having the same reservation cost and vice versa.

### 3.6.2 Minimizing the Makespan on Unrelated Machines with Fractional Reservation

In this section, we consider the scheduling problem $R \,|\, pmtn, r_j \,|\, C_{\max}$ with reservation cost and show that it can be solved in polynomial time if we relax the requirement that we have to pay the full unit reservation cost for a time slot once we use it, no matter how much we utilize its unit capacity. That is, utilzing a time slot $[t, t+1)$ to an extend of $p \in [0, 1]$ incurs now a reservation cost of $p \cdot e(t)$. We show that a standard LP-formulation for $R \,|\, pmtn, r_j \,|\, C_{\max}$ [58] can be adapted to solve the problem variant with *fractional reservation cost* $R \,|\, pmtn, r_j \,|\, C_{\max} + E_f$.

As in previous sections, we compute for every interval $I_k$, $k = 1, \dots, K$, an optimal makespan $C^k_{\max}$ assuming that an optimal makespan $C^*_{\max}$ lies in $I_k$ and then we return a makespan that leads to a solution with minimum total cost. Note that w.l.o.g. we have no interval-job combination $(k, j)$ with $r_j \in (s_k, d_k)$. Let $y_{ijk} \in [0, 1]$ be the total amount of time that machine $i$ works on job $j$ in interval $I_k$. We obtain the following LP-formulation for the $\kappa$th interval:

$$\min \sum_{k=1}^{\kappa} e_k \cdot x_k + C^\kappa_{\max} \tag{3.1a}$$

$$\text{s.t.} \sum_{i \in M} \sum_{k=1}^{\kappa} \frac{y_{ijk}}{p_{ij}} = 1 \qquad \forall j \in J, \tag{3.1b}$$

$$\sum_{i \in M} y_{ijk} \leq x_k \qquad \forall j \in J, k = 1, \dots, \kappa, \tag{3.1c}$$

$$\sum_{j \in J} y_{ijk} \leq x_k \qquad \forall i \in M, k = 1, \dots, \kappa, \tag{3.1d}$$

$$0 \leq x_k \leq d_k - s_k \qquad \forall k = 1, \dots, \kappa, \tag{3.1e}$$

$$C^\kappa_{\max} = s_\kappa + x_\kappa \tag{3.1f}$$

$$y_{ijk} \geq 0 \qquad \forall i \in M, j \in J, k = 1, \dots, \kappa, \tag{3.1g}$$

$$y_{ijk} = 0 \qquad \forall i \in M, j \in J, k : r_j > s_k. \tag{3.1h}$$

In an optimal solution for this LP, variable $x_k$ equals the total amount of time that we need to reserve in interval $I_k$, i.e., we reserve from $s_k$ to $s_k + x_k$.

Assuming that some jobs are processed in interval $I_\kappa$, we can show that for any feasible solution to this LP, there is a feasible schedule with the same total cost by using [58]. If there are no jobs processed in $I_\kappa$, then accepting $C^\kappa_{\max} = s_\kappa$ is fine, given our assumption that an optimal makespan $C^*_{\max}$ lies in $I_k$.

The above LP can easily be adapted if we have machine-individual fractional reservation cost. That is, we are given intervals $I_{ik} = [s_{ik}, d_{ik})$, $k = 1, \ldots, K$, for all $i \in M$, within which unit-size time slots have the same unit reservation cost $e_{ik}$. Note that we can assume w.l.o.g. that $s_{ik} = s_{i'k}$ and $d_{ik} = d_{i'k}$ for all $i, i' \in M$ and $k = 1, \ldots, K$. We obtain the following LP-formulation for the $\kappa$th interval:

$$\min \sum_{k=1}^{\kappa} e_{ik} \cdot x_{ik} + C^\kappa_{\max} \tag{3.2a}$$

$$\text{s.t.} \sum_{i \in M} \sum_{k=1}^{\kappa} \frac{y_{ijk}}{p_{ij}} = 1 \qquad \forall j \in J, \tag{3.2b}$$

$$\sum_{i \in M} y_{ijk} \leq x_k \qquad \forall j \in J, k = 1, \ldots, \kappa, \tag{3.2c}$$

$$\sum_{j \in J} y_{ijk} \leq x_{ik} \qquad \forall i \in M, k = 1, \ldots, \kappa, \tag{3.2d}$$

$$x_{ik} \leq x_k \qquad \forall i \in M, k = 1, \ldots, \kappa, \tag{3.2e}$$

$$0 \leq x_k \leq d_k - s_k \qquad \forall k = 1, \ldots, \kappa, \tag{3.2f}$$

$$0 \leq x_{ik} \leq d_k - s_k \qquad \forall i \in M, k = 1, \ldots, \kappa, \tag{3.2g}$$

$$C^\kappa_{\max} = s_\kappa + x_\kappa \tag{3.2h}$$

$$y_{ijk} \geq 0 \qquad \forall i \in M, j \in J, k = 1, \ldots, \kappa, \tag{3.2i}$$

$$y_{ijk} = 0 \qquad \forall i \in M, j \in J, k : r_j > s_k. \tag{3.2j}$$

Hence, we obtain the following result.

**Theorem 3.27** The problem $R \,|\, pmtn, r_j \,|\, C_{\max} + E_f$ can be solved in polynomial time, even if we have machine-individual fractional reservation cost.

### 3.6.3 Minimizing the Total Completion Time on a Single Machine

The scheduling problem $1 \,|\, pmtn \,|\, \sum C_j + E$ is solvable in polynomial time as we saw in Section 3.4. In this section, we show that the problem becomes NP-hard once release dates are considered. That is, we show the following statement.

**Theorem 3.28** The scheduling problem $1 \,|\, pmtn, r_j \,|\, \sum C_j + E$ is NP-hard, even for $K = 2$ with only one interval having positive unit reservation cost.

The proof is inspired by the NP-hardness proof for $1\,|\,pmtn, r_j, d_j\,|\,\sum C_j$ by [30]. The reduction is from the NP-complete Odd-Even Partition problem to the decision version of $1\,|\,pmtn, r_j\,|\,\sum C_j + E$.

---

**Odd-Even Partition**

---

Given: We are given $2n$ positive integers $a_1, \ldots, a_{2n}$ with $\sum_{j=1}^{2n} a_j = 2B$, where $a_j > a_{j+1}$ for $j = 1, \ldots, 2n-1$.

Task: Decide whether there is a partition of the integers into two sets, $A_1$ and $A_2$, such that

$$\sum_{j \in A_1} a_j = \sum_{j \in A_2} a_j,$$

where $A_1$ and $A_2$ each contain exactly one element of each odd-even pair $\{a_{2i-1}, a_{2i}\}$, $i = 1, \ldots, n$.

---

Without loss of generality, we can assume that

$$(\sum_{k=1}^{i-1} a_{2k}) + a_{2i-1} + a_{2i} > B \quad \forall i = 1, \ldots, n. \tag{3.3}$$

If a given instance of the Odd-Even Partition problem does not satisfy this assumption, then we manipulate the instance so that it does. With $b_i := 4^{n+1-i}B$ we set $a'_{2i-1} := a_{2i-1} + b_i$ and $a'_{2i} := a_{2i} + b_i$ for all $i = 1, \ldots, n$ so that now $\sum_{j=1}^{2n} a'_j = 2B + 2\sum_{i=1}^{n} b_i = 2B'$ with $B' = (4^n + 4^{n-1} + \cdots + 4^2 + 4 + 1)B$. This transformation can be done in polynomial time and does not change the answer to the problem instance. We observe that now

$$\begin{aligned}(\sum_{k=1}^{i-1} a'_{2k}) + a'_{2i-1} + a'_{2i} &> (\sum_{k=1}^{i-1} b_k) + 2b_i \\ &= (4^n + 4^{n-1} + \cdots + 4^{n+3-i} + 4^{n+2-i})B + 2 \cdot 4^{n+1-i}B \\ &> B' \quad \forall i = 1, \ldots, n,\end{aligned}$$

because $2 \cdot 4^{n+1-i} > 4^{n+1-i} + 4^{n-i} + \cdots + 4^2 + 4 + 1$ for all $i = 1, \ldots, n$.

We construct the following instance of the decision version of $1\,|\,pmtn, r_j\,|\,\sum C_j + E$: We create $2n$ jobs $j = 1, \ldots, 2n$ with processing time $p_j = a_j$. We set $r_1 = 0$ and job $2i-1$ with $i \in \{2, \ldots, n\}$ is released at $r_{2i-1} = \sum_{k=1}^{i-1}(n-k+2)(a_{2k-1} - a_{2k}) + a_{2k} + \ell_k$, where $\ell_k = \rho(1-\rho)^{k-1} \cdot L$ with $\rho = 2n/(2n+1)$ and $L = (1/(1-\rho)^n)(B + \sum_{i=1}^{n}(n-i+2)(a_{2i-1} - a_{2i}) + a_{2i})$. Job $2i$ with $i \in \{1, \ldots, n\}$ is released at $r_{2i} = r_{2i-1} + (n-i+2)(a_{2i-1} - a_{2i})$. We have only two intervals $I_1 = [s_1, d_1) = [0, r_{2n} + a_{2n} + \ell_n)$ and $I_2 = [s_2, d_2) = [d_1, d_1 + B)$ with unit reservation cost $e_1 = 2nd_2 + 1$ and $e_2 = 0$. See also Fig. 3.1 for a schematic representation of this instance.

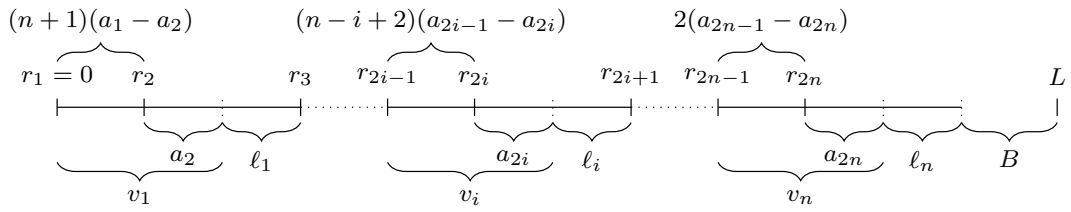


Figure 3.1: Schematic representation of the constructed instance of $1\,|\,pmtn, r_j\,|\,\sum C_j + E$.

Let $v_i := r_{2i} - r_{2i-1} + a_{2i}$ for $i = 1, \ldots, n$. Note that $d_2 = L$, since $d_2 = \sum_{i=1}^{n} v_i + \sum_{i=1}^{n} \ell_i + B = \sum_{i=1}^{n} v_i + B + (1-(1-\rho)^n)L = L$.

In the following, we show that the instance of the Odd-Even Partition problem is a Yes-instance if and only if there exists a feasible solution for the constructed instance of $1\,|\,pmtn, r_j\,|\,\sum C_j + E$ with total cost at most $B(e_1 - 1) + D + \sum_{i=1}^{n} a_{2i}$ with $D := (\sum_{i=1}^{n} r_{2i} + a_{2i}) + nd_2 - \sum_{i=1}^{n}(n-i)a_{2i-1}$.

**Lemma 3.29** In an optimal solution for the constructed instance we reserve exactly $B$ time slots in interval $I_1$.

***Proof.*** We know that every feasible solution has to reserve at least $B$ time slots in $I_1$, since $d_2 - s_2 = B$ and $\sum_{j=1}^{2n} p_j = 2B$.

Note that in any feasible solution $\sum_{j=1}^{2n} C_j < 2nd_2$. Therefore, if we reserve exactly $B$ time slots in $I_1$, the total cost is at most $Be_1 + 2nd_2 < (B+1)e_1$. However, if we reserve more than $B$ time slots in $I_1$, then the total cost is greater than $(B+1)e_1$. □

**Lemma 3.30** In an optimal solution for the constructed instance exactly one job of each odd-even pair $\{2i-1, 2i\}$, $i = 1, \ldots, n$, is finished at or before $r_{2i} + a_{2i}$.

***Proof.*** The proof is by strong induction on $i$. Consider the base case $i = 1$: Because of (3.3) and Lemma 3.29 we cannot finish both job 1 and job 2 before $r_2 + a_2$. If we finish none of them at or before $r_2 + a_2$, then $\sum_{j=1}^{2n} C_j > 2n\ell_1 =: z_1$. If we finish at least one of them not later than $r_2 + a_2$, then we have that $\sum_{j=1}^{2n} C_j \leq v_1 + (2n-1)L =: z_2$. We have

$$\begin{aligned} z_1 - z_2 &= 2n\rho L - v_1 - (2n-1)L \\ &= (2n(\rho-1)+1)L - v_1 \\ &= (1-\rho)L - v_1 \\ &> 0. \end{aligned}$$

Suppose the statement is true for $i = 1, \ldots, k-1$. Again, because of (3.3) and Lemma 3.29 we cannot finish both job $2k-1$ and job $2k$ before $r_{2k} + a_{2k}$. If

none of them is finished at or before $r_{2k} + a_{2k}$, then

$$\begin{aligned}
\sum_{j=1}^{2n} C_j \quad &> \quad (k-1)(L-B) \\
&\qquad +(v_1+\ell_1) \\
&\qquad +(v_1+\ell_1+v_2+\ell_2) \\
&\qquad +\cdots \\
&\qquad +(v_1+\ell_1+\cdots+v_{k-2}+\ell_{k-2}) \\
&\qquad +(2n-2(k-1))(v_1+\ell_1+\cdots+v_k+\ell_k) \\
&= \quad (k-1)(L-B) \\
&\qquad +(v_1+\ell_1)(2n-2(k-1)+(k-2)) \\
&\qquad +(v_2+\ell_2)(2n-2(k-1)+(k-3)) \\
&\qquad +\cdots \\
&\qquad +(v_{k-2}+\ell_{k-2})(2n-2(k-1)+1) \\
&\qquad +(v_{k-1}+\ell_{k-1})(2n-2(k-1)) \\
&\qquad +(v_k+\ell_k)(2n-2(k-1)) \\
&=: \quad z_1.
\end{aligned}$$

If we finish at least one of job $2k-1$ and job $2k$ not later than $r_{2k} + a_{2k}$, then

$$\begin{aligned}
\sum_{j=1}^{2n} C_j \quad &\leq \quad (k-1)L + v_1 + (v_1+\ell_1+v_2) \\
&\qquad +\cdots \\
&\qquad +(v_1+\ell_1+v_2+\ell_2+\cdots+v_{k-1}+\ell_{k-1}+v_k)+(2n-2k+1)L \\
&= \quad (2n-k)L + \sum_{i=1}^{k} v_i \\
&\qquad +(v_1+\ell_1)(k-1) \\
&\qquad +(v_2+\ell_2)(k-2) \\
&\qquad +\cdots \\
&\qquad +v_{k-1}+\ell_{k-1} \\
&=: \quad z_2.
\end{aligned}$$

We have

$$\begin{aligned} z_1 - z_2 &= (k-1)(L-B) - (2n-k)L + (2n-2k+1)\sum_{i=1}^{k} v_i - \sum_{i=1}^{k-1} v_i \\ &\quad +(2n-2k+2)\sum_{i=1}^{k} \ell_i - \sum_{i=1}^{k-1} \ell_i \\ &\geq L\left((1-\rho)^{k-1} - (2n-2k+2)(1-\rho)^k\right) - (k-1)B \\ &= L(1-\rho)^{k-1}\left(1-(2n-2k+2)(1-\rho)\right) - (k-1)B \\ &= \left(\sum_{i=1}^{n} v_i + B\right)\frac{2k-1}{(1-\rho)^{n-k}} - (k-1)B \\ &\geq kB \\ &> 0. \end{aligned}$$

This completes the proof. □

We now know that we finish exactly one job of every odd-even pair $\{2i-1, 2i\}$ not later than $r_{2i} + a_{2i}$. Let $J^{ini} := \{j \in J : C_j \leq d_1\}$ be the set of *initial jobs* and $J^{fin} := J \setminus J^{ini}$ be the set of *final jobs*.

**Lemma 3.31** In an optimal solution for the constructed instance at most one job is preempted, namely the final job of the odd-even pair $\{2n-1, 2n\}$.

***Proof.*** Consider an arbitrary set of initial jobs $J^{ini}$ with $p(J^{ini}) \leq B$. It is easy to see that after $s_2$ the final jobs are scheduled according to the Shortest Remaining Processing Time (SRPT) policy. Let $\delta := B - p(J^{ini})$ be the budget for shortening the processing time of the final jobs after $s_2$.

We first show that in an optimal solution the budget $\delta$ is invested in only one final job $j \in J^{fin}$, i.e., job $j$ is processed $\delta$ time units in interval $I_1$. By (3.3) and Lemma 3.29, we have $\delta < \min_{j \in J^{fin}} p_j$. Suppose the budget $\delta$ is invested in two final jobs $j_1$ and $j_2$, where $j_1$ is scheduled earlier than $j_2$. Then we can take the budget from $j_2$ and invest it in $j_1$, which results in a schedule with less scheduling cost, since the completion time of $j_2$ remains the same, but the jobs between $j_1$ and $j_2$ and also $j_1$ finish earlier. An iterative application of this argument shows the statement.

It remains to show that we invest the budget $\delta$ in one job of the odd-even pair $\{2n-1, 2n\}$, whichever is not in $J^{ini}$. Let

$$y_i := \begin{cases} p_{2i} & \text{, if } 2i-1 \in J^{ini} \\ p_{2i-1} & \text{, o.w.} \end{cases}$$

If we invest the budget $\delta$ to shorten $y_n$, we get

$$\sum_{j \in J^{fin}} C_j = s_2 + n\cdot(y_n - \delta) + (n-1)\cdot y_{n-1} + \cdots + 2y_2 + y_1 = s_2 - n\delta + \sum_{i=1}^{n} i y_i =: z_1.$$

Suppose we use the budget $\delta$ to shorten $y_k$ with $k < n$ and with respect to $J^{fin}$ job $k$ moves to position $n - (k' - 1)$ with $k \leq k' \leq n$. Then we have

$$\sum_{j \in J^{fin}} C_j = s_2 + \sum_{i=1}^{k-1} i y_i + \sum_{i=k'+1}^{n} i y_i + \sum_{i=k+1}^{k'} (i-1) y_i + k'(y_k - \delta) =: z_2.$$

We have

$$\begin{aligned} z_1 - z_2 &= (k' - n)\delta - \sum_{k+1}^{k'} y_i - (k' - k) y_k \\ &\leq (k' - k) y_k - (k' - k) y_k \\ &= 0. \end{aligned}$$

This completes the proof. □

Based on the previous lemmata, we can compute the total cost of an optimal solution just knowing the corresponding set of initial jobs.

**Lemma 3.32** Consider an arbitrary set of initial jobs $J^{ini}$ with $p(J^{ini}) \leq B$. The release dates of every odd-even pair $\{2i-1, 2i\}$ are set in such a way that choosing $2i-1$ instead of $2i$ as initial job results in a reduction of the scheduling cost of $a_{2i-1} - a_{2i}$. Here we assume that $p(J^{ini}) \leq B$ no matter which of the two jobs $2i-1$ and $2i$ is a initial job, i.e., the swap is feasible.

***Proof.*** Let again

$$y_i := \begin{cases} p_{2i} & \text{, if } 2i - 1 \in J^{ini} \\ p_{2i-1} & \text{, o.w.} \end{cases}$$

If $y_i = a_{2i}$, then the contribution of the $i$th pair to the scheduling cost is

$$C_{2i-1} + C_{2i} = r_{2i-1} + a_{2i-1} + s_2 + B - \sum_{k=1}^{i-1} y_k.$$

If $y_i = a_{2i-1}$, then

$$C_{2i-1} + C_{2i} = s_2 + B - \sum_{k=1}^{i-1} y_k + r_{2i} + a_{2i}.$$

Therefore, we have that

$$\sum_{j=1}^{2n} C_j = n \cdot (s_2 + B) - \sum_{k=1}^{n} (n-k) y_k + \sum_{j \in J^{ini}} r_j + a_j.$$

Let $z_{2i}$ and $z_{2i-1}$, respectively, be the total scheduling cost when job $2i$ and $2i-1$ is an initial job. By construction of the release dates, we get

$$\begin{aligned}\delta_i &:= z_{2i} - z_{2i-1}\\ &= (n-i+1)(a_{2i} - a_{2i-1}) + (r_{2i} - r_{2i-1})\\ &= (n-i+1)(a_{2i} - a_{2i-1}) + (n-i+2)(a_{2i-1} - a_{2i})\\ &= a_{2i-1} - a_{2i}.\end{aligned}$$ □

We are now ready to prove the following statement.

**Lemma 3.33** The given Odd-Even Partition instance is a Yes-instance if and only if there exists a feasible solution for the constructed instance of $1\,|\,pmtn, r_j\,|\,\sum C_j + E$ with total cost at most $B(e_1 - 1) + D + \sum_{i=1}^{n} a_{2i}$ with $D := (\sum_{i=1}^{n} r_{2i} + a_{2i}) + nd_2 - \sum_{i=1}^{n}(n-i)a_{2i-1}$.

***Proof.*** Consider the feasible solution in which we choose the small job $2i$ of every odd-even pair $\{2i-1, 2i\}$ as initial job. The total cost of this solution is exactly $Be_1 + D$. Based on this solution, we construct an optimal solution by feasibly swapping job pairs, i.e., we exchange job $2i$ by $2i-1$ in $J^{ini}$, so that the net change in scheduling cost is as large as possible. Using Lemma 3.32, this gives the following optimization problem, where $x_i$ is 1 if we swap the job pair $i$ and 0 otherwise:

$$\begin{aligned}\min\quad & D - (a_1 - a_2)x_1 - (a_3 - a_4)x_2 - \cdots - (a_{2n-1} - a_{2n})x_n\\ s.t.\quad & (a_1 - a_2)x_1 - (a_3 - a_4)x_2 - \cdots - (a_{2n-1} - a_{2n})x_n \le B - \sum_{i=1}^{n} a_{2i}\\ & x_i \in \{0,1\}.\end{aligned}$$

The constraint makes sure that we feasibly swap the jobs.

If the given Odd-Even Partition instance is a Yes-instance, then we find a solution $\bar{x}$ with

$$\begin{aligned}& a_1\bar{x}_1 + a_2(1-\bar{x}_1) + \cdots + a_{2n-1}\bar{x}_n + a_{2n}(1-\bar{x}_n) = B \qquad (3.4)\\ \iff\ & (a_1 - a_2)\bar{x}_1 - \cdots - (a_{2n-1} - a_{2n})\bar{x}_n = B - \sum_{i=1}^{n} a_{2i}.\end{aligned}$$

This means, there is a feasible solution for the constructed instance of $1\,|\,pmtn, r_j\,|\,\sum C_j + E$ with total cost at most $B(e_1 - 1) + D + \sum_{i=1}^{n} a_{2i}$.

If the given Odd-Even Partition instance is a No-instance, then there is no solution $\bar{x}$ satisfying (3.4). Therefore, there is no feasible solution for the constructed instance of $1\,|\,pmtn, r_j\,|\,\sum C_j + E$ with total cost at most $B(e_1 - 1) + D + \sum_{i=1}^{n} a_{2i}$. □

## 3.7 Conclusion and Open Problems

We considered a generalized scheduling model in which the processing of jobs causes some time-dependent cost in addition to the usual QoS measure. We devised optimal algorithms and best possible approximation algorithms for the scheduling objectives of minimizing the makespan and the sum of (weighted) completion times. Furthermore, we analyzed the influence of release dates on the problem complexity.

Cost-awareness in scheduling is highly relevant but hardly studied in scheduling theory. Taking the makespan objective as an example, we saw that incorporating the idea of cost-awareness into classical scheduling models does not necessarily result into an increase in the problem complexity. However, the resulting problems required new algorithmic ideas. For different objective functions and different problem types, such new cost considerations lead to new algorithmic challenges which are interesting and demanding and could be a rich source for new algorithmic techniques.

Our work with best possible algorithms for the makespan and the min-sum objective leaves open the approximability of the NP-hard scheduling problem $1 \,|\, pmtn, r_j \,|\, \sum(w_j)C_j + E$. Furthermore, it would be interesting to also understand $P \,|\, pmtn, r_j \,|\, C_{\max} + E$ from a complexity point of view, for which it is a priori not clear how many slots are reserved in an optimal schedule.

A different line of research shall assume *machine-individual* time-slot reservation cost, where a reservation decision is made for a time slot on each machine individually. We saw that it is not difficult to adapt a standard LP for optimally solving $R \,|\, pmtn, r_j \,|\, C_{\max}$ with fractional reservation cost. However, in our model where time slots can be reserved only integrally the resulting problems seems much harder.

# 4

# Cost-Aware Machine Scheduling under Data Uncertainty

Users of cloud computing services are offered rapid access to computing resources via the Internet. Cloud providers use different pricing options such as (i) time slot reservation in advance at a fixed price and (ii) on-demand service at a (hourly) pay-as-used basis. Choosing the best combination of pricing options is a challenging task for users, in particular, when the instantiation of computing jobs underlies uncertainty.

In this chapter, we propose and study a natural model for two-stage scheduling with reservation cost under uncertainty that captures such a resource provisioning and scheduling problem in the cloud. Reserving a time unit for processing jobs incurs some cost, which depends on when the reservation is made: a priori decisions, based only on distributional information, are much cheaper than on-demand decisions when the actual scenario is known. We consider both stochastic and robust versions of scheduling preemptive jobs with release dates on unrelated parallel machines and study the two objectives of minimizing the total sum of weighted completion times and the makespan.

After giving a problem definition in Section 4.1, we discuss related work and summarize our contributions in Section 4.2. In Section 4.3, we focus on the total weighted completion time objective in the stochastic setting. We analyze the complexity of the resulting two-stage problem in Section 4.3.1, devise LP-rounding based approximation algorithms in the Sections 4.3.2 to 4.3.4, and complement the investigation by improved approximation algorithms for special cases in Section 4.3.5. In Section 4.4, we give an approximation algorithm for the makespan objective in the stochastic setting. Then, we argue in Section 4.5 that at an arbitrarily small loss in the performance guarantee our approximation results hold for any arbitrary scenario distribution given by means of a black-box. Thereafter, in Section 4.6, we show that our techniques also yield approximation algorithms for two-stage robust scheduling. At the end of this chapter, we discuss technical aspects of our LP-rounding based technique in Section 4.7 and open problems in Section 4.8.

## 4.1 Problem Definition

In the underlying deterministic problem we are given a set of jobs $J = \{1, \dots, n\}$ and a set of machines $M = \{1, \dots, m\}$. Each job $j \in J$ is specified by a release date $r_j \geq 0$, before which $j$ cannot be processed, a machine-dependent processing time $p_{ij} \in \mathbb{N}$, the processing time when executing $j$ completely on machine $i \in M$, and a weight $w_j \geq 1$. In a feasible schedule each machine runs at most one job at the time and no job runs at more than one machine at the same time. A job may be preempted at any time and may resume processing on the same or any other machine. We assume that time is discretized into unit-size time slots. For ease of exposition let the completion time $C_j$ of a job $j \in J$ be the end of the unit-size time slot in which it actually completes. For every time slot, in which at least one machine is processing, a fixed *reservation cost* $c$ is paid. The objective is to minimize the sum of weighted completion times $\sum_{j \in J} w_j C_j$ or the makespan $C_{\max} := \max_j C_j$ plus total reservation cost.

In the *two-stage* version of this problem, the actual job set to be processed is one of a set $\mathcal{S}$ of possible scenarios. Any time slot can be reserved either in the first stage at cost $c$, and can then be used in every scenario, or in the second stage, for a specific scenario, at cost $\lambda c$, where $\lambda \geq 1$ is an inflation factor. We assume $\lambda$ to be defined by the scenario as well. The inflation factor together with the job set, $(\lambda_k \geq 1, J_k \subseteq J)$, make up a scenario $k \in \mathcal{S}$.

In the stochastic setting, we consider two models with respect to randomness. In the *polynomial-scenario model*, the distribution of $\mathcal{S}$ is given explicitly, i.e., each scenario $k \in \mathcal{S}$ is associated with a probability $\pi_k \in [0, 1]$ with $\sum_{k \in \mathcal{S}} \pi_k = 1$. In the *black-box model*, we have efficient access to an oracle that provides samples according to the unknown probability distribution with possibly exponentially many and dependent scenarios. In the robust setting, we restrict to the model with an explicit description of $\mathcal{S}$, called *discrete-scenario model*.

The overall goal in the stochastic setting is to minimize total expected reservation cost (in both stages) plus scheduling cost. In the robust setting, the overall goal is to minimize the maximum, over all scenarios, of reservation cost (in both stages) plus scheduling cost. That is, we do not make use of the given distributional information in the first stage, because we aim at finding a solution that minimizes the total cost in the worst-case.

## 4.2 Related Work and Contributions

**Related Work.** Preemptively scheduling unrelated machines to minimize the sum of (weighted) completion times, $R \,|\, pmtn, (r_j) \,|\, \sum (w_j) C_j$, is APX-hard [76] and admits a $(2 + \varepsilon)$-approximation [70]. The makespan problem $R \,|\, pmtn, r_j \,|\, C_{\max}$ can be solved in polynomial time [58].

Related to our scheduling problem with reservation cost is the problem of

cost-aware scheduling studied in the previous chapter. Our second-stage problem involves a special case with time slots having cost either 0 or some fixed amount.

With respect to the stochastic problem, our work is closer to two- or multi-stage stochastic versions of scheduling problems, see e.g. [3, 15], than to traditional *dynamic stochastic scheduling* [69], in which the algorithm's decision on processing a job or not crucially influences the information release. However, the former involve different scheduling problems than ours, and more importantly performance quality is assessed by computational experiments instead of rigorous worst-case analysis. The only work on approximation algorithms for a two-stage scheduling problem we are aware of is by Shmoys and Sozio [73]. They present a $(2+\varepsilon)$-approximation for two-stage stochastic throughput maximization on a single machine in which jobs can be deferred in the first stage gaining some profit.

The study of approximation algorithms for two-stage stochastic optimization problems was initiated in [31] with a polynomial-scenario model for a service-provision problem. Subsequently, next to [73] above, various two-stage stochastic versions of combinatorial optimization problems such as Set Cover, Network Design, Maximum Weight Matching, etc. were studied, see [80] for a nice overview on the earlier work. General frameworks for solving several two-stage stochastic combinatorial optimization problems approximately have been proposed in [43] and [74]. The cost-sharing based approach in [43] yields a 2-approximation for a two-stage stochastic scheduling problem without release dates on identical parallel machines [61]. It is not clear how to apply it when there are release dates.

In the black-box model, we adopt the *Sample Average Approximation (SAA) method* proposed in [55]. It replaces the distribution on the random parameters by its empirical distribution defined by samples from it. Under certain conditions, good approximate solutions are obtained by drawing only a polynomial number of samples and solving the resulting SAA problem instead [8, 20, 81].

In a two-stage setting, robust versions of multiple-scenario combinatorial optimization problems have been studied for covering and network design problems in [29, 34, 53]. We are not aware of any relevant scheduling work.

**Our Contribution.** We develop approximation algorithms for the stochastic and robust two-stage variants of classical scheduling problems. Our results rely on a new scheduling-tailored *time slot and job-set separation* procedure, which separates jobs into those processing exclusively on either first-stage reserved slots or second-stage reserved slots. It is inspired by [74] in which the idea of separating clients was introduced in the context of covering and network design problems. The separation in our setting is achieved through solving a generalization of the time-indexed unrelated machine scheduling LP [72] followed by an application of the slow-motion technique, proposed in [71] for min-sum single machine scheduling and extended later, among others, to unrelated machines scheduling in [70]. After separating, our rounding is applied independently to both separated in-

stances. The two (possibly overlapping) solutions are merged to a feasible joint solution. Carefully balancing the change in reservation and scheduling cost by exploiting properties of slow-motion and $\alpha$-points, the resulting procedure is proven to be an $(8+\varepsilon)$-approximation algorithm for the two-stage polynomial-scenario stochastic version of $R\,|\,pmtn, r_j\,|\,\sum w_jC_j$ (Section 4.3.4).

Our time slot and job-set separation procedure is based on a general result, which is interesting on its own in the polynomial-scenario model: Given a $\rho$-approximation algorithm for the special case in which slots are reserved only in the first stage, there is an $8\rho$-approximation algorithm for the two-stage model (Section 4.3.3). For this special case, we give a $\rho = (3+\varepsilon)$-approximation algorithm (Section 4.3.2).

Our techniques also yield a $(6+\varepsilon)$-approximation algorithm for the two-stage stochastic version of the makespan problem $R\,|\,pmtn, r_j\,|\,C_{\max}$ (Section 4.4).

Adopting the SAA framework, mentioned before, we apply our algorithms to arrive at a sampling-based $(8+\varepsilon)$-approximation algorithm for the min-sum problem and a $(6+\varepsilon)$-approximation for minimizing the makespan (Section 4.5). We notice that the work of [20, 81] leads to a first-stage reservation decision. It is not obvious in our model how to construct a good second-stage solution given a set of slots for free from the first-stage solution. In fact, considering this problem independently from the first-stage, it is unclear if a constant approximation exists. But even when considering the two stages jointly, the difficult part is to show how a second-stage solution for some scenario (not necessarily in the sample set) can be found and bounded by the SAA solution for the sample set.

Finally, we argue that our algorithms can be adopted for the min-max robust optimization model with a discrete set of scenarios (Section 4.6). For the min-$\sum w_jC_j$ problem, certain randomized steps of our algorithm must be replaced by deterministic ones losing a factor 2 in the approximation guarantee. For the robust makespan problem we derive a 2-approximation.

## 4.3 Polynomial-Scenario Model for Min-Sum Objective

Consider the two-stage stochastic version of $R\,|\,pmtn, r_j\,|\,\sum w_jC_j$ in the polynomial-scenario model, in which the set of scenarios $\mathcal{S}$ and its distribution are fully specified. For each scenario $k \in \mathcal{S} := \{1, \ldots, N\}$ the triple $(\pi_k, \lambda_k, J_k)$ describes its probability of occurring $\pi_k$, the inflation factor $\lambda_k$, and the set of jobs $J_k$.

We use a natural LP that generalizes and further relaxes the time-indexed LP for unrelated-machine scheduling [72]. To facilitate the exposition, we will present an LP with exponentially many variables and constraints and derive our algorithms based on its solution, even though we cannot expect to solve it in polynomial time. However, using the standard technique of time-intervals of

geometrically increasing size [72] we obtain polynomial-time algorithms loosing only a small factor in the performance. We discuss the technical aspects of this technique in Section 4.7.

To keep notation amenable, we re-index jobs, such that each job $j$ refers to a unique job-scenario combination, and we let $J := J_1 \cup \ldots \cup J_N$. We choose the time horizon $T = \max_{k\in\mathcal{S}, j\in J_k}\{r_j\} + \max_{k\in\mathcal{S}}\{\sum_{j\in J_k} \max_{i\in M} p_{ij}\}$, an obvious upper bound on any completion time in a reasonable schedule. Variables $x_t$ and $x_{kt}$ represent the first and second stage reservation decisions for time slot $[t, t+1)$. Let $y_{ijt}$ be the amount of time job $j$ is processed in interval $[t, t+1)$ on machine $i$. We refer to the following linear program and also its value as $LP$.

$$\min \sum_{t=0}^{T-1} cx_t + \sum_{k\in\mathcal{S}} \pi_k \left( \sum_{j\in J_k} w_j C_j^{LP} + \lambda_k c \sum_{t=0}^{T-1} x_{kt} \right) \tag{4.1a}$$

$$\text{s.t.} \sum_{j\in J_k} y_{ijt} \le x_t + x_{kt} \qquad \forall i \in M, k \in \mathcal{S}, 0 \le t \le T-1 \tag{4.1b}$$

$$\sum_{i\in M} y_{ijt} \le x_t + x_{kt} \qquad \forall j \in J, k \in \mathcal{S}, 0 \le t \le T-1 \tag{4.1c}$$

$$x_t + x_{kt} \le 1 \qquad \forall k \in \mathcal{S}, 0 \le t \le T-1 \tag{4.1d}$$

$$\sum_{t=r_j}^{T-1} \sum_{i\in M} \frac{y_{ijt}}{p_{ij}} = 1 \qquad \forall j \in J \tag{4.1e}$$

$$\sum_{t=r_j}^{T-1} \sum_{i\in M} (t+1) \cdot \frac{y_{ijt}}{p_{ij}} = C_j^{LP} \qquad \forall j \in J \tag{4.1f}$$

$$x_t, x_{kt}, y_{ijt} \in [0,1] \qquad \forall i \in M, j \in J, k \in \mathcal{S}, 0 \le t \le T-1 \tag{4.1g}$$

Constraints (4.1b),(4.1d),(4.1e),(4.1g) are self-explaining. With (4.1c) we ensure that no job is processed for more than the total amount reserved in $[t, t+1)$ guaranteeing non-parallelism. For (4.1f), consider an arbitrary schedule with $t_j = \max_t\{t | y_{ijt} > 0, i \in M\}$, then the *true completion time* of job $j$ in this schedule is $t_j + 1$, while $C_j^{LP}$ offers a lower bound. Thus, even if we enforce all variables to be integral, the corresponding ILP still gives a relaxation of our problem.

Given an LP solution $(x_t, x_{kt}, y_{ijt})$, we let $LP^r = \sum_t cx_t + c\sum_{k,t} \pi_k \lambda_k x_{kt}$ denote the reservation cost and we let $LP^s = \sum_{k,j\in J_k} \pi_k w_j C_j^{LP}$ denote the scheduling cost.

### 4.3.1 Complexity

The two-stage stochastic variant of $R \,|\, pmtn, r_j \,|\, \sum w_j C_j$ is NP-hard, since the underlying deterministic scheduling problem is already NP-hard. However, we show that taking the stochastic aspect into account increases the problem complexity by showing the following theorem. Note that the deterministic scheduling

problem $1\,|\,pmtn, r_j\,|\,\sum C_j$ is solvable in polynomial time by the Shortest Remaining Processing Time (SRPT) policy [5].

**Theorem 4.1** The two-stage stochastic variant of $1\,|\,pmtn, r_j\,|\,\sum C_j$ is NP-hard.

***Proof.*** The proof is by a reduction from the NP-complete Partition problem and is inspired by [62].

| Partition | |
|---|---|
| Given: | We are given positive integers $a_1, \ldots, a_n, B$ with $\sum_{i=1}^{n} a_i = 2B$. |
| Task: | Decide whether there exists a subset $A \subset \{1, \ldots, n\}$ with $\sum_{i \in A} a_i = B$. |

We construct the following instance of the two-stage stochastic variant of $1\,|\,pmtn, r_j\,|\,\sum C_j$. For every element $a_i$ we create two scenarios $k_i$ and $k'_i$, which both have exactly one job. The job of scenario $k_i$ is released at $r_i = \sum_{k<i}(3a_k + d)$ with $d = 2B+1$ and has processing time $p_i = a_i$. The job of scenario $k'_i$ is released at $r'_i = r_i + 2a_i$ and has also processing time $p'_i = a_i$. In addition to these $2n$ scenarios we have one scenario $k_{n+1}$ with also only one job that is released at $r_{n+1} = 0$ and has processing time $p_{n+1} = 3B$. All $2n+1$ scenarios have the same probability of occurrence $\pi = 1/(2n+1)$ and the same inflation factor $\lambda = \infty$. We set the unit reservation cost to $c = d(n-1) + 5B$ and scale the objective function with $2n+1$ so that we obtain $\sum_{t=0}^{T-1} c(2n+1)x_t + \sum_{k \in \mathcal{S}} C^k_{\max}$ where $x_t$ represents the decision whether we reserve time slot $[t, t+1)$ or not and $C^k_{\max}$ is the completion time of the single job in scenario $k$. In fact, now the unit reservation cost is $c(2n+1)$. We can set $T = r'_n + a_n = 6B + (n-1)d$, because any optimal solution does not reserve later slots. Observe that

$$LB^s := 8B + \sum_{i=1}^{n-1} 2(3a_i + d)(n-i)$$

is a lower bound on the scheduling cost for the jobs in scenarios $k_i, k'_i, i = 1, \ldots, n$ in any optimal schedule and $LB^s + 3B$ is a lower bound on the optimal total scheduling cost. We now show that the Partition instance is a Yes-instance if and only if there is a schedule for the constructed scheduling instance with total cost at most

$$C := 3Bc(2n+1) + 16B + d(n-1) + \sum_{i=1}^{n-1} 2(3a_i + d)(n-i).$$

Suppose the Partition instance is a Yes-instance, which means that we find a set $A \subset \{1, \ldots, n\}$ with $\sum_{i \in A} a_i = B$. We reserve $3B$ slots to enable the following schedule. The jobs of the scenarios $k_i$ with $i \in A$ are processed in $[r_i, r_i + a_i)$, whereas the jobs of the scenarios $k_i$ with $i \notin A$ and the jobs of the

scenarios $k'_i$ are processed in $[r'_i, r'_i + a_i)$. Note that this schedule requires $3B$ slots to which we can assign the processing of the job of scenario $k_{n+1}$. The total cost of this schedule is exactly $LB^s + \sum_{i \notin A} 2a_i + T = C$.

Suppose there is a schedule with total cost at most $C$. We first characterize the structure of an optimal schedule. The parameter $c$ is chosen in such a way that in any optimal schedule not more than $3B$ slots are reserved. A schedule that reserves more than $3B$ slots has total cost at least

$$(3B+1)c(2n+1) + LB^s + 3B > 3Bc(2n+1) + d(n-1) + 5B + LB^s + 3B = C,$$

which implies that such a schedule cannot be optimal. This observation has a crucial implication, namely that the job of scenario $k_{n+1}$ finishes at $T$.

The next observation is that the two jobs of the scenarios $k_i$ and $k'_i$ are completed before $r'_i + a_i + d$. Suppose not, then the total cost of this schedule is at least

$$3Bc(2n+1) + LB^S + d(n-1) + 6B + d > C,$$

which again implies that such a schedule is not optimal. This observation together with a standard interchange argument imply that the two jobs of scenario $k_i$ and $k'_i$ are completed at or before $r'_i + a_i$.

Consider the schedule in which for every scenario $k_i, i = 1, \ldots, n$, the corresponding job is processed in $[r'_i, r'_i + a_i)$. The total cost of this schedule is $C + 2B$ and it is obviously not optimal, because some jobs of the scenarios $k_i, i = 1, \ldots, n$, can be processed earlier to make use of all $3B$ reserved slots. In fact, if we process the job of scenario $k_i$ instead in $[r_i, r_i + a_i)$, then we save $2a_i$ in the objective function, and thus by postponing all jobs of the scenarios $k_i, i = 1, \ldots, n$, we lose in total $4B$ in the objective function. Hence, if there is a schedule with total cost at most $C$, then there must be a set $A := \{i : C^{k_i}_{\max} = r_i + a_i\} \subset \{1, \ldots, n\}$ such that $4B - \sum_{i \in A} 2a_i - (B - \sum_{i \in A} a_i) \leq 2B$ and $\sum_{i \in A} a_i \leq B$, which implies that $\sum_{i \in A} a_i = B$. We subtract $B - \sum_{i \in A} a_i$ on the left-hand side of the first inequality, because we have $B$ slots available and if $\sum_{i \in A} a_i < B$, we can reduce the completion time in the scenarios $k_i$ with $i \notin A$ by $B - \sum_{i \in A} a_i$. Hence, the given Partition instance is a Yes-instance. □

In the proof, we introduce for every scenario only one job and thus the two-stage stochastic variant of $1 \,|\, pmtn, r_j \,|\, C_{\max}$ is also NP-hard.

### 4.3.2 An Algorithm for First-Stage Reservation Only

Consider the special case of the two-stage stochastic version of $R \,|\, pmtn, r_j \,|\, \sum w_j C_j$ in which all reservation must be done in the first stage; as if all inflation factors $\lambda_k$ are excessive. We refer to it as *problem with first-stage reservation only*. A lower bound is given by $\mathrm{LP}_1$ obtained from the above LP by setting $x_{kt} = 0$, for all $k, t$. W.l.o.g. we omit the $\pi_k$

pre-multiplication in the objective function by assuming it to be incorporated into the weights $w_j, j \in J_k$.

We describe a procedure for rounding a fractional solution $(x_t, y_{ijt})$ of $\text{LP}_1$ to a feasible integer-value solution. We first round the first-stage decision on reserving slots $x_t$ by maintaining a feasible LP solution, and then we determine the actual schedule. In the first step, it is important to maintain a fractional scheduling solution in which the true completion time of a job $j$, i.e., $\max\{t+1 \mid y_{ijt} > 0, i \in M\}$, does not diverge too much from $C_j^{LP}$. To that end, we utilize the idea of *slow-motion*, proposed in [71] for single machine scheduling, and extended to unrelated machines scheduling in [70].

For $\alpha \in [0,1]$, let $C_j(\alpha)$ denote the earliest point in time in the LP solution in which job $j$ has completed an $\alpha$-fraction of its total processing requirement. We use the following link between $C_j(\alpha)$ an $C_j^{LP}$ adopted from [39], which is used for the analysis of randomized algorithms.

**Lemma 4.2 ([39])** $\int_0^1 C_j(\alpha)d\alpha = \sum_t \sum_i y_{ijt}/p_{ij} \cdot (t+1/2) = C_j^{LP} - 1/2.$

For deterministic algorithms, however, we use the following relation.

**Lemma 4.3 ([77])** $C_j^{LP} \geq \alpha + (1-\alpha) \cdot C_j(\alpha).$

**Slow-Motion.** Given a fractional solution $(x_t, y_{ijt})$ for $\text{LP}_1$ and $\beta \geq 1$, we construct a new $\beta$-expanded solution $(x_t^\beta, y_{ijt}^\beta)$ that we obtain by mapping every time point $t$ to $\beta t$. Then $\beta x_t$ is the amount of reservation for the interval $[\beta t, \beta(t+1))$, and $\beta y_{ijt}$ the amount of job $j$ scheduled during $[\beta t, \beta(t+1))$.

Obviously $(x_t^\beta, y_{ijt}^\beta)$ is a new feasible solution to $\text{LP}_1$, which over-schedules each job by a fraction of $\beta - 1$. We simply delete the over-scheduled part and apply a lemma given in [71] that upper bounds the completion times of jobs in the expanded solution. We directly adopt their result to our requirement of job completion times being rounded up to integer values.

**Lemma 4.4 ([71])** The completion time of job $j$ in the expanded solution is at most $\lceil \beta C_j(1/\beta) \rceil$.

**Rounding time slot reservation.** Given any fractional solution $(x_t, y_{ijt})$, e.g. the expanded $\text{LP}_1$ solution, we show how to round the fractional reservation $x_t$ to 0 or 1 so that the number of slots reserved will not be much higher than $\sum_t x_t$ but sufficiently large to accommodate all workload. We reassign the workload to reserved slots ensuring that the completion times remain relatively small.

We first apply a standard rounding technique, which we call *accumulated reservation:* Let $X_t = \sum_{h=0}^{t} x_h$, for $t \in \{0, 1, \ldots, T-1\}$ and $X_{-1} = 0$. We set $\bar{x}_t = 1$, i.e., we reserve time slot $[t, t+1)$, if $\lfloor X_{t-1} \rfloor \leq \lfloor X_t \rfloor - 1$, and set $\bar{x}_t = 0$ otherwise. In total, we reserve $\lfloor \sum_t x_t \rfloor$ slots this way. To ensure sufficiently reserved time capacity we do an *extra reservation:* if $\bar{x}_t = 1$ and $\bar{x}_{t+1} = 0$ for

some $t$, we reserve additionally the slot $[t+1, t+2)$. The number of extra reserved time slots is no more than the number of accumulatively reserved slots.

**Proposition 4.5** Given a fractional solution $x_t$, the total cost for rounding it to an integral time slot reservation $\bar{x}_t$ is $c\sum_{t=0}^{T-1}\bar{x}_t \le 2c\lfloor\sum_{t=0}^{T-1}x_t\rfloor \le 2LP^r$.

Our reservation policy creates intervals $I_1, I_2, \ldots$ of consecutive reserved time slots, each of them starting with a set of accumulatively reserved time slots and ending with a single extra reserved time slot.

**Lemma 4.6** Every interval $I_h = [\underline{t}_h, \bar{t}_h+2)$ has enough capacity to accommodate all workload $y_{ijt}$ assigned to time units $[\bar{t}_{h-1}+1, \bar{t}_{h-1}+2), \ldots, [\underline{t}_{h+1}-1, \underline{t}_{h+1})$.

***Proof.*** Consider interval $I_h$. Its last time slot $[\bar{t}_h+1, \bar{t}_h+2)$ is the extra reserved time slot. The total number (capacity) of the time slots in $I_h$ is $\lfloor X_{\bar{t}_h}\rfloor - \lfloor X_{\bar{t}_{h-1}}\rfloor + 1$. By definition of our accumulative reservation and according to constraints (4.1b) and (4.1c), the total workload in terms of $y_{ijt}$ in the interval $[\bar{t}_{h-1}+1, \underline{t}_{h+1})$ is bounded by

$$\sum_{t=\bar{t}_{h-1}+1}^{\underline{t}_{h+1}-1} x_t = \sum_{t=0}^{\underline{t}_{h+1}-1} x_t - \sum_{t=0}^{\bar{t}_{h-1}} x_t \le X_{\underline{t}_{h+1}-1} - \lfloor X_{\bar{t}_{h-1}}\rfloor \le \lfloor X_{\bar{t}_h}\rfloor - \lfloor X_{\bar{t}_{h-1}}\rfloor + 1. \qquad \square$$

The lemma implies that none of the workload $y_{ijt}$, fractionally assigned to time slots up to time slot $[\bar{t}_h, \bar{t}_h+1)$, needs to be done later than $\bar{t}_h+2$ if appropriately reassigned. In particular, this holds for the last reserved interval, i.e., all jobs in all scenarios will have been processed. Even stronger, workload assigned to the time slots $[\bar{t}_h+1, \bar{t}_h+2), \ldots, [\underline{t}_{h+1}-1, \underline{t}_{h+1})$ can be done within interval $I_h$, unless the release date of some job $j$ is larger than $\bar{t}_h+1$, preventing it to be scheduled in $I_h$.

**Reassigning workload.** Given a (not necessarily feasible) solution with fractional scheduling variables $y_{ijt}$ and integer-valued reserved time slots $\bar{x}_t$, we describe a reassignment procedure to arrive at a feasible solution in which all workload $\bar{y}_{ijt}$ is assigned to reserved slots.

In increasing order of $t$ we claim a total fraction $x_t$ from time slot $[\underline{t}_h, \underline{t}_h+1)$ if $\bar{t}_{h-1}+1 \le t < \underline{t}_h$ for some $h$. All of the $y_{ijt}$ is moved into this claimed space and added to $\bar{y}_{ij\underline{t}_h}$. Otherwise if $[t, t+1) \in I_h$, and $t \ne \bar{t}_h+1$, then we claim $\frac{\lfloor X_t\rfloor - X_{t-1}}{X_t - X_{t-1}}x_t$ from $[t, t+1)$ and $\frac{X_t - \lfloor X_t\rfloor}{X_t - X_{t-1}}x_t$ from $[t+1, t+2)$. All of the $y_{ijt}$ is moved in equal proportions $\frac{\lfloor X_t\rfloor - X_{t-1}}{X_t - X_{t-1}}y_{ijt}$ and $\frac{X_t - \lfloor X_t\rfloor}{X_t - X_{t-1}}y_{ijt}$ into the claimed space and added, respectively, to $\bar{y}_{ijt}$ and $\bar{y}_{ij(t+1)}$.

This assignment leaves (unclaimed) capacity $\lceil X_{\bar{t}_h}\rceil - X_{\bar{t}_h}$ of the extra reserved time slot $[\bar{t}_h+1, \bar{t}_h+2)$, for each $h$. As a second reassignment step we remove all $y_{ijt}$ with $\bar{t}_{h-1}+1 \le t < \underline{t}_h$ and $j$ with $r_j \le \bar{t}_{h-1}+1$ from $\bar{y}_{ij\underline{t}_h}$ and add it to $\bar{y}_{ij\tau}$ where $\tau = \bar{t}_{h-1}+1$. This is feasible by Lemma 4.6.

**Lemma 4.7** Applying the reservation and reassignment procedures to a feasible solution $(x_t, y_{ijt})$ of $\mathrm{LP}_1$ increases the completion time of any job $j$ by at most 1.

***Proof.*** For every $t$ for which $y_{ijt} > 0$, there are two cases:
*Case 1*: $[t, t+1) \in I_h$, and $t \neq \bar{t}_h + 1$ for some $h$. Then by the assignment procedure $y_{ijt}$ is moved into $[t, t+1)$ and $[t+1, t+2)$, whence that part of job $j$ finishes at most 1 time unit later than in the unrounded solution. In particular, this holds for the last $t$ such that $y_{ijt} > 0$.

*Case 2*: $\bar{t}_{h-1} + 1 \leq t < \underline{t}_h$ for some $h$. If $r_j \leq \bar{t}_{h-1} + 1$, then $y_{ijt}$ is moved into $[\bar{t}_{h-1} + 1, \bar{t}_{h-1} + 2)$ and finishes earlier, or if $r_j > \bar{t}_{h-1} + 1$, then $y_{ijt}$ is moved into $[\underline{t}_h, \underline{t}_h + 1)$. However, since $p_{ij} \geq 1$ (viz. integer), by the shifted-reservation policy job $j$ cannot have been completed before $\underline{t}_h$, i.e., there must be a $t \geq \underline{t}_h$ and/or another $i$ such that $y_{ijt} > 0$. □

**The one-stage reservation algorithm.** Given an optimal solution to $\mathrm{LP}_1$, we apply slow-motion to expand the solution and the time slot reservation to obtain integral reservations to which we reassign the workload. It remains to specify the actual schedule for workload $\bar{y}_{ijt}$ within a time slot $[t, t+1)$ by ensuring that a job is not scheduled in parallel on multiple machines. This is essentially $R|pmtn|C_{max}$ in each time slot, which is polynomial-time solvable [58].

**Theorem 4.8** The one-stage reservation algorithm is a 3.5-approximation algorithm for two-stage stochastic scheduling on unrelated machines with first-stage reservation only.

***Proof.*** Consider an optimal solution to $\mathrm{LP}_1$. By slow-motion we derive a $\beta$-expanded solution with a reservation cost of $\beta LP^r$, and job $j$ completes at time $\lceil \beta C_j(1/\beta) \rceil$ by Lemmata 4.2 and 4.4. Applying the rounding of time slot reservation and then reassigning workload, Proposition 4.5 shows that the reservation cost becomes $2\beta LP^r$, while Lemma 4.7 ensures that job $j$ completes not later than $\lceil \beta C_j(1/\beta) \rceil + 1$. Thus, by choosing the expansion parameter $\beta$ at random according to the density function $f(\alpha) = 3\alpha^2$ where $\alpha = 1/\beta \in [0, 1]$, the total cost for reservation and scheduling can be bounded by:

$$\begin{aligned} 2LP^r \int_0^1 \frac{1}{\alpha} f(\alpha) d\alpha \;\; + \;\; & \sum_j w_j \int_0^1 (\lceil C_j(\alpha)/\alpha \rceil + 1) f(\alpha) d\alpha \\ \leq \;\; & 3(LP^r + LP^s) + 1/2 \sum_j w_j. \end{aligned}$$

Obviously, $\sum_j w_j \leq \sum_j w_j C_j^{LP}$, since $C_j^{LP} \geq 1$, which implies that the algorithm produces a solution with objective value bounded by $3LP^r + 3.5LP^s$. □

A refinement of the algorithm and its analysis give an improved bound.

**Theorem 4.9** There exists a $(3+\varepsilon)$-approximation algorithm for two-stage stochastic scheduling on unrelated machines with first-stage reservation only.

***Proof.*** Recall that we have an algorithm which outputs a feasible schedule of cost $3(LP^r + LP^s) + 1/2\sum_j w_j$, where the additive term $1/2\sum_j w_j$ comes from the fact that after rounding, the completion time of job $j$ becomes $\lceil \beta C_j(1/\beta)\rceil + 1$. The reader will have no difficulty in verifying that if we can get rid of the $+1$ term in the completion time, then the cost can be reduced to $3(LP^r + LP^s)$.

Consider the term $\lceil \beta C_j(1/\beta)\rceil + 1$. For any small constant $\varepsilon > 0$ such that $1/\varepsilon$ is an integer, if $\beta C_j(1/\beta) \geq 2/\varepsilon$, then $\lceil \beta C_j(1/\beta)\rceil + 1 \leq (1+\varepsilon)\beta C_j(1/\beta)$. Hence, we only need to consider those completion times where $\beta C_j(1/\beta) < 2/\varepsilon$.

In our 3.5-approximation algorithm, we first apply the slow-motion algorithm with parameter $\beta$ so as to derive an expanded solution $(x_t^\beta, y_{ijt}^\beta)$. Then we apply the reservation and reassignment procedures to this expanded solution. Now, however, we perform it in a slightly different way: After the slow-motion procedure, we apply the rounding time slot reservation procedure with accumulated reservation and extra reservation from this section as before and obtain a rounded reservation solution $\bar{x} \in \{0,1\}^T$. In addition to that we reserve the first $2/\varepsilon$ slots, regardless of the $\bar{x}_t$ values, i.e., we set $\bar{x}_t = 1$ for $t < 2/\varepsilon$. This is similar to the previous rounding time slot reservation procedure except that we now reserve all slots $[t, t+1)$ with $t < 2/\varepsilon$. Then we reassign the workload $y_{ijt}(\beta)$ in the following way. We do not reassign the workload from the interval $[0, 2/\varepsilon - 1)$. We start the reassigning workload procedure at slot $[2/\varepsilon - 1, 2/\varepsilon)$. It can be easily seen that the reservation cost of this modified procedure is at most $2\beta LP^r + 2/\varepsilon \cdot c$.

Let $OPT$ denote the optimal cost. If $OPT \geq 2/\varepsilon^2 \cdot c$, then the reservation cost is bounded by $2\beta LP^r + \varepsilon OPT$. When reassigning the workload $y_{ijt}(\beta)$, the completion time of job $j$ is at most $\lceil \beta C_j(1/\beta)\rceil$ if $\beta C_j(1/\beta) < 2/\varepsilon$, which follows from Lemma 4.4 and the fact that if $\beta C_j(1/\beta) < 2/\varepsilon$ holds, then the workload of job $j$ is not reassigned. If $\beta C_j(1/\beta) \geq 2/\varepsilon$, the completion time of job $j$ is at most $\lceil \beta C_j(1/\beta)\rceil + 1 \leq (1+\varepsilon)\beta C_j(1/\beta)$, which follows from Lemma 4.7. Thus the total cost is at most $(3+\varepsilon)LP^s + 3LP^r + 2/\varepsilon \cdot c \leq (3+2\varepsilon)OPT$. This shows that there is a $(3+\varepsilon)$-approximation algorithm if the optimal cost is sufficiently large.

The proof of the theorem is completed by the following lemma. □

**Lemma 4.10** There is a $\rho$-approximation algorithm with $\rho = 6/(12-\pi^2) + \varepsilon \approx 2.816 + \varepsilon$ for the two-stage stochastic scheduling problem on unrelated machines with only first-stage reservation if there exists a constant $\gamma$ such that the optimal cost is less than $\gamma \cdot c$.

***Proof.*** If $OPT < \gamma \cdot c$, then every optimal solution reserves not more than a constant number $\gamma$ of slots. With enumeration we can 'guess' which slot is reserved by an optimal solution. We geometrically cut the time horizon $[0, T)$ into $O(\log T)$ sub-intervals $I_u$ so that $I_u := [u, u+1)$ for small $u = O(1/\varepsilon)$, and

$|I_{u+1}|/|I_u| = 1 + O(\varepsilon)$ for large $u$. Then we guess for each interval $I_u$ how many of the slots reserved in an optimal solution belong to that interval, and this gives rise to $O(\log^\gamma T)$ different possibilities. With $O(\varepsilon)$ loss we can assume that the slots are always reserved at the end of $I_u$, which gives rise to a fixed first stage reservation $\bar{x}_t \in \{0, 1\}$.

We establish an LP in which the $y_{ijt}$'s are still variable, while $x_t = \bar{x}_t$ is fixed. Again let $LP^r$ and $LP^s$ be the reservation and scheduling costs, respectively. We have $LP^r + LP^s \leq (1+\varepsilon)OPT$ if we 'guessed' the correct slots that are reserved.

Now we apply the slow-motion algorithm to the optimal solution $(\bar{x}_t, y_{hjt})$ (notice again that here $\bar{x}_t$ is determined through enumeration) of LP in a slightly different way: We arbitrarily specify $\alpha \in (0, 1)$ (i.e., $\beta = 1/\alpha$) and expand the solution by $\lceil\beta\rceil$ times. This suggests that we reserve the interval $[\lceil\beta\rceil t, \lceil\beta\rceil t+\lceil\beta\rceil))$ by the amount of $\lceil\beta\rceil\bar{x}_t$, i.e., if $\bar{x}_t = 0$ then any slot of this interval is not reserved, while if $\bar{x}_t = 1$ each slot is reserved. In other words, slow-motion with parameter $\lceil\beta\rceil$ provides us with a new solution $(x^\beta_t, y^\beta_{ijt})$ where $x^\beta_t \in \{0, 1\}$. By deleting the over-scheduled part of each job we ensure that the completion time of job $j$ is at most $\lceil\lceil\beta\rceil C_j(1/\beta)\rceil$, and the reservation cost is bounded by $\lceil\beta\rceil LP^r$.

Given some density function $f(\alpha)$, the expected total cost is

$$\begin{aligned} & \int_0^1 \lceil 1/\alpha\rceil \cdot LP^r f(\alpha)d\alpha + \int_0^1 \sum w_j \lceil\lceil 1/\alpha\rceil C_j(\alpha)\rceil f(\alpha)d\alpha \\ \leq & \int_0^1 \lceil 1/\alpha\rceil \cdot LP^r f(\alpha)d\alpha + \int_0^1 \sum w_j(\lceil 1/\alpha\rceil C_j(\alpha) + 1) f(\alpha)d\alpha. \end{aligned}$$

Taking $f(\alpha) = \rho/\lceil 1/\alpha\rceil$ with $\rho = 6/(12 - \pi^2) \approx 2.816$, we obtain a $(2.816 + \varepsilon)$-approximation algorithm. □

### 4.3.3 A Generic Algorithm for Two-Stage Scheduling

We first give a simple algorithm that allows a black-box application of the one-stage reservation algorithm from the previous section to obtain the following general result.

**Theorem 4.11** Given a $\rho$-approximation algorithm for the two-stage stochastic problem with only first-stage reservation, there exists an $8\rho$-approximation algorithm for the two-stage stochastic problem with reservation in both stages.

The crucial ingredient is to separate the time slots and jobs to be considered for only first-stage or only second-stage reservation.

**Lemma 4.12** Given an optimal solution $(x_t, x_{kt}, y_{ijt})$ to the LP with objective value $LP^r + LP^s$, there exists a feasible solution $(x'_t, x'_{it}, y'_{hjt})$ satisfying the following **separation property**:

- Any time unit is reserved either in the first stage or in the second stage, or not at all; i.e., for all $t$, $x'_t > 0 \Rightarrow x'_{kt} = 0\ \forall k$.

- A job is scheduled either completely in slots reserved in the first stage, or completely in slots reserved in the second stage, i.e., $J = J_I \cup J_{II}$, s.t. $J_I = \{j \mid x'_t = 0 \Rightarrow y'_{ijt} = 0 \; \forall ijt\}$ and $J_{II} = \{j \mid \sum_k x'_{kt} = 0 \Rightarrow y'_{ijt} = 0 \, \forall ijt\}$.
- The objective value is at most $2LP^r + 4LP^s$.

***Proof.*** We first double the number of time units: for every time unit $[t, t+1)$ we obtain two time units $[2t, 2t+1)$ and $[2t+1, 2t+2)$. We reserve $x_t$ of the even slot $[2t, 2t+1)$, and $x_{kt}$ of the odd slot $[2t+1, 2t+2)$. We split $y_{ijt}$ accordingly, such that for each of the slots $[2t, 2t+1)$ and $[2t+1, 2t+2)$ constraints (1b) and (1c) are satisfied. Notice that in this way we have blown up the scheduling cost by a factor of 2, while the reservation cost remains the same. Furthermore, notice that every job is processed at least half either in odd slots or in even slots. Thus by doubling again each slot and reserving in each of the two the same fraction, we can enforce a job to be either entirely scheduled in slots that are reserved in the first stage, or in slots reserved in the second stage. □

***Proof (Theorem 4.11).*** Let $(x'_t, x'_{it}, y'_{ijt})$ be a feasible LP solution that satisfies the separation property and has objective value $Z' \leq 2LP^r + 4LP^s$. We show that the existence of a $\rho$-approximation algorithm for the problem with first-stage reservation only implies the existence of an algorithm that produces a feasible schedule for the two-stage problem with total cost at most $2\rho Z'$.

Since jobs are divided into $J_I$ and $J_{II}$ in the solution $(x'_t, x'_{it}, y'_{ijt})$, we denote by $Z'_I$ and $Z'_{II}$ their contributions in $Z'$ respectively: $Z' = Z'_I + Z'_{II}$. Consider scheduling $J_I$ with only first-stage reservation and let $Z_I$ be the optimal value of the corresponding LP. Similarly, let $Z_{II}$ be the optimal value of the LP for scheduling $J_{II}$ with only second-stage reservation. Clearly, $Z_I \leq Z'_I$ and $Z_{II} \leq Z'_{II}$. The $\rho$-approximation algorithm for the problem with only first-stage reservation for $J_I$ returns a feasible schedule of cost at most $\rho Z_I$. The problem of reserving and scheduling jobs only in the second stage can be separated into $N$ single scenario problems, each of which is like a first-stage reservation problem, we thus also get a feasible schedule of cost at most $\rho Z_{II}$ for $J_{II}$.

Now we need to merge the two schedules for $J_I$ and $J_{II}$. Notice that the two schedules may overlap in the sense that some slot is reserved in both schedules. To handle this we further double the two schedules. For the schedule of $J_I$, we double the time units and put whatever is scheduled in $[t, t+1)$ into the even slot $[2t, 2t+1)$, while for the schedule of $J_{II}$, we put whatever is scheduled in $[t, t+1)$ into the odd slot $[2t+1, 2t+2)$. Now the total cost of the merged solution is bounded by $2\rho(Z_I + Z_{II}) \leq 2\rho Z'$. □

Directly applying Theorem 4.11 gives us a $(24 + \varepsilon)$-approximation algorithm.

### 4.3.4 A Refined Two-Stage Algorithm

**Theorem 4.13** There is an $(8+\varepsilon)$-approximation algorithm for the two-stage stochastic variant of $R\,|\,pmtn, r_j\,|\,\sum w_jC_j$ in the polynomial-scenario model.

***Proof.*** Given an optimal LP solution $(x_t, x_{kt}, y_{ijt})$, we first apply slow-motion to get a $\beta$-expanded solution $(x^\beta_t, x^\beta_{kt}, y^\beta_{ijt})$. Then we apply time slot and job-set separation (Lemma 4.12) and obtain jobs and slots to be covered by either first-stage or second-stage reservation only. Then, we apply the technique of accumulative and extra reservation and reassign the workload *separately* on the slots reserved in the first and second stage. Here the last procedure must be carried out with caution so that after we separately round first and second stage reservation, they do not overlap. The following analysis gives the result.

Again, let $LP^r$ and $LP^s$ be the reservation and scheduling cost of $(x_t, x_{kt}, y_{ijt})$, respectively. Applying *slow-motion* with some parameter $\beta$ yields an expanded fractional solution $(x^\beta_t, x^\beta_{kt}, y^\beta_{ijt})$ with reservation cost at most $\beta LP^r$ and in which any job $j$ completes not later than the (fractional) time $\beta C_j(1/\beta)$ [71].

Next we apply *slot-partition*, i.e., we double the time horizon, reserve $x^\beta_t$ amount of the even slot $[2t, 2t+1]$ and $x^\beta_{kt}$ amount of the odd slot $[2t+1, 2t+2)$ for each scenario $k$. We then split $y_{ijt}$ accordingly (i.e., proportional to $x^\beta_t$ and $x^\beta_{kt}$) and schedule the split parts onto the two intervals $[2t, 2t+1]$ and $[2t+1, 2t+2)$, so that the constraints in LP are still satisfied. By doing so we obtain a new fractional solution in which the reservation cost is still bounded by $\beta LP^r$, and every job $j$ completes not later than the (fractional) time $2\beta C_j(1/\beta)$.

Next we perform *job-partition*, i.e., we double every time slot $[t, t+1)$ again, reserve in each of the two the same fraction and schedule the same amount. By doing so we know that the slots $[4t, 4t+1)$ and $[4t+1, 4t+2)$ can only be reserved in the first stage, and $[4t+2, 4t+3)$ and $[4t+3, 4t+4)$ can only be reserved in the second stage. We call them first stage slots and second stage slots, respectively. Notice that now every job is actually scheduled twice in the doubled solution, and we can remove the over-scheduled part so that every job is either entirely scheduled in first stage slots, or entirely scheduled in second stage slots. It is easy to see that, in this fractional solution, the total reservation cost is $2\beta LP^r$, and every job $j$ completes not later than the (fractional) time $4\beta C_j(1/\beta)$.

We now apply the procedures *rounding time slot reservation* and *reassigning workload*. We apply the rounding procedure separately on the first stage slots and second stage slots. Let $(\hat{x}_t, \hat{x}_{kt}, \hat{y}_{ijt})$ be the current solution, we may simply take it as a combination of two solutions, one for scheduling a subset of jobs on first stage slots, and the other for scheduling the remaining jobs on second stage slots. Consider the first stage slots with its fractional reservation, we perform *accumulated reservation* and *extra reservation* on it. More precisely, let $X_t = \sum_{h=0}^{t} \hat{x}_h$ and $X_{-1} = 0$. We set $\bar{x}_t = 1$ if $\lfloor X_{t-1} \rfloor \leq \lfloor X_t \rfloor - 1$ and reserve this slot, otherwise $\bar{x}_t = 0$. Since $x_h = 0$ for $h = 4t+2, 4t+3$, we know that in this way only first stage slots can be reserved. For the extra reservation, if $\bar{x}_{4t} = 1$ and $\bar{x}_{4t+1} = 0$,

we reserve additionally $[4t+1, 4t+2)$; or if $\bar{x}_{4t+1} = 1$ and $\bar{x}_{4t+4} = 0$, we reserve additionally $[4t+4, 4t+5)$.

Following the same proof of Lemma 4.6 and Lemma 4.7, we are able to derive an integral reservation for first stage slots in which the first stage reservation doubles, and $j$ completes not later than the (fractional) time $4\beta C_j(1/\beta) + 3$ if it is scheduled entirely in the first stage slots. Here we have the additive term $+3$ instead of $+1$, since in the extra reservation, the slot $[4t+4, 4t+5)$ has a distance of 3 slots from $[4t+1, 4t+2)$.

Similarly, we perform *accumulated reservation* and *extra reservation* for the second stage slots and we know that the second stage reservation cost doubles, and that any job $j$ scheduled entirely in second stage slots now also completes not later than the (fractional) time $4\beta C_j(1/\beta) + 3$. Thus, the overall reservation cost is bounded by $4\beta LP^r$. If we choose $\beta = 1/\alpha$ randomly according to $f(\alpha) = 2\alpha$, then the total cost is bounded by

$$\begin{aligned} \int_0^1 4/\alpha \cdot LP^r f(\alpha) d\alpha \quad &+ \quad \int_0^1 \sum_j w_j(\lceil 4/\alpha C_j(\alpha) \rceil + 3) f(\alpha) d\alpha \\ &\leq \quad 8(LP^r + LP^s). \end{aligned}$$

□

### 4.3.5 Improvements for Special Cases

We now consider the special case of two-stage stochastic single machine scheduling. First, we provide an optimal algorithm for the two-stage stochastic variant of $1 \,|\, | \sum w_j C_j$. Then, we consider the two-stage stochastic variant of $1 \,|\, pmtn, r_j \,|\, \sum w_j C_j$, which is NP-hard by Theorem 4.1, and give a $(2+\varepsilon)$-approximation algorithm for the setting in which all reservation must be done in the first stage and we present a $(3+\varepsilon)$-approximation algorithm for the general setting with two-stage reservation. Again, we perform LP-rounding similar to the rounding described in the previous sections. The LP given on p. 59 is again the basis for that, but now we ignore the index $i$ for the machines. We shall simplify some rounding steps and reassign the workload by using a list scheduling policy.

**Without Release Dates.** We show the following statement.

**Theorem 4.14** The two-stage stochastic version of $1 \,|\, | \sum w_j C_j$ can be solved in polynomial time.

***Proof.*** Let $p(J_k) := \sum_{j \in J_k} p_j$ be the total processing volume in scenario $k$. In the absence of release dates, any reasonable algorithm reserves an interval $[0, x)$, $x \in \mathbb{N}$, in the first stage, augments this in scenario $k$ by reserving additional $\max\{p(J_k) - x, 0\}$ slots after the time point $x$, and schedules the jobs $j \in J_k$

according to Smith's rule [78]. Note that now the scheduling cost is a scenario-dependent constant, which can be computed beforehand and can be ignored in the minimization problem, which looks as follows:

$$\begin{aligned} \min \quad & c \cdot x + \sum_{k=1}^{N} \pi_k \cdot c \cdot \lambda_k \cdot \max\{p(J_k) - x; 0\} \\ \text{s.t.} \quad & x \in \mathbb{N}. \end{aligned}$$

Observe that the objective function is convex in $x$ and thus we can use binary search to find an optimal $x$ in polynomial time. □

**With Release Dates and First-Stage Reservation Only.** We now give a $(2+\varepsilon)$-approximation algorithm for the two-stage stochastic variant of $1\,|\,pmtn, r_j\,|\,\sum w_j C_j$ with first-stage reservation only. We solve the LP (with fixed $x_{kt} = 0$) and obtain an optimal solution $(x_t, y_{jt})$ which we round as follows.

First, we apply the *slow motion technique* with parameter $\beta > 1$, which is chosen later, to obtain a solution $(x_t^\beta, y_{jt}^\beta)$. By Lemma 4.4, every job $j \in J$ completes in $(x_t^\beta, y_{jt}^\beta)$ not later than $\lceil \beta C_j(1/\beta) \rceil$. Second, we round the time slot reservation by only applying the *accumulated reservation procedure* without the extra reservation to obtain a rounded $\bar{x}^\beta$. Third, we *reassign the workload* for every scenario $k \in \mathcal{S}$ by using list scheduling in order of non-decreasing $C_j(1/\beta)$ in the original LP solution. That is, we order the jobs $j \in J_k$ in non-decreasing order of $C_j(1/\beta)$ and for simplicity, we assume that the jobs $j \in J_k$ are labeled according to this ordering by $1, \ldots, |J_k|$. List scheduling for scenario $k$ based on non-decreasing $C_j(1/\beta)$, $j \in J_k$, reassigns the workload as follows. Consider any time slot $[t, t+1)$. If $\bar{x}_t^\beta = 0$, then we leave this slot empty, since it is not reserved. Otherwise $\bar{x}_t^\beta = 1$, and we process in $[t, t+1)$ one unit of the first job in our list that is not finished yet and is released at or before $t$ and move on to the next time slot. If there is no such job, we leave the time slot $[t, t+1)$ empty in scenario $k$. To prove that this is a $(2+\varepsilon)$-approximation algorithm we need the following lemma.

**Lemma 4.15** Given a feasible LP solution $(x_t^\beta, y_{jt}^\beta)$, let $\bar{x}^\beta$ be the rounded time slot reservation after applying the accumulated reservation procedure without the extra reservation. For any $a, b \in \mathbb{N}$ and any positive integer $z \in \mathbb{N}$ it holds that if $\sum_{t=a}^{b} x_t^\beta \geq z$, then $\sum_{t=a}^{b} \bar{x}_t^\beta \geq z$.

***Proof.*** Notice that for any non-negative $x \in \mathbb{R}$ and $z \in \mathbb{N}$, we have $\lfloor x + z \rfloor = \lfloor x \rfloor + z$. This implies for $X_t^\beta = \sum_{h=0}^{t} x_h^\beta$ that

$$\lfloor X_b^\beta \rfloor = \lfloor X_{a-1}^\beta + \sum_{t=a}^{b} x_t^\beta \rfloor \geq \lfloor X_{a-1}^\beta \rfloor + z.$$

Therefore, the accumulated reservation procedure reserves at least $z$ slots in the interval $[a, b+1)$, which implies that $\sum_{t=a}^{b} \bar{x}_t^\beta \geq z$. □

We now prove the following theorem.

**Theorem 4.16** There is a $(2+\varepsilon)$-approximation algorithm for the two-stage stochastic variant of $1\,|\,pmtn, r_j\,|\,\sum w_j C_j$ with first-stage reservation only.

***Proof.*** Consider any scenario $k \in \mathcal{S}$. We first show that after list scheduling in order of non-decreasing $C_j(1/\beta)$ in the original LP solution, every job $j \in J_k$ completes at or before $\tau_j := \lceil \beta C_j(1/\beta) \rceil$. Note that we assume for simplicity that the jobs $j \in J_k$ are labeled according to non-decreasing $C_j(1/\beta)$ by $1, \ldots, |J_k|$. Suppose the statement holds for all jobs $j \leq \ell - 1$ where $1 \leq \ell \leq |J_k|$. Consider job $\ell \in J_k$.

By Lemma 4.4, every job $j \in J_k$ completes in $(x_t^\beta, y_{jt}^\beta)$ not later than $\tau_j$. Hence, we have $\sum_{t=0}^{\tau_\ell - 1} x_t^\beta \geq \sum_{j=1}^{\ell} p_j$. Using Lemma 4.15, we get $\sum_{t=0}^{\tau_\ell - 1} \bar{x}_t^\beta \geq \sum_{j=1}^{\ell} p_j$. Consider all reserved time slots before $\tau_\ell$, i.e., all slots $[t, t+1)$ before $\tau_\ell$ with $\bar{x}_t^\beta = 1$. We now consider two cases.

**Case 1.** No time slot reserved before $\tau_\ell$ is empty in scenario $k$ or occupied by a job $j \in J_k \setminus \{1, \ldots, \ell\}$. This implies that every slot reserved before $\tau_\ell$ is also used in scenario $k$ to process the jobs $1, \ldots, \ell$. Since we reserve at least $\sum_{j=1}^{\ell} p_j$ slots before $\tau_\ell$, which is necessary to process all jobs $1, \ldots, \ell$, we conclude that also job $\ell \in J_k$ completes at or before $\tau_\ell$.

**Case 2.** There exists a time slot reserved before $\tau_\ell$ that is empty in scenario $k$ or occupied by a job $j \in J_k \setminus \{1, \ldots, \ell\}$. Consider the last such time slot and denote it by $[t', t'+1)$. If job $\ell$ is released at or before $t'$, then this job must be finished before $t' \leq \tau_\ell$, by the construction of our list scheduling policy. Otherwise, job $\ell$ is released at or after $t'+1$. Let $J' \subseteq \{1, \ldots, \ell\}$ be the subset of jobs that are released at or after $t'+1$. It follows directly that $\sum_{t=t'+1}^{\tau_\ell - 1} x_t^\beta \geq \sum_{j \in J'} p_j$. Hence, we have $\sum_{t=t'+1}^{\tau_\ell - 1} \bar{x}_t^\beta \geq \sum_{j \in J'} p_j$ by Lemma 4.15. Now we apply the same argument as in the first case and it follows that also job $\ell$ is finished not later than $\tau_\ell$.

This implies that our algorithm returns a solution with objective function value at most

$$\beta LP^r + \sum_{k=1}^{N} \pi_k \sum_{j \in J_k} w_j \tau_j.$$

Thus, by choosing the expansion parameter $\beta$ at random according to the density function $f(\alpha) = 2\alpha$ where $\alpha = 1/\beta \in [0, 1]$ and using Lemma 4.2, we observe that the expected total cost is at most $2LP^r + 2LP^s = 2LP$. □

**With Release Dates and Two-Stage Reservation.** We now augment the algorithm for the special case with only first-stage reservation to a

$(3+\varepsilon)$-approximation algorithm for the general setting with two-stage reservation. As before, we solve the LP and obtain an optimal solution $(x_t, x_{kt}, y_{jt})$ and apply the slow motion technique with parameter $\beta > 1$, which is again chosen later, to obtain a solution $(x_t^\beta, x_{kt}^\beta, y_{jt}^\beta)$. By Lemma 4.4, every job $j \in J$ completes in $(x_t^\beta, x_{kt}^\beta, y_{jt}^\beta)$ not later than $\tau_j := \lceil \beta C_j(1/\beta) \rceil$. Then we round the time slot reservation to obtain rounded $\bar{x}^\beta$ and $\bar{x}_k^\beta$, $k \in \mathcal{S}$, as follows. Let $X_t^\beta := 2\sum_{h=0}^{t} x_h^\beta$ and $X_{kt}^\beta := 2\sum_{h=0}^{t} x_{kh}^\beta$ with $X_{-1}^\beta = X_{k,-1}^\beta = 0$. We round the first-stage reservation as follows for $t = 0, \ldots, \lceil \beta(T-1) \rceil$:

- If $\lfloor X_t^\beta \rfloor = \lfloor X_{t-1}^\beta \rfloor$, we temporarily set $\bar{x}_t^\beta = 0$.
- If $\lfloor X_t^\beta \rfloor = \lfloor X_{t-1}^\beta \rfloor + 1$, we permanently set $\bar{x}_t^\beta = 1$.
- If $\lfloor X_t^\beta \rfloor = \lfloor X_{t-1}^\beta \rfloor + 2$, we permanently set $\bar{x}_t^\beta = 1$ and find the largest $h$ with $0 \le h < t$ such that $[h, h+1)$ is not reserved yet ($\bar{x}_h^\beta = 0$ temporarily) and permanently set $\bar{x}_h^\beta = 1$.

Similarly, we round the second-stage reservation as follows for $t = 0, \ldots, \lceil \beta(T-1) \rceil$:

- If $\lfloor X_{kt}^\beta \rfloor = \lfloor X_{k,t-1}^\beta \rfloor$, we temporarily set $\bar{x}_{kt}^\beta = 0$.
- If $\lfloor X_{kt}^\beta \rfloor = \lfloor X_{k,t-1}^\beta \rfloor + 1$ and $\bar{x}_t^\beta = 0$, we permanently set $\bar{x}_{kt}^\beta = 1$.
- If $\lfloor X_{kt}^\beta \rfloor = \lfloor X_{k,t-1}^\beta \rfloor + 1$ and $\bar{x}_t^\beta = 1$, we find the largest $h$ with $0 \le h < t$ such that $[h, h+1)$ is not reserved yet ($\bar{x}_h^\beta = \bar{x}_{kh}^\beta = 0$ temporarily) and permanently set $\bar{x}_{kh}^\beta = 1$.
- If $\lfloor X_{kt}^\beta \rfloor = \lfloor X_{k,t-1}^\beta \rfloor + 2$, we find the largest $h_1 < h_2 \le t$ such that $\bar{x}_{h_1}^\beta = \bar{x}_{kh_1}^\beta = \bar{x}_{h_2}^\beta = \bar{x}_{kh_2}^\beta = 0$ and permanently set $\bar{x}_{kh_1}^\beta = \bar{x}_{kh_2}^\beta = 1$.

After rounding first- and second-stage reservation, we reassign the workload for every scenario $k \in \mathcal{S}$ again by using list scheduling in order of non-decreasing $C_j(1/\beta)$ in the original LP solution. For simplicity, we assume that the jobs $j \in J_k$ are labeled according to this ordering by $1, \ldots, |J_k|$.

To prove that this is a $(3+\varepsilon)$-approximation algorithm we need the following lemmata.

**Lemma 4.17** For any non-negative $u, v, c, d \in \mathbb{R}$ with $c + d \ge 1$, we have

$$\lfloor u + 2c \rfloor + \lfloor v + 2d \rfloor - \lfloor u \rfloor - \lfloor v \rfloor \ge \lfloor c + d \rfloor.$$

***Proof.*** If $\lfloor c + d \rfloor = 1$, we can assume w.l.o.g. that $c \ge 1/2$. Therefore, we have

$$\lfloor u + 2c \rfloor + \lfloor v + 2d \rfloor - \lfloor u \rfloor - \lfloor v \rfloor \ge \lfloor u \rfloor + 1 + \lfloor v \rfloor - \lfloor u \rfloor - \lfloor v \rfloor = 1 = \lfloor c + d \rfloor.$$

If $\lfloor c+d \rfloor \geq 2$, we have

$$\begin{aligned}\lfloor u+2c \rfloor + \lfloor v+2d \rfloor - \lfloor u \rfloor - \lfloor v \rfloor &\geq u+2c+v+2d-2-\lfloor u \rfloor - \lfloor v \rfloor \\ &\geq 2(c+d)-2 \geq \lfloor c+d \rfloor.\end{aligned}$$

□

**Lemma 4.18** Given a feasible LP solution $(x_t^\beta, x_{kt}^\beta, y_{jt}^\beta)$, let $\bar{x}^\beta$ and $\bar{x}_k^\beta$, $k \in \mathcal{S}$, be the rounded time slot reservations after applying the described rounding procedure. For any $a, b \in \mathbb{N}$ and any positive integer $z \in \mathbb{N}$ it holds that if $\sum_{t=a}^{b}(x_t^\beta + x_{kt}^\beta) \geq z$, then $\sum_{t=a}^{b}(\bar{x}_t^\beta + \bar{x}_{kt}^\beta) \geq z$.

***Proof.*** We prove this statement by induction. Consider any scenario $k \in \mathcal{S}$. Suppose $b-a=0$, then

$$\sum_{t=a}^{b}(x_t^\beta + x_{kt}^\beta) = x_a^\beta + x_{ka}^\beta \leq 1.$$

It follows that $z=1$ and $x_a^\beta + x_{ka}^\beta = 1$. We show that either $\bar{x}_a^\beta = 1$ or $\bar{x}_{ka}^\beta = 1$. Notice that either $x_a^\beta \geq 1/2$ or $x_{ka}^\beta \geq 1/2$. If $x_a^\beta \geq 1/2$, then $\lfloor X_a^\beta \rfloor - \lfloor X_{a-1}^\beta \rfloor \geq 1$ and thus $\bar{x}_a^\beta = 1$, by our rounding procedure. If, however, $x_{ka}^\beta \geq 1/2$, then $\lfloor X_{ka}^\beta \rfloor - \lfloor X_{k,a-1}^\beta \rfloor \geq 1$ and thus either $\bar{x}_a^\beta = 1$ or $\bar{x}_{ka}^\beta = 1$.

Suppose the statement holds for any $a, b$ with $b-a \leq \ell-1$ and $\ell \geq 1$. Consider the case that $b-a=\ell$. We now consider two cases.

**Case 1.** Suppose, $\bar{x}_a^\beta = 1$ or $\bar{x}_{ka}^\beta = 1$. Then $\sum_{t=a+1}^{b}(x_t^\beta + x_{kt}^\beta) \geq z-1$, since $x_t^\beta + x_{kt}^\beta \leq 1$ for any $t \in \{0, \ldots, \lceil \beta(T-1) \rceil\}$. By the induction hypothesis, we have $\sum_{t=a+1}^{b}(\bar{x}_t^\beta + \bar{x}_{kt}^\beta) \geq z-1$, which implies that $\sum_{t=a}^{b}(\bar{x}_t^\beta + \bar{x}_{kt}^\beta) \geq z$.

**Case 2.** Suppose, $\bar{x}_a^\beta = 0$ and $\bar{x}_{ka}^\beta = 0$. Notice that according to our rounding procedure we try to reserve $\lfloor X_b^\beta \rfloor - \lfloor X_{a-1}^\beta \rfloor$ many slots in the first stage in the interval $[a, b+1)$. That is, according to our rounding scheme we reserve at least $\lfloor X_b^\beta \rfloor - \lfloor X_{a-1}^\beta \rfloor$ many slots in the first stage in the interval $[a, b+1)$, since $\bar{x}_a^\beta = 0$. Similarly, we know that we reserve at least $\lfloor X_{kb}^\beta \rfloor - \lfloor X_{k,a-1}^\beta \rfloor$ many slots in the second stage in the interval $[a, b+1)$, since also $\bar{x}_{ka}^\beta = 0$. Therefore, we have

$$\sum_{t=a}^{b}(\bar{x}_t^\beta + \bar{x}_{kt}^\beta) \geq \lfloor X_b^\beta \rfloor + \lfloor X_{kb}^\beta \rfloor - \lfloor X_{a-1}^\beta \rfloor - \lfloor X_{k,a-1}^\beta \rfloor.$$

Applying now Lemma 4.17 with $u = X_{a-1}^\beta, v = X_{k,a-1}^\beta, c = \sum_{t=a}^{b} x_t^\beta$, and $d = \sum_{t=a}^{b} x_{kt}^\beta$, we get the result. □

We now prove the following theorem.

**Theorem 4.19** There is a $(3+\varepsilon)$-approximation algorithm for the two-stage stochastic variant of $1 \,|\, pmtn, r_j \,|\, \sum w_j C_j$.

***Proof.*** Using Lemma 4.18, the proof that every job $j \in J$ finishes not later than $\tau_j$ follows the same line of argumentation as given in the proof for Theorem 4.16. This implies that our algorithm returns a solution with objective function value at most

$$2\beta LP^r + \sum_{k=1}^{N} \pi_k \sum_{j \in J_k} w_j \tau_j.$$

Thus, by choosing the expansion parameter $\beta$ at random according to the density function $f(\alpha) = 3\alpha^2$ where $\alpha = 1/\beta \in [0,1]$ and using Lemma 4.2, we observe that the expected total cost is at most $3LP^r + 3LP^s = 3LP$. □

## 4.4 Polynomial-Scenario Model for Makespan Objective

In this section, we argue that our techniques from the previous section lead to the following result for the makespan objective.

**Theorem 4.20** There is a $(6+\varepsilon)$-approximation algorithm for the two-stage stochastic variant of $R \,|\, pmtn, r_j \,|\, C_{\max}$ in the polynomial-scenario model.

We prove this statement in the following. For the two-stage stochastic version of $R \,|\, pmtn, r_j \,|\, C_{\max}$ we obtain an LP relaxation by replacing the objective function by $\sum_{t=0}^{T-1} cx_t + \sum_{k \in \mathcal{S}} \pi_k \left( C_{\max}^{k,LP} + \lambda_k c \sum_{t=0}^{T-1} x_{kt} \right)$ and adding the constraint $C_{\max}^{k,LP} \geq C_j^{LP}$ for all $j \in J_k$ and for all $k \in \mathcal{S}$ in our LP on p. 59. We solve this LP and apply slow-motion with parameter $\beta$ to be determined later. Denote by $(x_t, x_{kt}, y_{ijt})$ the fractional solution obtained and adapt $T := \max\{t : x_t > 0 \text{ or } x_{kt} > 0 \text{ for some } k \in \mathcal{S}\} + 1$. We adopt our rounding procedure to derive an integral reservation as follows.

**Rounding first stage reservation.** We apply the accumulated reservation procedure (see Section 4.3.2) with some slight modification. Let $\gamma \geq 1$ be a parameter to be determined later. We round $x_t$ in such a way that for any time $t$, we reserve in total $\min\{\lfloor \gamma \sum_{h=t}^{T-1} x_h \rfloor, T-t\}$ slots during $[t,T)$. This can be achieved with the following iterative procedure starting at time $T-1$ and going backwards in time. Suppose we already reserve $\tau_{t+1} = \min\{\lfloor \gamma \sum_{h=t+1}^{T-1} x_h \rfloor, T-t-1\}$ slots during $[t+1,T)$, we then consider the interval $[t,T)$ with $\tau_t = \min\{\lfloor \gamma \sum_{h=t}^{T-1} x_h \rfloor, T-t\}$. If $\tau_{t+1} = \tau_t$, we do not reserve the slot $[t,t+1)$. Otherwise $\tau_t - \tau_{t+1} \geq 1$, we reserve $[t,t+1)$ and additionally the $\tau_t - \tau_{t+1} - 1$ latest slots in $[t+1,T)$ that are not yet reserved. Let $\bar{x}_t$ be the rounded first stage reservation.

**Rounding second stage reservation.** We now round the second stage reservation for each specific scenario $k$. Let $1 > \rho \geq 1/\beta$ be a parameter to be determined later. Obviously, at time $T_k = \max_{j \in J_k} \lceil \beta C_j(\rho) \rceil$ every job $j \in J_k$ is finished. In fact, every job is over-scheduled by a factor of $\rho \cdot \beta$ in the fractional solution $(x_t, x_{kt}, y_{ijt})$. We round the second stage reservation as follows. During $[0, T_k)$ and among all the slots that are not reserved in the first stage, we reserve in the second stage the latest slots such that in total we reserve

$$\min\{\sum_{h=0}^{T_k-1} \bar{x}_h + \lfloor \gamma \sum_{h=0}^{T_k-1} x_{kh} \rfloor, T_k\}$$

and we denote by $\bar{x}_{kt}$ the rounded second stage reservation.

It is easy to verify that the total reservation cost is bounded by $\gamma\beta LP^r$, where $LP^r$ is the reservation cost of the fractional solution *before* slow motion. Furthermore, the makespan for scenario $k$ is bounded by

$$T_k = \max_{j \in J_k} \lceil \beta C_j(\rho) \rceil \leq \lceil \beta \cdot (C_{\max}^{k,LP} - \rho)/(1-\rho) \rceil.$$

Here we use the relation given in Lemma 4.3.

**Reassigning workload.** We now argue that it is possible to reassign the workload. We have the following lemma.

**Lemma 4.21** Let $(x_t, x_{kt})$ be the LP solution after slow-motion and let $(\bar{x}_t, \bar{x}_{kt})$ be the rounded solution. We can feasibly reassign all the workload of scenario $k$ if $\sum_{t=t_0}^{T_k-1} \bar{x}_t + \bar{x}_{kt} \geq \sum_{t=t_0}^{T_k-1} x_t + x_{kt}$ for all $t_0 \in [0, T_k - 1)$.

***Proof.*** Let $\tau := \sum_{t=0}^{T_k-1} \bar{x}_t + \bar{x}_{kt}$. We prove the statement by induction on $\tau$. The base case $\tau = 1$ is obviously fulfilled, as we can feasibly reassign the workload of all the slots with $x_t + x_{kt} > 0$ to the only integrally reserved slot. Consider the induction step from $\tau - 1$ to $\tau$. Let $T'_k$ be the earliest time point where $\sum_{t=T'_k}^{T_k-1} \bar{x}_t + \bar{x}_{kt} = 1$ and let $t'$ with $T'_k \leq t' \leq T_k - 1$ be the time point where $\bar{x}_{t'} + \bar{x}_{kt'} = 1$. Starting with the time slot $[t'-1, t')$ and going backwards in time, we iteratively move (a portion of) the workload of the considered slot to $[t', t'+1)$ until the total capacity of $[t', t'+1)$, which is 1, is met. After this, we reduce the values of the variables $x_t$ and $x_{kt}$ according to the reassignment, i.e., if we move a $\delta$ fraction of the workload of slot $[t, t+1)$ to $[t', t'+1)$, then we decrease $x_t$ by $\delta x_t$ and $x_{kt}$ by $\delta x_{kt}$. Let $(\hat{x}_t, \hat{x}_{kt})$ be the updated fractional reservation with $\hat{x}_{t'} + \hat{x}_{kt'} = 1$ after reassignment. The condition that $\sum_{t=t_0}^{T'_k-1} \bar{x}_t + \bar{x}_{kt} \geq \sum_{t=t_0}^{T'_k-1} \hat{x}_t + \hat{x}_{kt}$ for all $0 \leq t_0 < T'_k - 1$ is again fulfilled, because of the following fact. Let $a := \sum_{t=t_0}^{T'_k-1} \bar{x}_t + \bar{x}_{kt}$, $b := \sum_{t=T'_k}^{T_k-1} \bar{x}_t + \bar{x}_{kt} = 1$, $c := \sum_{t=t_0}^{T'_k-1} x_t + x_{kt}$, $d := \sum_{t=T'_k}^{T_k-1} x_t + x_{kt}$ and $\hat{c} := \sum_{t=t_0}^{T'_k-1} \hat{x}_t + \hat{x}_{kt}$. We know that $a + b \geq c + d$ and

$b \geq d$, which implies that $a \geq c + d - 1$ and $1 \geq d$. Moreover, we know that $\hat{c} = \max\{0, c - (1 - d)\}$. If $\hat{c} = 0$, then $a \geq \hat{c}$ and if $\hat{c} = c - (1 - d) > 0$, then $a \geq \hat{c} + 1 - d + d - 1 = \hat{c}$. We use our induction hypothesis with the updated $T_k = T'_k$. □

In the following, we argue that our rounding procedure satisfies the condition of Lemma 4.21.

**Lemma 4.22** For any $t \in [0, T_k - 1)$, either

$$\sum_{h=t}^{T_k-1} \bar{x}_h \geq \sum_{h=t}^{T_k-1} x_h - \frac{\gamma \sum_{h=t}^{T_k-1} x_h - \lfloor \gamma \sum_{h=t}^{T_k-1} x_h \rfloor}{\gamma} \geq \sum_{h=t}^{T_k-1} x_h - 1/\gamma$$

or all slots in $[t, T_k)$ are reserved in the first stage.

***Proof.*** Notice that

$$\begin{aligned} \sum_{h=t}^{T_k-1} \bar{x}_h &= \min\{\lfloor \gamma \sum_{h=t}^{T-1} x_h \rfloor, T - t\} - \min\{\lfloor \gamma \sum_{h=T_k}^{T-1} x_h \rfloor, T - T_k\} \\ &\geq \min\{\lfloor \gamma \sum_{h=t}^{T-1} x_h \rfloor, T - t\} - \lfloor \gamma \sum_{h=T_k}^{T-1} x_h \rfloor. \end{aligned}$$

If $\min\{\lfloor \gamma \sum_{h=t}^{T-1} x_h \rfloor, T - t\} = T - t$, then we reserve all slots in $[t, T)$ in the first stage and thus also in $[t, T_k)$. Otherwise

$$\sum_{h=t}^{T_k-1} \bar{x}_h \geq \lfloor \gamma \sum_{h=t}^{T-1} x_h \rfloor - \lfloor \gamma \sum_{h=T_k}^{T-1} x_h \rfloor \geq \lfloor \gamma \sum_{h=t}^{T_k-1} x_h \rfloor.$$

It suffices to show that

$$\lfloor \gamma \sum_{h=t}^{T_k-1} x_h \rfloor \geq \sum_{h=t}^{T_k-1} x_h - \frac{\gamma \sum_{h=t}^{T_k-1} x_h - \lfloor \gamma \sum_{h=t}^{T_k-1} x_h \rfloor}{\gamma} = \frac{\lfloor \gamma \sum_{h=t}^{T_k-1} x_h \rfloor}{\gamma},$$

which is obvious, knowing that $\gamma \geq 1$. □

Let $b := \sum_{h=0}^{T_k-1} x_{kh}$. In the following, we consider two cases, namely $b \geq 1/\gamma$ and $b < 1/\gamma$, and argue that the condition of Lemma 4.21 is satisfied in both cases. We show the following lemma for the first case.

**Lemma 4.23** If $\gamma \geq 3$ and $b \geq 1/\gamma$, then $\sum_{h=t}^{T_k-1}(\bar{x}_h + \bar{x}_{kh}) \geq \sum_{h=t}^{T_k-1}(x_h + x_{kh})$ holds for any $t \in [0, T_k - 1)$ and any scenario $k \in \mathcal{S}$.

To prove Lemma 4.24, we need the following lemma.

**Lemma 4.24** If $\gamma \geq 3$ and $b \geq 1/\gamma$, then $\lfloor \gamma b \rfloor \geq b + 1/\gamma$.

***Proof.*** We distinguish two cases. If $b \geq \frac{\gamma+1}{\gamma(\gamma-1)}$, then $\lfloor \gamma b \rfloor \geq \gamma b - 1 \geq b + 1/\gamma$. Otherwise $b < \frac{\gamma+1}{\gamma(\gamma-1)}$, notice that for $\gamma \geq 3$ we have $\gamma + 1 \leq (\gamma-1)^2$, $b < \frac{\gamma+1}{\gamma(\gamma-1)} \leq \frac{\gamma-1}{\gamma}$. Moreover, $\gamma b \geq 1$ and hence $\lfloor \gamma b \rfloor - 1/\gamma \geq 1 - 1/\gamma \geq b$. □

***Proof (Lemma 4.23).*** Let $\mu \leq T_k - 1$ be the largest index such that $\bar{x}_\mu + \bar{x}_{k\mu} = 0$. Then for $h \geq \mu + 1$, it holds that $\bar{x}_h + \bar{x}_{kh} = 1$ and hence $\sum_{h=t}^{T_k-1}(\bar{x}_h + \bar{x}_{kh}) \geq \sum_{h=t}^{T_k-1}(x_h + x_{kh})$ for any $t \geq \mu + 1$.

Now consider time points $t \leq \mu$. According to our second stage rounding, we reserve $\lfloor \gamma b \rfloor$ slots right before $T_k$ skipping the slots that are already reserved in the first stage. Therefore, we know that $\sum_{h=t}^{T_k-1} \bar{x}_{kh} = \lfloor \gamma b \rfloor$ for all $t \leq \mu$. According to Lemma 4.24, for $\gamma \geq 3$ and $b \geq 1/\gamma$, we have

$$\sum_{h=t}^{T_k-1} \bar{x}_{kh} \geq b + 1/\gamma \geq \sum_{h=t}^{T_k-1} x_{kh} + 1/\gamma.$$

According to Lemma 4.22, we have

$$\sum_{h=t}^{T_k-1} \bar{x}_h \geq \sum_{h=t}^{T_k-1} x_h - 1/\gamma.$$

By adding the two inequalities, we get $\sum_{h=t}^{T_k-1}(\bar{x}_h + \bar{x}_{kh}) \geq \sum_{h=t}^{T_k-1}(x_h + x_{kh})$. □

Now we consider the case that $b < 1/\gamma$. Notice that in our solution after slow motion every job is over-scheduled until $T_k$ by a factor of $\rho \cdot \beta$. We set parameters so that $\rho \cdot \beta = 1 + 1/\gamma$. Note that at most $b < 1/\gamma$ fraction of every job could be scheduled in the second stage in the fractional solution (we artificially split the workload into first stage workload $y_{ijt} \cdot x_t/(x_t + x_{kt})$ and second stage workload $y_{ijt} \cdot x_{kt}/(x_t + x_{kt})$). Hence, if we just discard the second stage reservation in the fractional solution $(x_t, x_{kt}, y_{ijt})$, i.e., we set $x'_{kt} = 0$ and $y'_{ijt} = y_{ijt} \cdot x_t/(x_t + x_{kt})$, then every job could still be finished before $T_k$. Let $(x_t, x'_{kt}, y'_{ijt})$ denote the modified solution. We show that the workload could be reassigned to $\bar{x}_t$.

Let $\mu \leq T_k - 1$ be the largest index such that $\bar{x}_\mu = 1$. We first observe that according to our rounding procedure $\sum_{h=\mu}^{T_k-1} x_h < 2/\gamma$, otherwise we reserve $\lfloor \gamma \sum_{h=\mu}^{T-1} x_h \rfloor - \lfloor \gamma \sum_{h=T_k}^{T-1} x_h \rfloor \geq 2$ slots within $[\mu, T_k)$, which is a contradiction.

We modify the fractional solution $(x_t, x'_{kt}, y'_{ijt})$ by setting $x'_\mu = \sum_{h=\mu}^{T_k-1} x_h < 2/\gamma \leq 1$, and move all the workload within $[\mu, T_k)$ into the slot $[\mu, \mu + 1)$. This is possible, because the processing time of each job is at least 1 on any machine, hence in the original solution $(x_t, x'_{kt}, y'_{ijt})$ no job is completely scheduled within $[\mu, T_k)$, implying that there is no release date after $\mu$. We are now ready to show the following lemma for the second case.

**Lemma 4.25** If $\gamma \geq 3$ and $b < 1/\gamma$, then $\sum_{h=t}^{T_k-1} \bar{x}_h \geq \sum_{h=t}^{T_k-1} x'_h$ holds for any $t \in [0, T_k - 1)$ and any scenario $k \in \mathcal{S}$.

***Proof.*** For $t > \mu$ the inequality is obviously true, since both sides are 0. For $t \leq \mu$ the left hand side is at least 1. If the right hand side is at most 1, then the inequality is also fulfilled. Therefore, it remains to consider the case that $\sum_{h=t}^{T_k-1} x'_h > 1$. Since

$$\sum_{h=t}^{T_k-1} \bar{x}_h \geq \lfloor \gamma \sum_{h=t}^{T-1} x'_h \rfloor - \lfloor \gamma \sum_{h=T_k}^{T-1} x'_h \rfloor \geq \lfloor \gamma \sum_{h=t}^{T_k-1} x'_h \rfloor \geq \sum_{h=t}^{T_k-1} x'_h$$

for $\sum_{h=t}^{T_k-1} x'_h > 1$ and $\gamma \geq 2$, the inequality is also fulfilled. □

Overall, we need to minimize $\max\{\beta/(1-\rho), \gamma \cdot \beta\}$ subject to $\rho \cdot \beta = 1 + 1/\gamma$. We take $\gamma = 3, \beta = 2, \rho = 2/3$ and obtain Theorem 4.20.

**Remark.** Similar to the proof of Theorem 4.14, we can show that if all jobs are released at time 0 and the underlying deterministic makespan problem is solvable in polynomial time, then we can also solve the two-stage stochastic variant of it in polynomial time. This is done by a combination of known algorithms for the single-stage single-scenario problem and binary search for finding an optimal $x \in \mathbb{N}$ such that the time slots in $[0, x)$ are reserved in the first stage. This holds for example for the scheduling problem $R \,|\, pmtn \,|\, C_{\max}$.

## 4.5 The Black-Box Model

We now show that at the expense of another $\varepsilon$ our results for the two-stage stochastic min-sum and makespan problem hold for any arbitrary scenario distribution given by means of a black-box. Besides that, the problem is as before.

Given a first-stage reservation $\bar{x} \in \{0,1\}^T$, a lower bound on the second-stage cost for a scenario $k$ is as follows:

$$q(\bar{x}, k) = \min \left\{ \sum_{j \in J_k} w_j C_j^{LP} + \lambda_k c \sum_{t=0}^{T-1} x_{kt} \mid \text{(4.1b) - (4.1g)} \wedge x_t = \bar{x}_t \ \forall t \right\}.$$

Let $c(x)$ denote the cost of a (possibly fractional) reservation $x \in [0,1]^T$. Then the following gives a lower bound on our two-stage stochastic scheduling problem.

$$\min_{x \in [0,1]^T} f(x) = c(x) + E_{k \in \mathcal{S}} \left[ q(x, k) \right]. \tag{4.2}$$

For an unknown distribution in the black-box model we cannot solve this problem efficiently. However, using the SAA method [55] we can approximate it. We draw

a number $N$ of independent sample scenarios $1, \ldots, N$ from the black-box and solve the following sample average problem:

$$\min_{x \in [0,1]^T} \hat{f}(x) = c(x) + \frac{1}{N} \cdot \sum_{k=1}^{N} q(x,k)\,. \tag{4.3}$$

Notice that (4.3) is exactly the LP of Section 4.3 with all $N$ scenarios having probability $1/N$, and can thus be solved efficiently. It remains to determine the number of samples $N$ that is needed to guarantee a certain approximation. We can show that our LP in (4.2) can be cast as a stochastic LP of type required in [81] for obtaining such a result. To that end, we must be given $\lambda := \max_{k \in S} \lambda_k$.

**Lemma 4.26 ([81])** There is a polynomially bounded number $N$ such that any optimal solution $x^{LP}$ to the sample average problem (4.3) with $N$ samples satisfies $f(x^{LP}) \leq (1+\varepsilon) \min_x f(x)$ with high probability.

***Proof (Applicability of Lemma 4.26).*** We show that the LP in (4.2) can be cast as a stochastic LP of type required in [81]. In particular, we show that we can apply Theorem 5.2 in [81] that says that for any $\varepsilon, \gamma > 0, (\gamma \leq 1)$, with probability at least $1-\delta$, any optimal solution $\hat{x}$ to the sample average problem (for us (4.3)) constructed with $poly(\mathcal{I}, \lambda, 1/\gamma, \ln(1/\varepsilon), \ln(1/\delta))$ samples satisfies $f(\hat{x}) \leq (1+\gamma) \cdot \min_x f(x) + 6\varepsilon$. Here, $\mathcal{I}$ denotes the input size and $f$ represents the objective function of the following stochastic LP

$$\begin{aligned} \min_{x \in \mathcal{P} \subseteq \mathbb{R}^m_{\geq 0}} \quad & f(x) = c \cdot x + E_{k \in \mathcal{S}}\left[q(x,k)\right] \\ \text{where} \quad & q(x,k) = \min\big\{ c^k \cdot r_k + q^k \cdot s_k \mid \; r_k \in \mathbb{R}^m_{\geq 0}, s_k \in \mathbb{R}^n_{\geq 0}, \\ & \qquad\qquad D^k s_k + T^k r_k \geq j^k - T^k x \big\}\,. \end{aligned} \tag{4.4}$$

We now argue that we can cast (4.2) as such an LP. For this, we replace the constraints in (4.1d) by the following two constraints

$$\sum_{j \in J_k} y_{ijt} \leq 1 \qquad \forall i \in M, k \in \mathcal{S}, 0 \leq t \leq T-1 \tag{4.5}$$

$$\sum_{i \in M} y_{ijt} \leq 1 \qquad \forall j \in J_k, k \in \mathcal{S}, 0 \leq t \leq T-1, \tag{4.6}$$

and furthermore, we remove the upper bound on the variables $x_{kt}$. We call the resulting program $\text{LP}_{mod}$. It is easy to see that $\text{LP}_{mod}$ is a relaxation of our LP, but that they have the same set of optimal solutions, since $\text{LP}_{mod}$ has no incentive to choose $x_t + x_{kt} > 1$ or $x_{kt} > 1$ for some $t \in \{0, \ldots, T-1\}$ and $k \in \mathcal{S}$. In this sense, they are equivalent. We let $x$ be the vector of our first-stage reservation decisions, $r_k$ be the vector of our second-stage reservation decisions in scenario $k$, and $s_k$ be the vector of workload assignment variables in scenario$k$. Therefore,

$m = T$, $n = |J| \cdot |M| \cdot T$, and $\mathcal{P} = [0,1]^T$. We set the coefficients in the objective function $c, c^k$, and $q^k$, the constraint matrices $D^k$ and $T^k$ and the right-hand side $j^k$ accordingly.

As required in [81], we also need to show, that (a) $T^k \geq \mathbf{0}$ for every $k \in \mathcal{S}$, and (b) $E_{k\in\mathcal{S}}[q(x,k)] \geq 0$ for every $x \in \mathcal{P}$, and the primal and dual problems corresponding to $q(x,k)$ are feasible for every scenario $k \in \mathcal{S}$. Requirement (a) is obviously fulfilled and requirement (b) is also fulfilled, if we assume that $\lambda < +\infty$.

In order to turn the performance guarantee into a purely multiplicative one, we need a sufficient lower bound on $\min_x f(x)$. In [81], it is shown that under certain assumptions, one can obtain such a lower bound. We need to show that (c) $x = \mathbf{0} \in \mathcal{P}$, and (d) for every scenario $k \in \mathcal{S}$, either $q(x,k)$ is minimized by setting $x = \mathbf{0}$ or the total cost $c \cdot x + c^k \cdot r_k + q^k \cdot s_k \geq 1$ for any feasible solution $(x, r_k, s_k)$. Assumption (c) is fulfilled, as we can reserve nothing in the first stage, and assumption (d) is also fulfilled, because every non-empty scenario $k \in \mathcal{S}$ contains at least one job $j$ with weight $w_j \geq 1$. □

Based on this lemma we can obtain a good integral first-stage solution. We draw $N$ samples and solve problem (4.3). Let $(x_t^{LP}, x_{kt}^{LP}, y_{ijt}^{LP})$ be an optimal (fractional) solution. Applying our rounding technique (Section 4.3) we derive a solution $(\bar{x}_t, \bar{x}_{kt}, \bar{y}_{ijt})$ with $(\bar{x}_t, \bar{x}_{kt}) \in \{0,1\}$. We fix $\bar{x}_t$ as first-stage reservation.

The difficult part is to find a second-stage solution for some scenario (that is not necessarily in the sample set) and bound it by the LP solution for the sample set. The key is that our rounding procedure for the first stage reservation $x$ only depends on $x$ itself and is independent of the scheduling solution. Given $\bar{x}_t$ and a scenario $k$, we solve the resulting second-stage problem as follows: we solve the problem $\min_{x\in[0,1]^T} c(x^{LP}) + q(x^{LP}, k)$, which is again exactly the LP of Section 4.3 with a single scenario $k$, after fixing first-stage reservation at $x_t = x_t^{LP}$. Let $(x'_{kt}, y'_{ijt})$ be the optimal solution. Plugging in $x_t^{LP}$ and applying our rounding procedure on $(x_t^{LP}, x'_{kt}, y'_{ijt})$, we get a feasible schedule of total cost at most $(\rho+\varepsilon)(c(x^{LP}) + q(x^{LP}, k))$, with $\rho = 8$ for the min-sum objective and $\rho = 6$ for the makespan. And, most importantly, the first stage reservation $\bar{x}_t$ is consistent with our first-stage reservation. Using Lemma 4.26 we have in expectation a total cost of at most $(\rho+\varepsilon)f(x^{LP}) \leq (\rho + O(\varepsilon)) \min_x f(x) \leq (\rho+\varepsilon')Z^*$.

**Theorem 4.27** In the black-box model, there is a $(\rho+\varepsilon)$-approximation algorithm for two-stage stochastic variant of $R\,|\,pmtn, r_j\,|\sum w_j C_j$ ($\rho = 8$) and $R\,|\,pmtn, r_j\,|\,C_{\max}$ ($\rho = 6$), respectively.

## 4.6 Two-Stage Robust Scheduling

In the robust setting, we restrict to the model with an explicit description of the scenario set $\mathcal{S}$. The objective is now to minimize the worst-case total cost instead of the expected total cost. Notice that the LP relaxations, that our algorithms in Section 4.3 rely on, can be easily adopted.

Our approximation algorithms for the stochastic model are risk-averse, i.e., the performance guarantee holds for every scenario. Therefore, the techniques used for the stochastic model also apply to the discrete-scenario robust model. For the min-$\sum w_jC_j$ problem, certain randomized steps of our algorithm must be replaced by deterministic ones losing a factor 2 in the approximation guarantee. Such an adaptation is not needed for the robust makespan problem and we directly obtain again a $(6+\varepsilon)$-approximation algorithm. However, the makespan problem is much easier and we provide a simple 2-approximation algorithm.

**Theorem 4.28** For two-stage discrete-scenario robust scheduling with reservation cost, there is a $\rho$-approximation algorithm for the scheduling problems $R\,|\,pmtn, r_j\,|\,\sum w_jC_j$ $(\rho = 16+\varepsilon)$ and $R\,|\,pmtn, r_j\,|\,C_{\max}$ $(\rho = 2)$, respectively.

We first prove that there is a $(16+\varepsilon)$-approximation algorithm for the two-stage robust version of $R\,|\,pmtn, r_j\,|\,\sum w_jC_j$.

***Proof (Part 1 of Theorem 4.28).*** The proof is similar to the one for Theorem 4.13. The difference is that we apply the slow-motion technique deterministically with $\beta = 2$ and we have a different objective function.

We solve the modified LP and obtain an optimal solution $(x_t, x_{kt}, y_{ijt})$. Let $LP^{r,I} := \sum_{t=0}^{T-1} x_t$ and $LP^{r,k} := \lambda_k c \sum_{t=0}^{T-1} x_{kt}$ for all $k \in \mathcal{S}$. We apply the slow-motion technique with $\beta = 2$ and derive an expanded fractional solution $(x^\beta_t, x^\beta_{kt}, y^\beta_{ijt})$. We know that the reservation cost is increased by a factor of $\beta$, while job $j$ completes not later than the (fractional) time $\beta C_j(1/\beta)$. Then we apply the slot-partition procedure (see proof of Theorem 4.13), which does not increases the reservation cost, but might increase the completion time of a job by a factor of 2. Then we also apply the job-partition procedure (see proof of Theorem 4.13), which increases both the reservation cost and the completion time of a job by a factor of 2. In the end, we round the time slot reservation and reassign the workload, which at most doubles the reservation cost and increases the completion time of every job by at most 3 time units. Using Lemma 4.3, we can bound the total cost of the constructed solution by

$$\begin{aligned}
& 8 \cdot LP^{r,I} + \max_{k\in\mathcal{S}} \left( 8 \cdot LP^{r,k} + \sum_{j\in J_k} w_j \left( \lceil 8 \cdot C_j(1/2) \rceil + 3 \right) \right) \\
\leq\ & 8 \cdot LP^{r,I} + \max_{k\in\mathcal{S}} \left( 8 \cdot LP^{r,k} + \sum_{j\in J_k} w_j \left( 8 \cdot C_j(1/2) + 4 \right) \right) \\
\leq\ & 8 \cdot LP^{r,I} + \max_{k\in\mathcal{S}} \left( 8 \cdot LP^{r,k} + \sum_{j\in J_k} 16 \cdot w_j C^{LP}_j \right) \\
\leq\ & 16 \cdot LP.
\end{aligned}$$

□

Before we prove that there is a 2-approximation algorithm for the two-stage robust version of $R\,|\,pmtn, r_j\,|\,C_{\max}$, we show the following.

**Theorem 4.29** The two-stage discrete-scenario robust variant of $R \,|\, pmtn \,|\, C_{\max}$ is solvable in polynomial time.

***Proof.*** For every scenario $k \in \mathcal{S}$, we solve the deterministic problem and obtain a makespan $C^k_{\max}$. Let $C^*_{\max} := \max_{k\in\mathcal{S}} C^k_{\max}$. A lower bound on the optimal total cost is $c\lceil C^*_{\max}\rceil + C^*_{\max}$. Our algorithm reserves in the first stage all slots in the interval $[0, \lceil C^*_{\max}\rceil)$ and nothing in the second stage, since all scenarios can be scheduled using only first-stage reserved slots. The total cost of this algorithm matches the given lower bound. □

***Proof (Part 2 of Theorem 4.28).*** We reserve in the first stage a consecutive time interval starting at the maximum single-stage single-scenario makespan over all scenarios and apply in the second stage the non-release date relaxation.

Consider an optimal solution for the two-stage robust version of $R \,|\, pmtn, r_j \,|\, C_{\max}$ and let $C^k_{\max}$ be the makespan in this solution for scenario $k$. Furthermore, let $R^*_0$ be the optimal number of slots reserved in the first stage and $R^*_k$ be the optimal number of slots reserved in the second stage in scenario $k$.

First, we solve for every scenario individually the single-stage single-scenario problem $R \,|\, pmtn, r_j \,|\, C_{\max}$, which gives a makespan $C^{k,ind}_{\max}$ for every $k \in \mathcal{S}$. Let $C^*_{\max} := \max_{k\in\mathcal{S}} C^{k,ind}_{\max}$. It is easy to see that $\lfloor C^*_{\max}\rfloor$ is a lower bound on the optimal total cost and that no job is released later than $\lfloor C^*_{\max}\rfloor$. By Theorem 4.29, we know that the two-stage robust version without release dates can be solved in polynomial time and we also know that solving the problem without release dates gives a solution with total cost at most

$$c \cdot R^*_0 + \max_{k\in\mathcal{S}} \left(\lambda_k c \cdot R^*_k + C^k_{\max}\right).$$

Then we increase the makespan of every scenario by $\lfloor C^*_{\max}\rfloor$ and obtain a solution that has total cost of at most

$$\begin{aligned} & c \cdot R^*_0 + \max_{k\in\mathcal{S}} \left(\lambda_k c \cdot R^*_k + C^k_{\max} + \lfloor C^*_{\max}\rfloor\right) \\ = \; & c \cdot R^*_0 + \max_{k\in\mathcal{S}} \left(\lambda_k c \cdot R^*_k + C^k_{\max}\right) + \lfloor C^*_{\max}\rfloor \\ \leq \; & 2 \cdot OPT. \end{aligned}$$

□

## 4.7 Interval-Indexed LP-Relaxation

The LP from Section 4.3 is a time-indexed formulation and thus has an exponential number of decision variables and constraints. However, a $(1 + O(\varepsilon))$-approximate solution for fixed $\varepsilon > 0$ can be derived in polynomial time by using the standard technique of solving an interval-indexed LP instead [72]. In the following, we argue that this technique can be applied to the LP that is the basis for our algorithms.

We partition the time horizon $[0, T)$ into intervals of exponentially increasing size and introduce decision variables $x_u$, $x_{ku}$ and $y_{iju}$ for each interval $I_u$ instead of for each time slot $[t, t+1)$. Here $x_u$ and $x_{ku}$ represent the amount of time slots reserved in $I_u$, and $y_{iju}$ the amount of time job $j$ is processed on machine $i$ in interval $I_u$. To ensure that all intervals contain at least one unit-size time slot, i.e., the length $|I_u| \geq 1$ for all $u$, we set the first $O(1/\varepsilon)$ intervals to be unit-size. More precisely, we set $I_u := [u, u+1)$ for $u \in \{0, \dots, t_0 - 1\}$ with $t_0 := \lceil (1+\varepsilon)^{u_0} \rceil$ and $I_u := \left[ \lceil (1+\varepsilon)^{u-(t_0-u_0)} \rceil, \lceil (1+\varepsilon)^{u+1-(t_0-u_0)} \rceil \right)$ for $u \in \{t_0, \dots, U\}$, where $u_0$ and $U$ are chosen to be the smallest integers so that $|I_u| \geq 1$ for all $u \geq t_0$ and $\lceil (1+\varepsilon)^{U-(t_0-u_0)} \rceil \geq T$. Note that $t_0 \in O(1/\varepsilon)$ and that $U$ is polynomially bounded in the input size, hence we have a polynomial number of decision variables. Let $s_u$ and $e_u$, respectively, be the starting and ending time of interval $I_u$.

As before, we want to set up an LP relaxation. To do so, the following lemma is helpful.

**Lemma 4.30** With $(1+\varepsilon)^4$ loss, we can assume that each job's release date coincides with an $s_u$ for some $u \in \{0, \dots, U\}$.

***Proof.*** Note that we do not touch the first constant unit-size time slots, since for them the statement is trivially fulfilled, as we assume integer release dates.

Consider any feasible solution for the problem and any interval $I_u$ for $u \in \{t_0, \dots, U\}$. Postponing all the workload in $I_u$ to $I_{u+1}$ increases the objective by at most $(1+\varepsilon)^4$, because $\lceil (1+\varepsilon)^{u+1-(t_0-u_0)} \rceil / \lceil (1+\varepsilon)^{u-(t_0-u_0)} \rceil \leq (1+\varepsilon)^2$ for $u \geq t_0$ with our choice of $u_0$. By this, it is feasible to round any release date $r_j \in I_u$ to $s_{u+1}$. Applying this workload reassignment to all intervals $I_u$ with $u \in \{t_0, \dots, U-1\}$ shows the statement. □

Using this observation, we obtain the following interval-indexed LP relaxation to which we refer as $\mathrm{LP}_\varepsilon$.

$$\min\ c \cdot \sum_{u=0}^{U} x_u + \sum_{k=1}^{N} \pi_k \left( \sum_{j \in J_k} w_j C_j^{LP_\varepsilon} + \lambda_k c \cdot \sum_{u=0}^{U} x_{ku} \right) \tag{4.7a}$$

$$\text{s.t.} \sum_{j \in J_k} y_{iju} \leq x_u + x_{ku} \qquad \forall i \in M, k \in \mathcal{S}, 0 \leq u \leq U \tag{4.7b}$$

$$\sum_{i \in M} y_{iju} \leq x_u + x_{ku} \qquad \forall j \in J, k \in \mathcal{S}, 0 \leq u \leq U \tag{4.7c}$$

$$x_u + x_{ku} \leq e_u - s_u \qquad \forall k \in \mathcal{S}, 0 \leq u \leq U \tag{4.7d}$$

$$\sum_{\substack{u=0 \\ s_u \geq r_j}}^{U} \sum_{i \in M} \frac{y_{iju}}{p_{ij}} = 1 \qquad \forall j \in J \tag{4.7e}$$

$$\sum_{\substack{u=0 \\ s_u \geq r_j}}^{U} \sum_{i \in M} e_u \cdot \frac{y_{iju}}{p_{ij}} = C_j^{LP_\varepsilon} \qquad \forall j \in J \tag{4.7f}$$

$$x_u, x_{ku}, y_{iju} \in [0, e_u - s_u] \qquad \forall i \in M, j \in J, k \in \mathcal{S}, 0 \leq u \leq U \tag{4.7g}$$

**Lemma 4.31** $\mathrm{LP}_\varepsilon \leq (1+\varepsilon)^6\mathrm{LP}$.

***Proof.*** Consider any feasible solution of the time-indexed LP and use Lemma 4.30 to create a feasible solution for the time-indexed linear program LP′ with rounded release dates. We know that $\mathrm{LP}' \leq (1+\varepsilon)^4\mathrm{LP}$. Any feasible solution $(x'_t, x'_{kt}, y'_{ijt})$ of LP′ can be translated into a feasible solution $(x_u, x_{ku}, y_{iju})$ of $\mathrm{LP}_\varepsilon$ by aggregating the reservation and workload in the corresponding intervals. It is easy to see that the two solutions have the same reservation cost. Furthermore, we know that $C_j^{LP_\varepsilon} \leq (1+\varepsilon)^2 C_j^{LP'}$ for all $j$, because $\lceil(1+\varepsilon)^{u+1-(t_0-u_0)}\rceil / \lceil(1+\varepsilon)^{u-(t_0-u_0)}\rceil \leq (1+\varepsilon)^2$ for $u \geq t_0$ with our choice of $u_0$. □

Given that we are able to obtain an approximate solution for LP in polynomial time, directly applying our rounding procedure to the approximate solution again yields a pseudo-polynomial running time. We argue that every step of our rounding procedure, with some slight modification, can be carried out in polynomial time.

We start with an optimal solution of $\mathrm{LP}_\varepsilon$, say $(x_u, x_{ku}, y_{iju})$, with an objective value of at most $(1+\varepsilon)^6\mathrm{LP}$. We can interpret it as a feasible solution of LP such that for every interval $I_u$, the first $\lfloor x_u \rfloor$ slots are reserved in the first stage, followed by a possibly 'mixed' slot with $x_u - \lfloor x_u \rfloor$ fraction first stage reservation. The remaining fraction of such a mixed slot is then reserved in the second stage, followed by the remaining second stage reservation. In this way, the sequence of $x_t$ consists of $O(U)$ subsequences of consecutive 1's each possibly followed by a mixed slot and so does $x_{kt}$ for every $k$. We equally distribute the workload $y_{iju}$ of a job $j$ on machine $i$ within interval $I_u$ over the reserved slots in $I_u$, i.e., we set $y_{ijt} = y_{iju}/(x_u + x_{ku})$ for entirely reserved slots in $I_u$, and a $\gamma$ fraction of this for any slot that is reserved for only a $\gamma$ fraction. The key observation is that the consecutive 1's (in $I_u$) of the sequence $x_t$ or $x_{kt}$ represent identical slots with jobs scheduled in the same way. In this way, we can split the sequences $x_t$, $x_{kt}$ and $y_{ijt}$ into polynomially many subsequences and for every subsequence we only need to store the reservation and scheduling pattern of one slot and in addition the start and end time of this subsequence. We call such a representation of the sequences $x_t$, $x_{kt}$ and $y_{ijt}$ *regular*. Obviously, such a representation has a polynomial-size encoding. We now argue why our rounding steps can be carried out in polynomial time and why we retain regularity of the schedule representation.

Consider the slow-motion technique with parameter $\beta$ and consider any of the polynomially many subsequences of the regular representation of $x_t$, $x_{kt}$ and $y_{ijt}$. Say, the subsequence starts at time $t = a$ and ends with $t = b$, i.e, we consider the interval $[a, b+1)$. Expanding by $\beta$ yields a stretched subsequence that starts at $t = \beta a$ and ends with $t = \beta(b+1) - 1$ and in which the reservation and scheduling pattern is stretched accordingly. That is, we have $x_t^\beta = x_a$, $x_{kt}^\beta = x_{ka}$

and $y^{\beta}_{ijt} = y_{ija}$ for $\beta a \leq t \leq \beta(b+1) - 1$ if $\beta$ is integer, while if $\beta$ is fractional, then $x^{\beta}_{t}$, $x^{\beta}_{kt}$ and $y^{\beta}_{ijt}$ may become fractional for $t = \lfloor \beta a \rfloor$ and $t = \lfloor \beta(b+1) \rfloor$ and may interfere with an earlier or later stretched subsequence. Hence, when applying slow-motion to the sequence $x_t$, $x_{kt}$ and $y_{ijt}$, we only need to consider the starting and ending points of each subsequence and in total there are at most $O(U)$ such points. Applying slow-motion increases the number of subsequences with the same reservation and scheduling pattern in the regular representation of $x_t$, $x_{kt}$ and $y_{ijt}$ by at most $O(U)$. Note that also cutting the over-scheduled part of every job does not destroy regularity of the schedule, since we have a polynomial number of jobs.

Consider the rounding step of slot-partition, in which we double the time horizon such as to obtain that even slots, i.e., $[t, t+1)$ with $t$ even, are first stage reserved and odd slots are second stage reserved. Again, consider any of the polynomially many subsequences of the regular representation of $x_t$, $x_{kt}$ and $y_{ijt}$ which starts at time $t = a$ and ends with $t = b$. After doubling the time horizon, a new subsequence $(x'_t, x'_{kt}, y'_{ijt})$ is derived from $(x_t, x_{kt}, y_{ijt})$ such that $x'_{2t} = x_a$, $x'_{2t+1} = 0$, $x'_{k,2t} = 0$ and $x'_{k,2t+1} = x_{ka}$ for $t \in [a, b]$. The workload $y_{ijt}$ is split accordingly, as described in Section 4.3.4. That is, we only need to update the starting and ending time of every subsequence and we store the first two slots of every subsequence and this pattern is then repeated till the end of the subsequence.

Consider the job-partition procedure, in which we further double the whole solution so as to obtain a solution in which a job is either entirely scheduled in first stage reserved slots or entirely scheduled in second stage reserved slots. The argumentation that this technique can be carried out in polynomial time and why we retain regularity of the schedule representation is similar to the one given for slow-motion, but now we store the first four slots of every subsequence which define the whole reservation and scheduling pattern for the subsequence.

Finally we consider the procedure of accumulated and extra reservation together with the reassignment of the workload. For simplicity, we still denote the regular sequences derived after all the above procedures as $x_t$, $x_{kt}$ and $y_{ijt}$. We only argue on $x_t$ and the corresponding workload. The argumentation for $x_{kt}$ follows the same line, because we have split the schedule into first and second stage. To get a contiguous schedule we omit the second-stage slots. First, we simplify the schedule representation. Consider a subsequence with fractional reservation, i.e., $0 < x_t < 1$ for some $t$ in the subsequence. We aggregate all the reservation of this subsequence and shift it to the beginning of the subsequence so that we now have two subsequences, one with only fully reserved slots and potentially one single slot with fractional reservation. The total workload of this subsequence is reassigned accordingly, which is feasible, because no job is released within a subsequence. Otherwise, we cut a subsequence into polynomially many subsequences so that no job is released within a subsequence. Now, we only have subsequences in which all the first-stage slots are fully reserved and we have

single slots with fractional reservation. In the accumulative reservation step, we define $X_t = \sum_{h=0}^{t} x_h$ and reserve a slot if the integral part, $\lfloor X_t \rfloor$, increases by 1. Clearly, for every $x_t = 1$ the time slot $[t, t+1)$ is reserved. Hence, for each subsequence where $x_t = 1$ for all first-stage slots all of the slots are reserved and thus they form intervals of consecutively reserved time slots. Thus, applying the accumulative reservation step to the polynomially many slots that are fractionally reserved is sufficient. The placement of the extra reserved slots can also be done in polynomial time, since the rounding step gives a polynomial number of intervals with consecutively reserved slots.

Note that every slot in an interval of consecutively reserved time slots, i.e., $I_h \setminus [\bar{t}_h + 1, \bar{t}_h + 2)$, is assigned the same workload. Therefore, when reassigning the workload we only need to compute the claimed space from the first slot of every interval $I_h$ and the resulting workload profile for the second slot. The other slots within the interval follow the profile of the second slot. This computation can be done in polynomial time, because we have a regular solution. Note that the extra reserved time slots also need to be handled separately, because we push workload back if the release dates allow.

## 4.8 Conclusion and Open Problems

Inspired by the resource provisioning problem of cloud users, we proposed an optimization model that reflects two-stage decision processes in which computing resources must be reserved under uncertainty about the set of computational tasks. It leads to a new class of scheduling problems. We presented first results that suggest higher approximation complexity than their single-stage single-scenario versions. The quest for better approximations and/or lower bounds on the approximability is left for future research. We also leave open the approximability of the equivalent non-preemptive scheduling problems with release dates. Notice that the second-stage problem would not admit a constant approximation (see Chapter 3), unless P = NP, when considering it independently of the first stage problem. However, a two-stage algorithm may yield a constant-factor approximation. Our investigation also leaves open the computational complexity of the two-stage robust version of $1 \,|\, pmtn, r_j \,|\, C_{\max}$.

Another interesting variation of the problem arises if a user may reserve machines individually, possibly at machine-dependent rates. We note that, even if reservation costs are uniform over the machines, our proposed LP relaxation has a non-constant integrality gap in this case.

Furthermore, it would be interesting to incorporate the idea of time-varying reservation cost from Chapter 3 also into the two-stage model from this chapter and see whether and how algorithmic techniques can be translated.

# 5

# Scheduling Maintenance Jobs in Networks

In this chapter, we investigate the problem of scheduling the maintenance of edges in a network with the objective of preserving connectivity between two distinguished vertices of the network. This problem is motivated by the servicing and replacement in transportation and telecommunication networks which requires a well-planned schedule to minimize the performance loss through temporary outages. We distinguish the objectives of minimizing the total network disruption time and maximizing the total time that the network is connected.

Our contributions are optimal algorithms, results on the computational complexity and approximability for different variants of the problem and different network structures. In Section 5.1, we give a formal problem definition, followed by an overview of related work and our contributions in Section 5.2. In Section 5.3, we show that the preemptive problem can be solved optimally in polynomial time in arbitrary networks. However, any restriction on the job preemption makes the problem considerably harder. Limiting the preemption to integral points in time makes the problem NP-hard and even inapproximable in the minimization version. Fully disallowing preemption only increases the complexity further; in Section 5.4, we give strong lower bounds on the approximability. Furthermore, we give tight bounds on the power of preemption in Section 5.5, that is, the maximum ratio of the values of non-preemptive and preemptive optimal solutions. We show that it is non-constant, even on simple paths. Interestingly, in such a network setting the preemptive as well as the non-preemptive problem are known to be efficiently solvable, whereas we show in Section 5.6 that mixing both leads to an NP-hard problem. Section 5.7 concludes this chapter and gives an overview of open problems for future research.

## 5.1 Problem Definition

In the Connectivity problem, we are given a planning horizon $T$ and an undirected graph $G = (V, E)$ with two distinguished vertices $s^+, s^- \in V$. We as-

sume w. l. o. g that the graph is simple; we can replace a parallel edge $\{u, w\}$ by a new node $v$ and two edges $\{u, v\}, \{v, w\}$. Every edge $e \in E$ needs to undergo $p_e \in \mathbb{Z}_{\geq 0}$ time units of maintenance within the time window $[r_e, d_e]$ with $r_e, d_e \in \mathbb{Z}_{\geq 0}$, where $r_e$ is called the release date and $d_e$ is called the deadline of the maintenance job for edge $e$. An edge $e = \{u, v\} \in E$ that is maintained at time $t$, is not available at $t$ in the graph $G$. We consider preemptive and non-preemptive maintenance jobs. If a job must be scheduled non-preemptively then, once it is started, it must run until completion without any interruption. If a job is allowed to be preempted, then its processing can be interrupted at any time and may resume at any later time without incurring extra cost.

A *schedule* $S$ for $G$ assigns the maintenance job of every edge $e \in E$ to a single time interval (if non-preemptive) or a set of disjoint time intervals (if preemptive) $S(e) := \{[a_1, b_1], \ldots, [a_k, b_k]\}$ with $r_e \leq a_i \leq b_i \leq d_e$ for $i \in [k]$ and $\sum_{[a,b] \in S(e)} (b - a) = p_e$. If not specified differently, we let $T = \max_{e \in E} d_e$. We do not limit the number of simultaneously maintained edges.

For a given maintenance schedule, we say that the network $G$ is *disconnected at time $t$* if there is no path from $s^+$ to $s^-$ in $G$ at time $t$, otherwise we call the network $G$ *connected at time $t$*. The goal is to find a maintenance schedule for the network $G$ so that the total time where $G$ is disconnected is minimized (MinConnectivity). When designing approximation algorithms, it is also interesting to consider the maximization variant of the problem. In this case, we want to find a schedule that maximizes the total time where $G$ is connected (MaxConnectivity). To express that a statement in this chapter holds for both objectives, we simply refer to Connectivity.

## 5.2 Related Work and Contributions

**Related Work.** To the best of our knowledge, the Connectivity problem has not been studied yet. However, the concept of combining scheduling with network problems has been considered by different communities lately. Boland et al. [10–12] study the combination of non-preemptive arc maintenance in a transport network, motivated by annual maintenance planning for the Hunter Valley Coal Chain [13]. Their goal is to schedule maintenance such that the flow over time through the network is maximized. They show strong NP-hardness for their problem and describe various heuristics and IP-based methods to address it. Also, they show in [11] that in their non-preemptive setting, if the input is integer, there is always an optimal solution that starts all jobs at integer time points. Their model is slightly more general than ours in the sense that they have capacities on their network – thus, positive results carry over to our setting, but hardness does not if it relies on the capacities. In [10], they consider a variant of their problem, where the number of concurrently performable maintenances is bounded by a constant.

Bley, Karch and D'Andreagiovanni [9] study how to upgrade a telecommunication network to a new technology employing a bounded number of technicians. Their goal is to minimize the total service disruption caused by downtimes. A major difference to our problem is that there is a set of given paths that shall be upgraded and a path can only be used if it is completely upgraded or not upgraded. They give ILP-based approaches for solving this problem and show strong NP-hardness for a non-constant number of paths by reduction from the linear arrangement problem.

Nurre et al. [67] consider the problem of restoring arcs in a network after a major disruption, with restoration per time step being bounded by the available work force. Such network design problems over time have also been considered by Kalinowski, Matsypura and Savelsbergh [51].

In scheduling, minimizing the busy time refers to minimizing the amount of time for which a machine is used. Such problems have applications for instance in the context of energy management [65] or fiber management in optical networks [35]. They have been studied from the complexity and approximation point of view in [18, 35, 54, 65]. The problem of minimizing the busy time is equivalent to our problem in the case of a path, because there we have connectivity at a time point when no edge in the path is maintained, i. e., no machine is busy. Thus, the results of Khandekar et al. [54] and Chang, Khuller and Mukherjee [18] have direct implications for us. They show that minimizing busy time can be done efficiently for purely non-preemptive and purely preemptive instances, respectively.

The *power of preemption* is a commonly used measure for the impact of preemption in scheduling [16, 28, 72, 79]. Other terms used in this context include *price of non-preemption* [26], *benefit of preemption* [68] and *gain of preemption* [44].

**Our Contribution.** Our contributions are optimal algorithms, results on the computational complexity and approximability for different variants of the problem. We show that the preemptive problem variants can be solved optimally in polynomial time, whereas any restriction on the job preemption makes the problem considerably harder. For *preemptive* maintenance jobs, we show that Connectivity can be solved optimally in polynomial time in arbitrary networks (Theorem 5.1). This result crucially requires that we are free to preempt jobs at arbitrary points in time. As soon as this is limited in any way, the problem complexity increases.

Under the restriction that we can *preempt* jobs only at *integral points in time*, the problem becomes NP-hard. More specifically, unless P = NP, MaxConnectivity does not admit a $(2-\varepsilon)$-approximation algorithm for any $\varepsilon > 0$ in this case, and MinConnectivity is inapproximable (Theorem 5.4). This is true even for unit-size jobs. This complexity result is interesting and may be surprising, as it is in contrast to results for standard scheduling problems, without an underlying network. Here, the restriction to integral preemption typically does

not increase the problem complexity when all other input parameters are integral. However, the same question remains open in a related problem concerning the busy-time in scheduling, studied in [18, 19].

For *non-preemptive* instances, we establish that there is no $(c\sqrt[3]{|E|})$-approximation algorithm for MaxConnectivity for some constant $c > 0$ and that MinConnectivity is inapproximable even for parallel paths, unless P = NP (Theorems 5.5 and 5.6). On the positive side, we provide an $(\ell + 1)$-approximation algorithm for MaxConnectivity in general graphs (Theorem 5.9), where $\ell$ is the number of distinct latest start times (deadline minus processing time) for jobs.

We use the notion *power of preemption* to capture the benefit of allowing arbitrary job preemption. It is defined as the maximum ratio of the objective values of an optimal non-preemptive and an optimal preemptive solution. We show that the power of preemption is $\Theta(\log |E|)$ for MinConnectivity on a path (Theorem 5.10) and unbounded for MaxConnectivity on a path (Theorem 5.13). This is in contrast to other scheduling problems, where the power of preemption is constant, e. g. [28, 72].

On paths, we show an interesting result concerning *mixed* instances of Connectivity. We prove that these instances are weakly NP-hard (Theorem 5.14) and give a simple 2-approximation algorithm for mixed instances of MinConnectivity (Theorem 5.15). The hardness result is of particular interest, as both purely non-preemptive and purely preemptive instances can be solved efficiently on a path (see Theorem 5.1 and [54]).

## 5.3 Preemptive Scheduling

In this section, we consider Connectivity instances in which we allow jobs to be preempted.

**Theorem 5.1** Preemptive Connectivity on general graphs can be solved optimally in polynomial time.

We prove this result by establishing a linear program (LP) for MaxConnectivity, showing that it is a relaxation of preemptive Connectivity, and that any optimal solution to it can be turned into a feasible schedule with the same objective function value.

Let $\{0\} \cup \{r_e, d_e : e \in E\}$ be the set of all *relevant time points* and let $\mathcal{I} =: I_1 \dot\cup \ldots \dot\cup I_k$ be the partition of all unit-size time slots into consecutive intervals such that two consecutive relevant time points form the boundary of one interval in $\mathcal{I}$. Note that $k \leq 2|E|$. Define $w_i := |I_i|$ to be the length of interval $I_i$. In our linear program, we model connectivity during interval $I_i$ by an $(s^+, s^-)$-flow $x^{(i)}$, $i \in \{1, \ldots, k\}$. To do so, we add for every undirected edge $e = \{u, v\}$ two directed arcs $(u, v)$ and $(v, u)$. Let $A$ be the resulting arc set. With

each edge/arc we associate a capacity variable $y_e^{(i)}$, which represents the fraction of availability of edge $e$ in interval $I_i$. Hence, $1 - y_e^{(i)}$ gives the relative amount of time spent on the maintenance of edge $e$ in $I_i$. Additionally, the variable $f_i$ expresses the fraction of availability for interval $I_i$. We consider the following linear program.

$$\max \quad \sum_{i=1}^{k} w_i \cdot f_i \tag{5.1}$$

$$\text{s.t.} \quad \sum_{u:(v,u)\in A} x_{(v,u)}^{(i)} - \sum_{u:(u,v)\in A} x_{(u,v)}^{(i)} = \begin{cases} f_i & \forall\, i \in [k],\ v = s^+, \\ 0 & \forall\, i \in [k],\ v \in V \setminus \{s^+, s^-\}, \\ -f_i & \forall\, i \in [k],\ v = s^-, \end{cases} \tag{5.2}$$

$$\sum_{i:I_i \subseteq [r_e, d_e]} (1 - y_e^{(i)}) w_i \geq p_e \qquad \forall\, e \in E, \tag{5.3}$$

$$x_{(u,v)}^{(i)}, x_{(v,u)}^{(i)} \leq y_{\{u,v\}}^{(i)} \qquad \forall\, i \in [k],\ \{u,v\} \in E, \tag{5.4}$$

$$f_i \leq 1 \qquad \forall\, i \in [k], \tag{5.5}$$

$$x_{(u,v)}^{(i)}, x_{(v,u)}^{(i)}, y_{\{u,v\}}^{(i)} \in [0,1] \qquad \forall\, i \in [k],\ \{u,v\} \in E. \tag{5.6}$$

**Lemma 5.2** The given LP is a relaxation of the preemptive Connectivity problem on general graphs.

***Proof.*** Given a feasible maintenance schedule, consider an arbitrary interval $I_i$, $i \in \{1, \ldots, k\}$, and let $[a_1^i, b_1^i] \,\dot\cup\, \ldots \,\dot\cup\, [a_{m_i}^i, b_{m_i}^i] \subseteq I_i$ be all intervals where $s^+$ and $s^-$ are connected in interval $I_i$. We set $f_i = \sum_{\ell=1}^{m_i} (b_\ell^i - a_\ell^i)/w_i$ and set $y_e^{(i)}$ to the fraction of time where edge $e$ is not maintained in interval $I_i$. Note that (5.3) is automatically fulfilled, since we consider a feasible schedule. It is left to construct a feasible flow $x^{(i)}$ for the fixed variables $f_i$ and $y^{(i)}$ for all $i = 1, \ldots, k$.

Whenever the given schedule admits connectivity we can send one unit of flow from $s^+$ to $s^-$ along some directed path in $G$. Moreover, in intervals where the set of processed edges does not change we can use the same path for sending the flow. Let $[a, b] \subseteq I_i$ be an interval where the set of processed edges does not change and in which we have connectivity. Let $\mathcal{C}_i$ be the collection of all such intervals in $I_i$. Then, we send a flow $x_{[a,b]}^{(i)}$ from $s^+$ to $s^-$ along any path of total value $(b-a)/w_i$ using only arcs for which the corresponding edge is not processed in $[a, b]$. The flow $x^{(i)} = \sum_{[a,b]\in\mathcal{C}_i} x_{[a,b]}^{(i)}$, which is a sum of vectors, gives the desired flow. The constructed flow $x^{(i)}$ respects the flow conservation (5.2) and uses no arc more than the corresponding $y_e^{(i)}$, since flow $x^{(i)}$ is driven by the schedule. □

**Lemma 5.3** Any feasible LP solution can be turned into a feasible maintenance schedule at no loss in the objective function value.

***Proof.*** Let $(x, y, f)$ be a feasible solution of the given LP and let $\mathcal{P}^i := (P^i_1, \dots, P^i_{\lambda_i})$ be a path decomposition of the $(s^+, s^-)$-flow $x^{(i)}$ for an arbitrary interval $I_i := [a_i, b_i]$, $i \in \{1, \dots, k\}$, after deleting all flow from possible circulations. Furthermore, let $x(P^i_\ell)$ be the value of the $(s^+, s^-)$-flow $x^{(i)}$ sent along the directed path $P^i_\ell$. For each arc $a \in A$ we have that $\sum_{\ell \in [\lambda_i]: a \in P^i_\ell} x(P^i_\ell) = x^{(i)}_a$ by the definition of $\mathcal{P}^i$. Hence, we get $\sum_{\ell \in [\lambda_i]} x(P^i_\ell) = f_i \leq 1$ by using (5.5). We now divide the interval $I_i$ into disjoint subintervals to allocate connectivity time for each path in our path decomposition. More precisely, we do *not* maintain any arc $(u, v)$ (resp. edge $\{u, v\}$) contained in $P^i_\ell$, $\ell = 1, \dots, \lambda_i$, in the time interval

$$[a_i + \sum_{m=1}^{\ell-1} w_i \cdot x(P^i_m), a_i + \sum_{m=1}^{\ell} w_i \cdot x(P^i_m)] \text{ of length } w_i \cdot x(P^i_\ell).$$

Inequality (5.4) and $\sum_{\ell \in [\lambda_i]: a \in P^i_\ell} x(P^i_\ell) = x^{(i)}_a$ thereby ensure that by now the total time where edge $e$ does not undergo maintenance in interval $I_i$ equals at most $w_i \cdot y^{(i)}_e$ time units. By Inequality (5.3), we can thus distribute the processing time of the job for edge $e$ among the remaining slots of all intervals $I_i$, $i = 1, \dots, k$. For instance, we could greedily process the job for edge $e$ as early as possible in available intervals. Note that arbitrary preemption of the processing is allowed. By construction, we have connectivity on path $P^i_\ell$, $\ell = 1, \dots, \lambda_i$, for at least $w_i \cdot x(P^i_\ell)$ time units in interval $I_i$. Thus, the constructed schedule has total connectivity time of at least $\sum_{i=1}^{k} w_i \sum_{\ell=1}^{\lambda_i} x(P^i_\ell) = \sum_{i=1}^{k} w_i \cdot f_i$. □

The statement of Theorem 5.1 crucially relies on the fact that we may preempt jobs arbitrarily. However, if we allow jobs to be preempted only at integral points in time, that is, we allow only schedules $S$ such that all intervals in $S(e) := \{[a_1, b_1], \dots, [a_k, b_k]\}$, $e \in E$, have integral boundaries, then the complexity of the problem changes. We prove the following theorem later in more generality. In fact, it is a corollary of Theorem 5.5. Interestingly, it holds already for instances with unit-size jobs only.

**Theorem 5.4** MaxConnectivity with preemption only at integral time points is NP-hard and does not admit a $(2 - \varepsilon)$-approximation algorithm for any $\varepsilon > 0$, unless P = NP. Furthermore, MinConnectivity with preemption only at integral time points is inapproximable.

***Proof.*** See the proof of Theorem 5.5 below and set $t_1 = 0$, $t_2 = 1$, and $T = 2$.□

For unit-size jobs we can simplify the given LP by restricting to the first $|E|$ slots within every interval $I_i$. This, in turn, allows to consider intervals of unit-size, i.e., we have $w_i = 1$ for all intervals $I_i$, which affects constraint (5.3). However, one can show that the constraint matrix of this LP is generally not totally unimodular. We illustrate the behaviour of the LP with the help of the following

exemplary instance in Figure 5.1, in which all edges have unit-size jobs associated and the label of an edge $e$ represents $(r_e, d_e)$. It is easy to verify that a schedule that preempts jobs only at integral time points, has maximum connectivity time of one. However, the following schedule with arbitrary preemption has connectivity time of two. We process $\{s^+, v_2\}$ in $[0, 0.5] \cup [1, 1.5]$, $\{s^+, v_3\}$ in $[0.5, 1] \cup [1.5, 2]$, $\{v_4, s^-\}$ in $[0, 0.5] \cup [1.5, 2]$, $\{v_5, s^-\}$ in $[0.5, 1.5]$, and the other edges are fixed by their time window. Moreover, one can verify that the constraint matrix of the corresponding LP for this instance is not totally unimodular. This instance shows that the integrality gap of the LP is at least two.

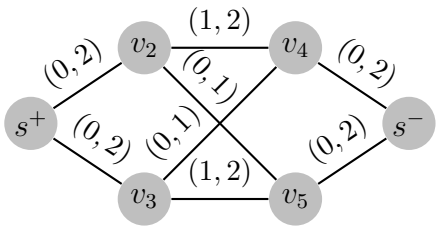


Figure 5.1: Example for the difference between arbitrary preemption and preemption only at integral time points.

## 5.4 Non-Preemptive Scheduling

In this section, we show that there is no $(c\sqrt[3]{|E|})$-approximation algorithm for non-preemptive MaxConnectivity for some $c > 0$ and that MinConnectivity is inapproximable, unless P = NP. Furthermore, we give an $(\ell + 1)$-approximation algorithm, where $\ell := |\{d_e - p_e \mid e \in E\}|$ is the number of distinct latest start times for jobs. We begin with the hardness results for MaxConnectivity. Before we can show no $(c\sqrt[3]{|E|})$-approximation algorithm exists, unless P = NP, we show a weaker result which provides us with a crucial gadget for the stronger result.

### 5.4.1 Complexity

**Theorem 5.5** Non-preemptive MaxConnectivity does not admit a $(2-\varepsilon)$-approximation algorithm for any $\varepsilon > 0$ and MinConnectivity is inapproximable, unless P = NP. This holds even for unit-size jobs.

***Proof.*** We show that the existence of a $(2-\varepsilon)$-approximation algorithm for non-preemptive MaxConnectivity allows to distinguish between Yes- and No-instances of 3SAT in polynomial time.

| 3SAT | |
|---|---|
| GIVEN: | Clauses $C_1, C_2, \ldots, C_m$ of exactly three variables in $X = \{x_1, x_2, \ldots, x_n\}$. |
| TASK: | Decide if there is a truth assignment to the variables in $X$ that satisfies all clauses. |

Given an instance of the strongly NP-complete 3SAT problem, we construct the following instance of non-preemptive MAXCONNECTIVITY. We pick two arbitrary but distinct time points $t_1 + 1 \leq t_2$ and a polynomially bounded time horizon $T$. We construct our instance such that connectivity is impossible outside $[t_1, t_1 + 1]$ and $[t_2, t_2 + 1]$. For this, $s^+$ is followed by a path $P$ from $s^+$ to a vertex $s'$ composed of three edges that disconnect $s^+$ from $s^-$ in the time intervals $[0, t_1]$, $[t_1+1, t_2]$, and $[t_2+1, T]$. These edges $e$ have $p_e = d_e - r_e$. Furthermore, we construct the network such that the total connectivity time is greater than one if and only if the 3SAT-instance is a YES-instance. And we show that if the total connectivity time is greater than one, then there is a schedule with maximum total connectivity time of two.

Let $Y(x_i)$ be the set of clauses containing the literal $x_i$ and $Z(x_i)$ be the set containing $\neg x_i$. Also set $k_i = 2|Y(x_i)|$ and $\ell_i = 2|Z(x_i)|$. We define the following node sets

- $V_1 := \{y_i^1, \ldots, y_i^{k_i} \mid i = 1, \ldots, n\}$,
- $V_2 := \{z_i^1, \ldots, z_i^{\ell_i} \mid i = 1, \ldots, n\}$,
- $V_3 := \{c_r \mid r = 1, \ldots, m + 1\}$,
- $V_4 := \{v_i \mid i = 1, \ldots, n + 1\}$

and set $V = \bigcup_{j=1}^{4} V_j \cup \{v : v \in P\} \cup \{s^-\}$. We introduce three edge types

- $\mathcal{E}_1 := \{e \in E : r_e = t_1, d_e = t_2 + 1, p_e = t_2 - t_1\}$,
- $\mathcal{E}_2 := \{e \in E : r_e = t_1, d_e = t_1 + 1, p_e = 1\}$, and
- $\mathcal{E}_3 := \{e \in E : r_e = t_2, d_e = t_2 + 1, p_e = 1\}$.

The graph $G = (V, E)$ consists of variable gadgets, shown in Figure 5.2, to which we connect the clause nodes $c_r$, $r = 1, \ldots, m + 1$. We define the following edge sets for the variable gadgets, namely,

- $E_1 := \{\{s', v_1\}, \{v_{n+1}, s^-\}\}$ of type $\mathcal{E}_2$,
- $E_2 := \{\{v_i, y_i^1\}, \{v_i, z_i^1\}, \{y_i^{k_i}, v_{i+1}\}, \{z_i^{\ell_i}, v_{i+1}\} : i = 1, \ldots, n\}$ of type $\mathcal{E}_2$,
- $E_3 := \{\{y_i^q, y_i^{q+1}\} : i = 1, \ldots, n; q = 1, 3, \ldots, k_i - 3, k_i - 1\}$ of type $\mathcal{E}_1$,
- $E_4 := \{\{z_i^q, z_i^{q+1}\} : i = 1, \ldots, n; q = 1, 3, \ldots, \ell_i - 3, \ell_i - 1\}$ of type $\mathcal{E}_1$,

- $E_5 := \{\{y_i^q, y_i^{q+1}\} : i = 1, \ldots, n; q = 2, 4, \ldots, k_i - 4, k_i - 2\}$ of type $\mathcal{E}_2$, and
- $E_6 := \{\{z_i^q, z_i^{q+1}\} : i = 1, \ldots, n; q = 2, 4, \ldots, \ell_i - 4, \ell_i - 2\}$ of type $\mathcal{E}_2$.

Note that a variable $x_i$ may only appear positive ($\ell_i = 0$) or only negative ($k_i = 0$) in our set of clauses. In this case, we also have an edge of type $\mathcal{E}_2$ connecting $v_i$ and $v_{i+1}$ besides the construction for the negative ($z$ nodes) or positive part ($y$ nodes). Finally, we add edges to connect the clause nodes to the graph. If some positive literal $x_i$ appears in clause $C_r$ and $C_r$ is the $q$-th clause with positive $x_i$, we add the edges $\{c_r, y_i^{2q-1}\}$ and $\{y_i^{2q}, c_{r+1}\}$ both of type $\mathcal{E}_3$. Conversely, if some $x_i$ appears negated in $C_r$ and $C_r$ is the $q$-th clause with $\neg x_i$, we add the edges $\{c_r, z_i^{2q-1}\}$ and $\{z_i^{2q}, c_{r+1}\}$ both of type $\mathcal{E}_3$. We also connect $c_1$ and $c_{m+1}$ to the graph by adding $\{s', c_1\}$ and $\{c_{m+1}, s^-\}$ of type $\mathcal{E}_3$. We define $E$ to be the union of all introduced edges. Observe that the network $G$ has $O(n + m)$ nodes and edges.

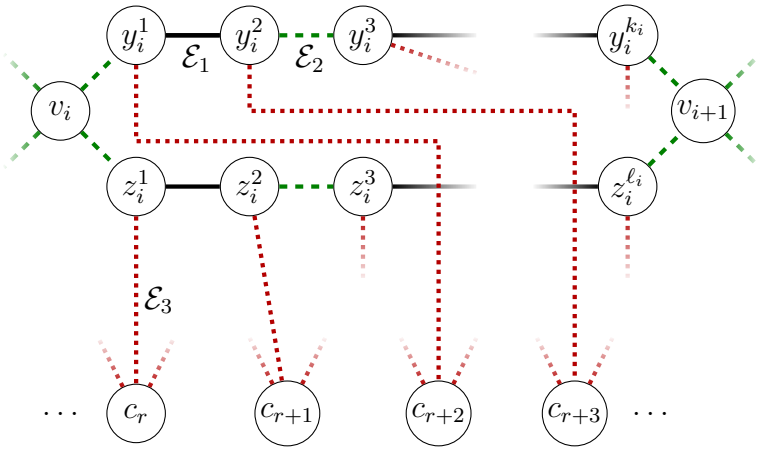


Figure 5.2: Schematic representation of the gadget for variable $x_i$, which appears negated in clause $C_r$ and positive in clause $C_{r+2}$ among others.

We call an $(s^+, s^-)$-path that contains no node from $V_3$ a *variable path* and an $(s^+, s^-)$-path with no node from $V_4$ a *clause path*. An $(s^+, s^-)$-path containing edges of type $\mathcal{E}_2$ and $\mathcal{E}_3$ does not connect $s^+$ with $s^-$ in $[t_1, t_1 + 1]$ or in $[t_2, t_2 + 1]$. Therefore, all paths other than variable paths and relevant clause paths are irrelevant for the connectivity of $s^+$ with $s^-$.

When maintaining all edges of type $\mathcal{E}_1$ in $[t_1, t_2]$, we have connectivity in $[t_2, t_2 + 1]$ exactly on all variable paths. Conversely, maintaining all edges of type $\mathcal{E}_1$ in $[t_1 + 1, t_2 + 1]$ yields connectivity in $[t_1, t_1 + 1]$ exactly on all relevant clause paths. On the other hand, any clause path can connect $s^+$ with $s^-$ only in $[t_1, t_1 + 1]$ and any variable path only in $[t_2, t_2 + 1]$. We now claim that there is a schedule with total connectivity time greater than one if and only if the 3SAT-instance is a YES-instance.

Let $S$ be a schedule with total connectivity time greater than one. Then there is a variable path $P^v$ with positive connectivity time in $[t_2, t_2 + 1]$ and a clause

path $P^c$ with positive connectivity time in $[t_1, t_1 + 1]$. As the total connectivity time is greater than one, $P^c$ cannot walk through both the positive part ($y$ nodes) and the negative part ($z$ nodes) of the gadget for any variable $x_i$. This allows to assume w.l.o.g. that $P^v$ and $P^c$ are disjoint between $s'$ and $s^-$. Say $P^v$ and $P^c$ share an edge on the negative part ($z$ nodes) of the gadget for variable $x_i$. Then we can redirect the variable path $P^v$ to the positive part ($y$ nodes) without decreasing the total connectivity time. The same works if they share an edge on the positive part.

Now set $x_i$ to FALSE if $P^v$ uses the nodes $y_i^1, \ldots, y_i^{k_i}$, that is the upper part of the variable gadget, and to TRUE otherwise. With this setting, whenever $P^c$ uses edges of a variable gadget, e.g. the sequence $c_r, z_i^{2q-1}, z_i^{2q}, c_{r+1}$ for some $r, q$, disjointness of $P^v$ and $P^c$ implies that clause $C_r$ is satisfied with the truth assignment of variable $x_i$. Since every node pair $c_r, c_{r+1}$ is only connected with paths passing through variable gadgets, and at least one of them belongs to $P^c$ we conclude that every clause $C_r$ is satisfied.

Consider a satisfying truth assignment. We define a schedule that admits a variable path $P^v$ with connectivity in $[t_2, t_2 + 1]$. This path $P^v$ uses the upper part ($y_i$-part) if $x_i$ is set to FALSE and the lower part ($z_i$-part) if $x_i$ is set to TRUE. That is, we maintain all edges of type $\mathcal{E}_1$ on the upper path ($y_i$-path) of the variable gadget for $x_i$ in $[t_1, t_2]$ if $x_i$ is FALSE and in $[t_1 + 1, t_2 + 1]$ if $x_i$ is TRUE. Conversely, edges of type $\mathcal{E}_1$ on the lower path ($z_i$-path) of the variable gadget for $x_i$ are maintained in $[t_1, t_2]$ if $x_i$ is TRUE and in $[t_1 + 1, t_2 + 1]$ if $x_i$ is FALSE. This implies for the part of the gadget for $x_i$ that is not used by $P^v$ that the corresponding edges of type $\mathcal{E}_1$ are scheduled to allow connectivity during $[t_1, t_1 + 1]$. These edges can be used in a clause path to connect node $c_r$ with $c_{r+1}$ for some clauses $C_r$ that is satisfied by the truth assignment of $x_i$. Since all clauses are satisfied by some variable $x_i$ there exists a clause path $P^c$ admitting connectivity in $[t_1, t_1+1]$. Therefore, the constructed schedule allows connectivity during both intervals $[t_1, t_1 + 1]$ and $[t_2, t_2 + 1]$.

For $t_1 = 0$, $t_2 = 1$, and $T = 2$, this construction uses only unit-size jobs, and in the MINCONNECTIVITY case YES-instances have an objective value of 0 and NO-instances a value of 1. □

We reuse the construction in the proof of Theorem 5.5 repeatedly to obtain the following improved lower bound.

**Theorem 5.6** There is a constant $c > 0$ such that there does not exist a $(c\sqrt[3]{|E|})$-approximation algorithm for non-preemptive MAXCONNECTIVITY, unless P = NP.

***Proof.*** We reuse the construction in the proof of Theorem 5.5 to construct a network that has maximum connectivity time $n$ if the given 3SAT instance is a YES-instance and maximum connectivity time 1 otherwise. This implies that there cannot be an $(n - \varepsilon)$-approximation algorithm for non-preemptive MAXCONNECTIVITY, unless P = NP. Here, $n$ is again the number of variables in the

given 3SAT instance. Note that the construction in the proof of Theorem 5.5 has $\Theta(n)$ maintenance jobs and thus there exists a constant $c_1 > 0$ such that $|E| \leq c_1 \cdot n$. In this proof, we will introduce $\Theta(n^2)$ copies of the construction and thus $|E| \leq c_2 \cdot n^3$ for some $c_2 > 0$, which gives that $n \geq c_3 \sqrt[3]{|E|}$ for some $c_3 > 0$. This gives the statement.

For the construction, we use $n^2 - n$ copies of the 3SAT-network from the proof of Theorem 5.5, where each one uses different $(t_1, t_2)$-combinations with $t_1, t_2 \in \{0, \ldots, n-1\}$ and $t_1 \neq t_2$. We use these copies as 3SAT-gates and mutually connect them as depicted in Figure 5.3. Recall that for one such 3SAT-network we have the freedom of choosing the intervals $[t_1, t_1 + 1]$ and $[t_2, t_2 + 1]$, which are relevant for connectivity. This choice now differs for every 3SAT-gate.

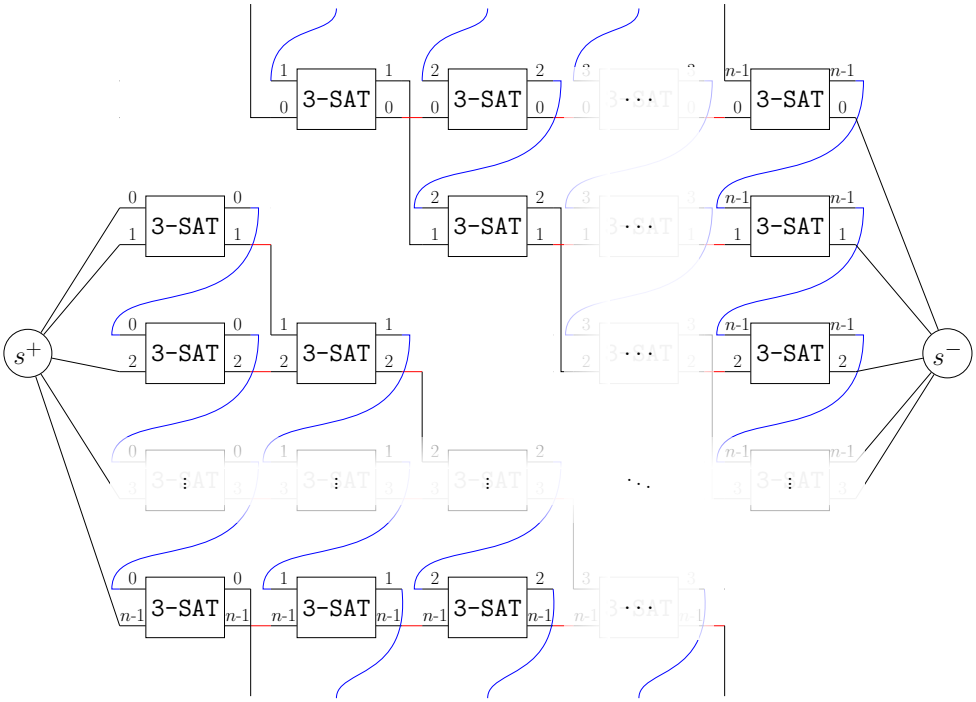


Figure 5.3: Schematic representation of the network of 3SAT-gates.

Think of the construction as an $(n \times n)$-matrix $M$ with an empty diagonal. Entry $(i, j)$, $i, j \in \{0, \ldots, n-1\}$, in $M$ corresponds to a 3SAT-gate in that variable paths only exist in time slot $[i, i+1]$ and relevant clause paths exist only in $[j, j+1]$. This is enforced by the edges of type $\mathcal{E}_2$, which prevent variable paths in $[j, j+1]$, and edges of type $\mathcal{E}_3$, which prevent relevant clause paths in $[i, i+1]$. Edges between the $s^+$-copy and $s'$-copy of the 3SAT-gate$(i, j)$ prevent connectivity outside of $[i, i+1]$ and $[j, j+1]$. Note that now

- $\mathcal{E}_1 := \{e \in E : r_e = i, d_e = j + 1, p_e = j - i\}$ if $i < j$, and
- $\mathcal{E}_1 := \{e \in E : r_e = j, d_e = i + 1, p_e = i - j\}$ if $i > j$.

The $s^+$-copy of the 3SAT-gate$(i,j)$ is connected to two paths, where one of them allows connectivity only during $[i,i+1]$ and the other one only during $[j,j+1]$. The same is done for the $s^-$-copy of the 3SAT-gate$(i,j)$. In Figure 5.3, this is illustrated by labels on the paths. A label $i \in \{0,\dots,n-1\}$ means, that this path allows connectivity only during $[i,i+1]$. The upper path connected to a 3SAT-gate specifies the time slot, where variable paths may exist, and the lower path specifies the time slot, where relevant clause paths may exist. When following the path with label $k \in \{0,\dots,n-1\}$, we pass the gadgets in column $j = 0,\dots,k-1$ on the lower path having $j$ on the upper path. In column $k$, we walk through all gadgets on the upper path and then we proceed with column $j = k+1,\dots,n-1$ on the lower path having $j$ again on the upper path. Eventually, we connect the 3SAT-gate$(n-1,k)$ to the vertex $s^-$.

Note that within 3SAT-gate$(i,j)$ we have connectivity during $[i,i+1]$ *and* $[j,j+1]$ if and only if the corresponding 3SAT-instance is a YES-instance. Also notice that we can assume due to [11] that all jobs start at integral times, which allows us to ignore schedules with fractional job starting times and therefore fractional connectivity within a time interval $[i,i+1]$. Now, if the 3SAT-instance is a YES-instance, there is a global schedule such that its restriction to every gate 3SAT-gate$(i,j)$ allows connectivity during both intervals. Thus for each label $k \in \{0,\dots,n-1\}$ there exists a path with this label that has connectivity during $[k,k+1]$. This implies that the maximum connectivity time is $n$.

Conversely, suppose there exists a global schedule with connectivity during $[i,i+1]$ and $[j,j+1]$ for some $i \neq j$. Then there must exist two paths $P_1, P_2$ from $s^+$ to $s^-$ with two distinct labels $i$ and $j$, each realizing connectivity during one of both intervals. By construction they walk through the 3SAT-gate$(i,j)$. This implies by the proof of Theorem 5.5, that the global schedule restricted to this gate corresponds to a satisfying truth assignment for the 3SAT-instance. That is, the 3SAT-instance is a YES-instance. With the previous observation, it follows that an optimal schedule has maximum connectivity time of $n$. □

The results above were for general graphs, but even for graphs as simple as parallel paths, the problem does not become easy, as the following theorem shows.

**Theorem 5.7** Non-preemptive CONNECTIVITY is strongly NP-hard even on parallel paths, and non-preemptive MINCONNECTIVITY is inapproximable even on parallel paths.

***Proof.*** We reduce from the strongly NP-complete 3SAT problem consisting of $m$ clauses $C_1, C_2, \dots, C_m$ each of exactly three variables in $X = \{x_1, x_2, \dots, x_n\}$. We construct a network with $2n$ paths from $s^+$ to $s^-$, two for each variable of the 3SAT instance. Let $P_i$ and $\bar{P}_i$ denote the two paths for variable $x_i$. We will introduce several maintenance jobs for each path, understanding that each new job is associated with a different edge of the path. Since the ordering of these edges does not matter, we will directly associate each job with a path without explicitly

specifying the respective edge of the job. The network will allow a schedule that maintains connectivity if and only if the 3SAT instance is satisfiable.

For convenience, assume that $n \geq m$, otherwise we introduce additional dummy variables. We define a time horizon $T = 8n$ that we subdivide into five intervals $A = [0, 2n), B = [2n, 3n), C = [3n, 5n), D = [5n, 6n), E = [6n, 8n]$. For each variable $x_i$, we define a job each on paths $P_i$ and $\bar{P}_i$ with the time window $[0, T]$ and processing time $3n$. We will ensure that neither job is scheduled to cover the time interval $C$ entirely in any feasible schedule for the Connectivity problem. This implies that a variable job either covers $B$ or $D$ without intersecting the other. The job on $P_i$ (resp. $\bar{P}_i$) covering $B$ will correspond to the literal $x_i$ (resp. $\bar{x}_i$) being set to True. We will of course ensure that not both literals can be set to True simultaneously, but we will allow both to be False, which simply means that the truth assignment remains satisfying, no matter how the variable is set.

In the following, we introduce blocking jobs that all have a time window of unit length and unit processing time. In this way, introducing a blocking job at time $t$ simply renders the corresponding path unusable during the time interval $[t, t+1)$. To ensure that the variable jobs for variable $x_i$ do not cover $C$ completely, we add a blocking job at time $t_i = 3n + 2(i-1)$ to all paths except $P_i$ and a blocking job at time $t'_i = 3n+2(i-1)+1$ to all paths except $\bar{P}_i$. The first job forces the variable job for the literal $x_i$ not to cover $C$ completely, since otherwise connectedness is interrupted during the time interval $[t_i, t'_i)$. The second blocking job accomplishes the same for the literal $\bar{x}_i$. Note that the blocking jobs for each literal occupy a unique part of the time window $C$.

In order to force at most one literal of each variable $x_i$ to be set to True, we introduce a blocking job at time $t''_i = 2n + (i-1)$ on all paths except $P_i$ and $\bar{P}_i$. These blocking jobs ensure that either path $P_i$ or $\bar{P}_i$ must be free during time $[t''_i, t''_i + 1)$, which means not both variable jobs may be scheduled to cover $B$ (recall each variable job either covers $B$ or $D$ without intersecting the other). Again, the blocking jobs for each variable occupy a unique part of the time window $B$.

For each clause $C_j$ we introduce a blocking job at time $5n + j$ on each path except the three paths that correspond to literals in $C_j$. These blocking jobs force that at least one of the literals of the clause has to be set to True, i.e., be scheduled to overlap $B$ instead of $D$, otherwise connectivity is interrupted during time $[5n + j, 5n + j + 1)$. Note again that the blocking jobs for each clause occupy a unique part of the time window $D$.

It is now easy to verify that each satisfying truth assignment leads to a feasible schedule without disconnectedness for the Connectivity problem and vice versa. □

### 5.4.2 An Approximation Algorithm

We now give an algorithm that computes an $(\ell + 1)$-approximation for non-preemptive MaxConnectivity, where $\ell \leq |E|$ is the number of different time points $d_e - p_e, e \in E$. The basic idea is that we consider a set of $\ell+1$ feasible maintenance schedules, whose total time of connectivity upper bounds the maximum total connectivity time of a single schedule. Then the schedule with maximum connectivity time among our set of $\ell + 1$ schedules is an $(\ell + 1)$-approximation.

The schedules we consider start every job either immediately at its release date, or at the latest possible time. In the latter case it finishes exactly at the deadline. More precisely, for a fixed time point $t$, we start the maintenance of all edges $e \in E$ with $d_e - p_e \geq t$ at their latest possible start time $d_e - p_e$. All other edges start maintenance at their release date $r_e$. This yields at most $\ell+1 \leq |E|+1$ different schedules $S_t$, as for increasing $t$, each time point where $d_e - p_e$ is passed for some edge $e$ defines a new schedule. Algorithm 1 formally describes this procedure, where $E(t) := \{e \in E : e \text{ is not maintained at } t\}$.

**Input:** MaxConnectivity instance.
**Output:** $(\ell + 1)$-approximate schedule $S$.
1: Let $t_1 < \cdots < t_\ell$ be all different time points $d_e - p_e, e \in E$.
2: Let $t_0 = 0$ and $t_{\ell+1} = T$.
3: Let $S_i$ be the schedule, where all edges $e$ with $d_e - p_e < t_i$ start maintenance at $r_e$ and all other edges at $d_e - p_e$, $i = 1, \ldots, \ell + 1$.
4: For each $S_i$, initialize total connectivity time $c(t_i) \leftarrow 0, i = 1, \ldots, \ell + 1$.
5: **for** $i = 1$ to $\ell + 1$ **do**
6: Partition the interval $[t_{i-1}, t_i]$ into subintervals such that each time point $r_e, r_e + p_e, d_e$, $e \in E$, in this interval defines a subinterval bound.
7: **for all** subintervals $[a, b] \subseteq [t_{i-1}, t_i]$ **do**
8: **if** $(V, E(1/2 \cdot (a + b)))$ contains an $(s^+, s^-)$-path for $S_i$ **then**
9: Increase $c(t_i)$ by $b - a$.
10: **return** Schedule $S_i$ for which $c(t_i), i = 1, \ldots, \ell + 1$, is maximized.

**Algorithm 1:** Approximation Algorithm for Non-preemptive MaxConnectivity

Algorithm 1 considers finitely many intervals in total, as all (sub-)interval bounds are defined by a time point $r_e, r_e + p_e, d_e - p_e$ or $d_e$ of some $e \in E$. As we can check the network for $(s^+, s^-)$-connectivity in polynomial time, and the algorithm does this for each (sub-)interval, Algorithm 1 runs in polynomial time. For the proof of the approximation factor, the following lemma is crucial.

**Lemma 5.8** Consider a given MaxConnectivity instance and a time interval $[a, b]$ that does not contain any of the time points $r_e, r_e + p_e, d_e - p_e, d_e$ for any $e \in E$ in its interior. Consider the schedule $S$, where every edge $e \in E$ with $d_e - p_e < b$ starts maintenance at its release date $r_e$ and all other

edges at $d_e - p_e$. If there is a schedule that allows an $(s^+, s^-)$-path at a time point $t \in (a, b)$, then we have connectivity for the whole time interval $[a, b]$ in the schedule $S$.

***Proof.*** Observe first, as no time point $r_e, r_e + p_e, d_e - p_e$ or $d_e$ of an edge $e \in E$ lies in the interior of $[a, b]$, any edge maintained within $[a, b]$ in the schedule $S$ is actually maintained for the complete interval. Now consider an edge $e$ that is maintained within $[a, b]$ in $S$. The maintenance for $e$ cannot start at time $d_e - p_e$, as we choose this start time only if $d_e - p_e \geq b$. From $d_e - p_e < b$ it follows that $d_e - p_e \leq a$. This, however, means that the job for $e$ starts in any feasible schedule before or at time point $a$. The job is scheduled in $S$ at time $r_e$ and finishes at time $r_e + p_e$. As edge $e$ is maintained during the complete time interval $[a, b]$, we have $b \leq r_e + p_e$. This means the job for $e$ finishes in any feasible schedule after time point $b$ and thus edge $e$ is maintained in any feasible schedule during the complete interval $[a, b]$. Therefore, if there is no connectivity in $[a, b]$ for $S$, then there is also no connectivity during this interval for any other feasible schedule. □

Using this lemma, we can prove the approximation guarantee for our algorithm.

**Theorem 5.9** When $\ell \leq |E|$ is the number of different time points $d_e - p_e, e \in E$, Algorithm 1 is an $(\ell+1)$-approximation algorithm for non-preemptive MaxConnectivity on general graphs.

***Proof.*** Observe first, that all schedules $S_i, i = 1, \ldots, \ell+1$, as we defined them, are feasible. Let $c(t_i)$ be the connectivity time of schedule $S_i$. Then, by Lemma 5.8, the optimal schedule has total connectivity time no greater than $\sum_{i=1}^{\ell+1} c(t_i)$. This is because at any time point $t$, where the network is connected in the optimal schedule, the schedule $S_i$ where $t \in [t_{i-1}, t_i]$ is also connected. The algorithm returns a schedule $S_i$ that maximizes $c(t_i)$, which completes the proof. □

## 5.5 Power of Preemption

In this section, we investigate the power of preemption the for Connectivity. We first focus on MinConnectivity on a path and analyze how much we can gain by allowing preemption.

First, we show that there is an algorithm that computes a non-preemptive schedule whose value is bounded by $O(\log |E|)$ times the value of an optimal preemptive schedule. Second, we argue that one cannot gain more than a factor of $\Omega(\log |E|)$ by allowing preemption.

**Theorem 5.10** The power of preemption is $\Theta(\log |E|)$ for MinConnectivity on a path.

Observe that if at least one edge of a path is maintained at time $t$, then the whole path is disconnected at $t$. We first give an algorithm for MinConnectivity on a path that constructs a non-preemptive schedule with cost at most $O(\log |E|)$ times the cost of an optimal preemptive schedule.

We first compute an optimal preemptive schedule. This can be done in polynomial time by Theorem 5.1. Let $x_t$ be a variable that is 1 if there exists a job $j$ that is processed at time $t$ and 0 otherwise. We shall refer to $x$ also as the *maintenance profile*. Furthermore, let $a := \int_0^T x_t \, \mathrm{d}t$ be the active time, i.e., the total time of maintenance. Then we apply the following *splitting procedure*. We compute the time point $\bar{t}$ where half of the maintenance is done, i.e., $\int_0^{\bar{t}} x_t \, \mathrm{d}t = a/2$. Let $E(t) := \{e \in E \mid r_e \leq t \wedge d_e \geq t\}$ and $p_{\max} := \max_{e \in E(t)} p_e$. We reserve the interval $[\bar{t} - p_{\max}, \bar{t} + p_{\max}]$ for the maintenance of the jobs in $E(\bar{t})$, although we might not need the whole interval. We schedule each job in $E(\bar{t})$ around $\bar{t}$ so that the processing time before and after $\bar{t}$ is the same. If the release date (deadline) of a jobs does not allow this, then we start (complete) the job at its release date (deadline). Then we mark the jobs in $E(\bar{t})$ as scheduled and delete them from the preemptive schedule.

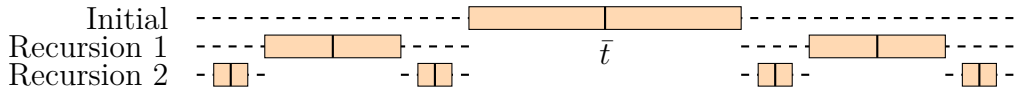


Figure 5.4: A sketch of the splitting procedure and the reserved intervals.

This splitting procedure splits the whole problem into two separate instances $E_1 := \{e \in E \mid d_e < \bar{t}\}$ and $E_2 := \{e \in E \mid r_e > \bar{t}\}$. Note that in each of these sub-instances the total active time in the preemptive schedule is at most $a/2$. We apply the splitting procedure to both sub-instances and follow the recursive structure of the splitting procedure until all jobs are scheduled.

**Lemma 5.11** For MinConnectivity on a path, the given algorithm constructs a non-preemptive schedule with cost $O(\log |E|)$ times the cost of an optimal preemptive schedule.

***Proof.*** The progression of the algorithm can be described by a binary tree in which a node corresponds to a partial schedule generated by the splitting procedure for a subset of the job and edge set $E$. The root node corresponds to the partial schedule for $E(\bar{t})$ and the (possibly) two children of the root correspond to the partial schedules generated by the splitting procedure for the two subproblems with initial job sets $E_1$ and $E_2$. We can cut a branch if the initial set of jobs is empty in the corresponding subproblem. We associate with every node $v$ of this tree $B$ two values $(s_v, a_v)$ where $s_v$ is the number of scheduled jobs in the subproblem corresponding to $v$ and $a_v$ is the amount of maintenance time spent for the scheduled jobs.

The binary tree $B$ has the following properties. First, $s_v \geq 1$ holds for all $v \in B$, because the preemptive schedule processes some job at the midpoint $\bar{t}_v$ which means that there must be a job $e \in E$ with $r_e \leq \bar{t}_v \wedge d_e \geq \bar{t}_v$. This observation implies that the tree $B$ can have at most $|E|$ nodes and since we want to bound the worst total cost we can assume w.l.o.g. that $B$ has exactly $|E|$ nodes. Second, $\sum_{v \in B} a_v = \int_0^T y_t \, \mathrm{d}t$ where $y_t$ is the maintenance profile of the non-preemptive solution.

The cost $a_v$ of the root node (level-0 node) is bounded by $2p_{\max} \leq 2a$. The cost of each level-1 node is bounded by $2 \cdot a/2 = a$, so the total cost on level 1 is also at most $2a$. It is easy to verify that this is invariant, i.e., the total cost at level $i$ is at most $2a$ for all $i \geq 0$, since the worst node cost $a_v$ halves from level $i$ to level $i+1$, but the number of nodes doubles in the worst case. We obtain the worst total cost when $B$ is a complete balanced binary tree. This tree has at most $O(\log |E|)$ levels and therefore the worst total cost is $a \cdot O(\log |E|)$. The total cost of the preemptive schedule is $a$. □

We now provide a matching lower bound for the power of preemption for MinConnectivity on a path.

**Lemma 5.12** The power of non-preemption is $\Omega(\log |E|)$ for MinConnectivity on a path.

***Proof.*** We construct a path with $|E|$ edges and divide the $|E|$ jobs into $\ell$ levels such that level $i$ contains exactly $i$ jobs for $1 \leq i \leq \ell$. Hence, we have $|E| = \ell(\ell+1)/2$ jobs. Let $P$ be a sufficiently large integer such that all of the following numbers are integers. Let the $j$th job of level $i$ have release date $(j-1)P/i$, deadline $(j/i)P$, and processing time $P/i$, where $1 \leq j \leq i$. Note that now no job has flexibility within its time window, and thus the value of the resulting schedule is $P$.

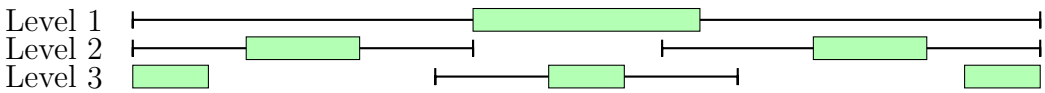


Figure 5.5: A rough sketch of the instance for 3 levels.

We now modify the instance as follows. At every time point $t$ where at least one job has a release date and another job has a deadline, we stretch the time horizon by inserting a gap of size P. This stretching at time $t$ can be done by adding a value of $P$ to all time points after the time point $t$, and also adding a value of $P$ to all release dates at time $t$. The deadlines up to time $t$ remain the same. Observe that the value of the optimal preemptive schedule is still $P$, because when introducing the gaps we can move the initial schedule accordingly such that we do not maintain any job within the gaps of size $P$. Figure 5.5 shows a rough sketch of this construction.

We now consider the optimal non-preemptive schedule. The cost of scheduling the only job at level 1 is $P$. In parallel to this job we can schedule at most one job from each other level, without having additional cost. This is guaranteed by the introduced gaps. At level 2 we can fix the remaining job with additional cost $P/2$. As before, in parallel to this fixed job, we can schedule at most one job from each level $i$ where $3 \leq i \leq \ell$. Applying the same argument to the next levels, we notice that for each level $i$ we introduce an additional cost of value $P/i$. Thus the total cost is at least $\sum_{i=1}^{\ell} P/i \in \Omega(P \log \ell)$ with $\ell \in \Theta(\sqrt{|E|})$. □

For MaxConnectivity, preemption is even more crucial, as the power of preemption can be unbounded, as the next example shows.

**Theorem 5.13** For non-preemptive MaxConnectivity on a path the power of preemption is unbounded.

***Proof.*** Consider a path of four consecutive edges $e_1 = \{s^+, u\}, e_2 = \{u, w\}$, $e_3 = \{w, v\}, e_4 = \{v, s^-\}$, each associated with a maintenance job as depicted in Figure 5.6. That is, $r_1 = r_2 = 0$, $d_1 = r_3 = p_1 = p_4 = 1$, $p_2 = p_3 = 2$, $r_4 = d_2 = 3, d_3 = d_4 = 4$.

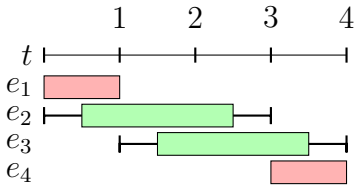


Figure 5.6: Example for an unbounded power of preemption.

There is no non-preemptive schedule that allows connectivity at any point in time, as the maintenance job of edge $e_i$ blocks edge $e_i$ in time slot $[i-1, i]$. On the other hand, when allowing preemptive schedules, we can process the job of edge $e_2$ in $[0, 2]$ and the job of edge $e_3$ in $[1, 2]$ and $[3, 4]$. Then no maintenance job is scheduled in the time interval $[2, 3]$ and therefore we have connectivity for one unit of time. □

## 5.6 Mixed Scheduling

We know that both the non-preemptive and preemptive Connectivity on a path are solvable in polynomial time by Theorem 5.1 and [54], respectively. Interestingly, the complexity changes when mixing the two job types – even on a simple path.

**Theorem 5.14** Connectivity with preemptive and non-preemptive maintenance jobs is NP-hard, even on a path.

***Proof.*** We reduce the NP-hard PARTITION problem to our problem. See the proof of Theorem 4.1 for a definition of the PARTITION problem.

Given an instance of PARTITION, we create an instance for CONNECTIVITY on a path consisting of $3n+2$ edges between $s^+$ and $s^-$ with preemptive and non-preemptive maintenance jobs. We create three types of job sets denoted as $J_1, J_2$ and $J_3$, where the first two job sets model the binary decision involved in choosing a subset of numbers to form a partition, whereas the third job set performs the summation over the numbers picked for a partition. The construction is visualized in Figure 5.7.

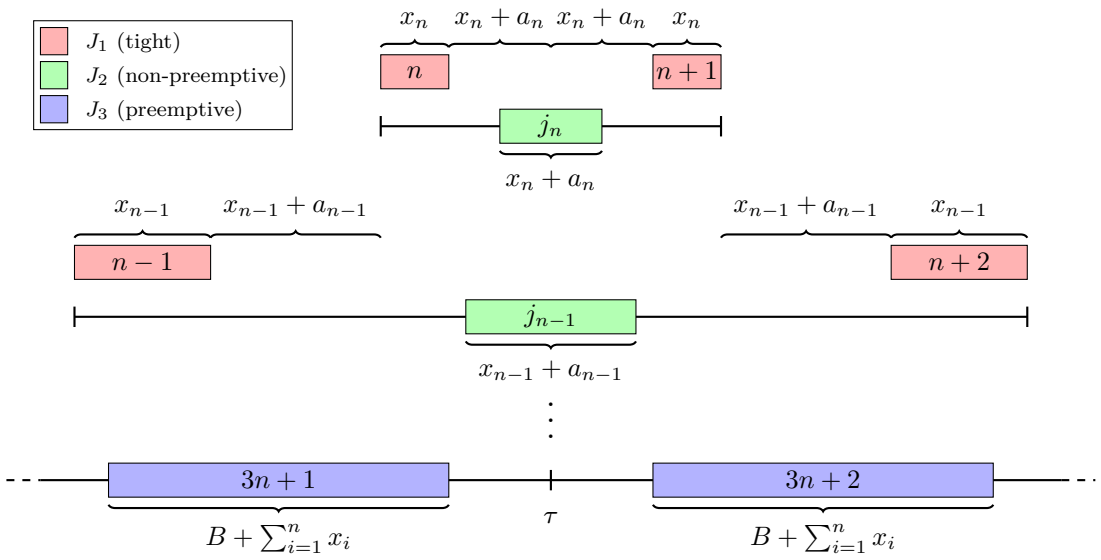


Figure 5.7: Schematic representation of the constructed instance for CONNECTIVITY.

The job set $J_1 := \{1, 2, \ldots, 2n-1, 2n\}$ contains $2n$ *tight* jobs, i.e., $r_j + p_j = d_j$ for all $j \in J_1$. For every element $a_i$ we have two tight jobs $i$ and $2n-(i-1)$ both having processing time $4^{n-i}B =: x_i$. The release date of a job $j \in \{2, \ldots, n\} \subset J_1$ is $r_j = \sum_{k=1}^{j-1} 2x_k + a_k$ and $r_1 = 0$. Let $\tau := \sum_{k=1}^{n} 2x_k + a_k$. For $j \in \{n+1, \ldots, 2n\} \subset J_1$ we have $d_j = \tau + \sum_{k=n+1}^{j} 2x_{2n-k+1} + a_{2n-k+1}$. Note that the tight jobs in $J_1$ are constructed in such a way that everything is symmetric with respect to the time point $\tau$.

The job set $J_2 := \{2n+1, \ldots, 3n\}$ contains $n$ non-preemptive jobs. Let $j_i := 2n + i$. For every element $a_i$ we introduce job $j_i$ with processing time $p_{j_i} = x_i + a_i$, release date $r_{j_i} = r_i$, and deadline $d_{j_i} = d_{2n-(i-1)}$. Again, everything is symmetric with respect to time point $\tau$.

Finally, the set $J_3 := \{3n+1, 3n+2\}$ contains two preemptive jobs, where each of them has processing time $W := B + \sum_{i=1}^{n} x_i$. Furthermore, we have $r_{3n+1} = 0$, $d_{3n+1} = \tau$, $r_{3n+2} = \tau$, $d_{3n+2} = 2\tau$.

We now show that there is a feasible schedule for the constructed instance

that disconnects the path for at most $2W$ time units if and only if the given Partition instance is a Yes-instance.

Suppose there is a subset $A \subset \{1, \ldots, n\}$ with $\sum_{i \in A} a_i = B$. For each $i \in A$, we start the corresponding job $j_i \in J_2$ at its release date and the remaining jobs in $J_2$ corresponding to the elements $a_i$ with $i \in \{1, \ldots, n\} \setminus A$ are scheduled such that they complete at their deadline. This creates $B + \sum_{i=1}^{n} x_i$ time slots in both intervals $[0, \tau]$ and $[\tau, 2\tau]$ with no connection between $s^+$ and $s^-$. The jobs $3n+1$ and $3n+2$ can be preempted in $[0, \tau]$ and $[\tau, 2\tau]$, respectively, and thus if we align their processing with the chosen maintenance slots, we get a schedule that disconnects $s^+$ and $s^-$ for $2W = 2(B + \sum_{i=1}^{n} x_i)$ time units.

Conversely, suppose that there is a feasible schedule for the constructed instance that disconnects the path for at most $2W$ time units. By induction on $i$, we show that every job $j_i = 2n + i$ either starts at its release date or it completes at its deadline in such a schedule.

Consider the base case of $i = 1$. We first observe that w.l.o.g. job $j_1$ either starts at its release date or completes at its deadline or is scheduled somewhere in $[x_1, 2\tau - x_1]$. Suppose it starts somewhere in $(0, x_1)$ or completes somewhere in $(\tau - x_1, \tau)$. Then we do not increase the total time where the path is disconnected if we push job $j_1$ completely to the left or completely to the right. If we schedule job $j_1$ in $[x_1, 2\tau - x_1]$, then the total time where the path is disconnected is at least $3x_1 + a_1 > 2x_1 + x_1$. We will now show that $x_1 \geq 2(B + \sum_{k=2}^{n} x_k)$ for $n \geq 2$, which shows that the path is then disconnected for more than $2W$ time units, and thus job $j_1$ cannot be processed in $[x_1, 2\tau - x_1]$. The inequality is true for $n \geq 2$, since

$$\begin{aligned} 2B + 2\sum_{k=2}^{n} x_k &= 2B(1 + \sum_{k=2}^{n} 4^{n-k}) \\ &= 2B(1 + \sum_{k=0}^{n-2} 4^{k}) \\ &= 2B(1 + 1/3(4^{n-1} - 1)) \\ &\leq 4^{n-1}B = x_1. \end{aligned}$$

This finishes the proof for $i = 1$.

Suppose, the statement is true for $i = 1, \ldots, \ell - 1$ with $\ell \in \{2, \ldots, n-1\}$. As in the base case, we can show that job $j_\ell$ either starts at its release date or completes at its deadline or is scheduled somewhere in $[r_{j_\ell} + x_\ell, d_{j_\ell} - x_\ell]$. If job $j_\ell$ is processed in $[r_{j_\ell} + x_\ell, d_{j_\ell} - x_\ell]$, then the total time where the path is disconnected is at least

$$\sum_{k=1}^{\ell-1}(2x_k + a_k) + 3x_\ell + a_\ell > \sum_{k=1}^{\ell} 2x_k + x_\ell.$$

Again, we will show that $x_\ell \geq 2(B + \sum_{k=\ell+1}^{n} x_k)$ for $\ell \in \{2, \ldots, n-1\}$, which shows that the path is then disconnected for more than $2W$ time units, and

thus job $j_\ell$ cannot be processed in $[r_{j_\ell} + x_\ell, d_{j_\ell} - x_\ell]$. The inequality is true for $\ell \in \{2, \ldots, n-1\}$, since

$$\begin{aligned} 2B + 2\sum_{k=\ell+1}^{n} x_k &= 2B(1 + \sum_{k=\ell+1}^{n} 4^{n-k}) \\ &= 2B(1 + \sum_{k=0}^{n-\ell-1} 4^{k}) \\ &= 2B(1 + 1/3(4^{n-\ell} - 1)) \\ &\leq 4^{n-\ell}B = x_\ell. \end{aligned}$$

For $i = n$, we again use the fact that $j_n$ either starts at its release date or completes at its deadline or is scheduled somewhere in $[r_{j_n} + x_n, d_{j_n} - x_n]$. If the latter case is true, then the total time where the path is disconnected is at least

$$\begin{aligned} \sum_{k=1}^{n-1}(2x_k + a_k) + 3x_n + a_n &= \sum_{k=1}^{n}(2x_k + a_k) + x_n \\ &> 2(B + \sum_{k=1}^{n} x_k) = 2W. \end{aligned}$$

There is a feasible schedule for the constructed instance that disconnects the path for at most $2(B + \sum_{k=1}^{n} x_k)$ time units. This means that in both $[0, \tau]$ and $[\tau, 2\tau]$ the path is disconnected for exactly $B + \sum_{k=1}^{n} x_k$ time units. Consider the set $A := \{i : j_i \text{ starts at its release date}\}$. We conclude that

$$\sum_{k=1}^{n} x_k + \sum_{k \in A} a_k = \sum_{k=1}^{n} x_k + \sum_{k \notin A} a_k = \sum_{k=1}^{n} x_k + B.$$

□

We complement this hardness result with a simple 2-approximation algorithm. For MinConnectivity it is easy to see that running the optimal preemptive and the optimal non-preemptive algorithms on the preemptive and non-preemptive job sets individually gives a 2-approximation.

**Theorem 5.15** There is a 2-approximation algorithm for MinConnectivity on a path with preemptive and non-preemptive maintenance jobs.

***Proof.*** Consider an optimal schedule $S^*$ for the mixed instance and let $|S^*|$ be the total time of disconnectivity in $S^*$. Furthermore, let $S^*_{np}$ (resp. $S^*_p$) be the restriction of $S^*$ to only non-preemptive (resp. preemptive) jobs. Note that the schedule $S^*_{np}$ (resp. $S^*_p$) is feasible for the corresponding non-preemptive (resp. preemptive) instance. We separate the preemptive from the non-preemptive jobs and obtain two separate instances. Solving them individually in polynomial time and combining the resulting two solutions $S_{np}$ and $S_p$ to a schedule $S$ gives the claimed result, because $|S| \leq |S_{np}| + |S_p| \leq |S^*_{np}| + |S^*_p| \leq 2|S^*|$. □

## 5.7 Conclusion and Open Problems

The concept of scheduling maintenance work and thus combining network flows with scheduling aspects is a very recent field of research. While there are solutions using-IP based methods and heuristics, exact and approximation algorithms have not been considered extensively. We provide a theoretical understanding of Connectivity, which is inherent to all forms of maintenance scheduling. This yields the basis for analyzing more involved settings for maintenance scheduling, such as maximizing the network throughput or limiting the number of concurrently performed maintenance jobs.

In particular, maximizing the network throughput on a path is already covered by our results and all our hardness results carry over to maximizing the network throughput between two designated nodes $s^+$ and $s^-$. On general graphs, the absence of $c\sqrt{|E|}$-approximation algorithms indicates that heuristics and IP-based methods [10–12] might be a good way of approaching this problem. However, an interesting open question is whether the inapproximability results carry over to restricted families of graphs such as series-parallel graphs, as the network motivating the works by [10–12] is indeed series-parallel.

Our results on the power of preemption as well as the efficient algorithm for preemptive instances show that allowing preemption is very desirable. Thus, it could be interesting to study models where preemption is allowed, but comes at a cost to make it more realistic.

On a path, our results have implications for minimizing busy time, as we want to minimize the number of times where some edge on the path is maintained. Here, an interesting open question is to improve on our 2-approximation for the mixed case, or to show an inapproximability result for it.

# Bibliography


[1] Amazon EC2 Pricing Options: `https://aws.amazon.com/ec2/pricing/`.

[2] F. Abed, L. Chen, Y. Disser, M. Groß, N. Megow, J. Meißner, A. Richter, and R. Rischke. Scheduling maintenance jobs in networks. Submitted.

[3] T. Al-Khamis and R. M'Hallah. A two-stage stochastic programming model for the parallel machine scheduling problem with machine capacity. *Comput. Oper. Res.*, 38(12):1747–1759, 2011.

[4] S. Albers. Energy-efficient algorithms. *Commun. ACM*, 53(5):86–96, 2010.

[5] K. Baker. *Introduction to Sequencing and Scheduling.* John Wiley & Sons, New York, 1974.

[6] K. R. Baker, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Preemptive scheduling of a single machine to minimize maximum cost subject to release dates and precedence constraints. *Oper. Res.*, 31(2):381–386, 1983.

[7] N. Bansal and K. Pruhs. The geometry of scheduling. *SIAM J. Comput.*, 43(5):1684–1698, 2014.

[8] A. Baveja, A. Chavan, A. Nikiforov, A. Srinivasan, and P. Xu. Improved bounds in stochastic matching and optimization. In *Proceedings of the 18th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX) and 19th International Workshop on Randomization and Computation (RANDOM)*, volume 40 of *LIPIcs*, pages 124–134. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[9] A. Bley, D. Karch, and F. D'Andreagiovanni. WDM fiber replacement scheduling. *Electronic Notes in Discrete Mathematics*, 41:189–196, 2013.

[10] N. Boland, T. Kalinowski, and S. Kaur. Scheduling arc shut downs in a network to maximize flow over time with a bounded number of jobs per time period. *J. Comb. Optim.*, pages 1–21, 2015.

[11] N. Boland, T. Kalinowski, and S. Kaur. Scheduling network maintenance jobs with release dates and deadlines to maximize total flow over time: Bounds and solution strategies. *Comput. Oper. Res.*, 64:113–129, 2015.

[12] N. Boland, T. Kalinowski, H. Waterer, and L. Zheng. Scheduling arc maintenance jobs in a network to maximize total flow over time. *Discrete Appl. Math.*, 163:34–52, 2014.

[13] N. L. Boland and M. W. P. Savelsbergh. Optimizing the hunter valley coal chain. In H. Gurnani, A. Mehrotra, and S. Ray, editors, *Supply Chain Disruptions: Theory and Practice of Managing Risk*, pages 275–302. Springer, 2012.

[14] M. Burcea, W. Hon, H. H. Liu, P. W. H. Wong, and D. K. Y. Yau. Scheduling for electricity cost in smart grid. In *Proceedings of the 7th International Conference on Combinatorial Optimization and Applications (COCOA)*, volume 8287 of *LNCS*, pages 306–317. Springer, 2013.

[15] G. M. Campbell. A two-stage stochastic program for scheduling and allocating cross-trained workers. *J. Oper. Res. Soc.*, 62(6):1038–1047, 2011.

[16] R. Canetti and S. Irani. Bounding the power of preemption in randomized scheduling. *SIAM J. Comput.*, 27(4):993–1015, 1998.

[17] S. Chaisiri, B.-S. Lee, and D. Niyato. Optimization of resource provisioning cost in cloud computing. *IEEE Trans. Serv. Comput.*, 5(2):164–177, 2012.

[18] J. Chang, S. Khuller, and K. Mukherjee. LP rounding and combinatorial algorithms for minimizing active and busy time. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 118–127. ACM, 2014.

[19] J. Chang, S. Khuller, and K. Mukherjee. Active and busy time minimization. In *Proceedings of the 12th Workshop on Models and Algorithms for Planning and Scheduling Problems (MAPSP)*, pages 247–249, 2015.

[20] M. Charikar, C. Chekuri, and M. Pál. Sampling bounds for stochastic optimization. In *Proceedings of the 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX) and 9th International Workshop on Randomization and Computation (RANDOM)*, volume 3624 of *LNCS*, pages 257–269. Springer, 2005.

[21] L. Chen, N. Megow, R. Rischke, and L. Stougie. Stochastic and robust scheduling in the cloud. Journal version of [22], in preparation.

[22] L. Chen, N. Megow, R. Rischke, and L. Stougie. Stochastic and robust scheduling in the cloud. In *Proceedings of the 18th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX) and 19th International Workshop on Randomization and Computation (RANDOM)*, volume 40 of *LIPIcs*, pages 175–186. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[23] L. Chen, N. Megow, R. Rischke, L. Stougie, and J. Verschae. Optimal algorithms and a PTAS for cost-aware scheduling. Journal version of [24], in preparation.

[24] L. Chen, N. Megow, R. Rischke, L. Stougie, and J. Verschae. Optimal algorithms and a PTAS for cost-aware scheduling. In *Proceedings of the 40th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 9235 of *LNCS*, pages 211–222. Springer, 2015.

[25] M. Cheung and D. B. Shmoys. A primal-dual approximation algorithm for min-sum single-machine scheduling problems. In *Proceedings of the 14th International Workshop on Approximation, Randomization, and Combinatorial Optimization (APPROX)*, volume 6845 of *LNCS*, pages 135–146. Springer, 2011.

[26] V. Cohen-Addad, Z. Li, C. Mathieu, and I. Milis. Energy-efficient algorithms for non-preemptive speed-scaling. In *12th International Workshop on Approximation and Online Algorithms (WAOA)*, volume 8952 of *LNCS*, pages 107–118. Springer, 2015.

[27] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 151–158. ACM, 1971.

[28] J. R. Correa, M. Skutella, and J. Verschae. The power of preemption on unrelated machines and applications to scheduling orders. *Math. Oper. Res.*, 37(2):379–398, 2012.

[29] K. Dhamdhere, V. Goyal, R. Ravi, and M. Singh. How to pay, come what may: Approximation algorithms for demand-robust covering problems. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 367–378. IEEE Computer Society, 2005.

[30] J. Z. Du and J. Y. T. Leung. Minimizing mean flow time with release time and deadline constraints. *J. Algorithms*, 14(1):45–68, 1993.

[31] S. Dye, L. Stougie, and A. Tomasgard. The stochastic single resource service-provision problem. *Naval Res. Logist.*, 50(8):869–887, 2003.

[32] W. L. Eastman, S. Even, and M. Isaac. Bounds for the optimal scheduling of n jobs on m processors. *Management Sci.*, 11(2):268–279, 1964.

[33] L. Epstein, A. Levin, A. Marchetti-Spaccamela, N. Megow, J. Mestre, M. Skutella, and L. Stougie. Universal sequencing on an unreliable machine. *SIAM J. Comput.*, 41(3):565–586, 2012.

[34] U. Feige, K. Jain, M. Mahdian, and V. S. Mirrokni. Robust combinatorial optimization with exponential scenarios. In *Proceedings of the 12th International Conference on Integer Programming and Combinatorial Optimization (IPCO)*, volume 4513 of *LNCS*, pages 439–453. Springer, 2007.

[35] M. Flammini, G. Monaco, L. Moscardelli, H. Shachnai, M. Shalom, T. Tamir, and S. Zaks. Minimizing total busy time in parallel scheduling with application to optical networks. *Theor. Comput. Sci.*, 411(40–42):3553–3562, 2010.

[36] H. L. Gantt. Organizing for work. New York: Harcourt, Brace and Howe, 1919.

[37] M. R. Garey and D. S. Johnson. "Strong" NP-completeness results: Motivation, examples, and implications. *J. ACM*, 25(3):499–508, 1978.

[38] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, New York, 1 edition, 1979.

[39] M. X. Goemans. Improved approximation algorthims for scheduling with release dates. In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 591–598. ACM/SIAM, 1997.

[40] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Tech. J.*, 45(9):1563–1581, 1966.

[41] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math.*, 17(2):416–429, 1969.

[42] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Ann. of Discrete Math.*, 5:287–326, 1979.

[43] A. Gupta, M. Pál, R. Ravi, and A. Sinha. Sampling and cost-sharing: Approximation algorithms for stochastic optimization problems. *SIAM J. Comput.*, 40(5):1361–1401, 2011.

[44] S. Ha. *Compile-time scheduling of dataflow program graphs with dynamic constructs*. PhD thesis, University of California, Berkeley, 1992.

[45] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *J. ACM*, 34(1):144–162, 1987.

[46] W. Höhn and T. Jacobs. On the performance of Smith's rule in single-machine scheduling with nonlinear cost. *ACM Trans. Algorithms*, 11(4):25, 2015.

[47] W. Höhn, J. Mestre, and A. Wiese. How unsplittable-flow-covering helps scheduling with job-dependent cost functions. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 8572 of *LNCS*, pages 625–636. Springer, 2014.

[48] J. R. Jackson. Scheduling a production line to minimize maximum tardiness. Technical Report 43, Managment Science Research Project, University of California, 1955.

[49] J. R. Jackson. An extension of Johnson's results on job lot scheduling. *Naval Res. Logist. Quart.*, 3:201–203, 1956.

[50] S. M. Johnson. Optimal two and three-stage production schedules with setup times included. *Naval Res. Logist. Quart.*, 1:61–67, 1954.

[51] T. Kalinowski, D. Matsypura, and M. W. Savelsbergh. Incremental network design with maximum flows. *European J. Oper. Res.*, 242(1):51–62, 2015.

[52] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller, J. W. Thatcher, and J. D. Bohlinger, editors, *Complexity of Computer Computations*, pages 85–103, Boston, MA, 1972. Springer.

[53] R. Khandekar, G. Kortsarz, V. S. Mirrokni, and M. R. Salavatipour. Two-stage robust network design with exponential scenarios. *Algorithmica*, 65(2):391–408, 2013.

[54] R. Khandekar, B. Schieber, H. Shachnai, and T. Tamir. Real-time scheduling to minimize machine busy times. *J. Sched.*, 18(6):561–573, 2015.

[55] A. J. Kleywegt, A. Shapiro, and T. Homem-de Mello. The sample average approximation method for stochastic discrete optimization. *SIAM J. Optim.*, 12(2):479–502, 2001.

[56] B. H. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer, Berlin und Heidelberg, 4 edition, 2008.

[57] J. Kulkarni and K. Munagala. Algorithms for cost-aware scheduling. In *Proceedings of the 10th International Workshop on Approximation and Online Algorithms (WAOA)*, volume 7846 of *LNCS*, pages 201–214. Springer, 2013.

[58] E. L. Lawler and J. Labetoulle. On preemptive scheduling of unrelated parallel processors by linear programming. *J. ACM*, 25(4):612–619, 1978.

[59] C.-Y. Lee. Machine scheduling with availability constraints. In J. Y.-T. Leung, editor, *Handbook of Scheduling*. CRC Press, 2004.

[60] J. Lenstra, A. R. Kan, and P. Brucker. Complexity of machine scheduling problems. *Ann. of Discrete Math.*, 1:343–362, 1977.

[61] S. Leonardi, N. Megow, R. Rischke, L. Stougie, C. Swamy, and J. Verschae. Scheduling with time-varying cost: Deterministic and stochastic models. Presentation at the 11th Workshop on Models and Algorithms for Planning and Scheduling Problems (MAPSP 2013), 2013.

[62] A. Marchetti-Spaccamela and L. Stougie. Personal communication, May 2, 2013.

[63] R. McNaughton. Scheduling with deadlines and loss functions. *Management Sci.*, 6:1–12, 1959.

[64] N. Megow and J. Verschae. Dual techniques for scheduling on a machine with varying speed. In *Proceedings of the 40th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 7965 of *LNCS*, pages 745–756. Springer, 2013.

[65] G. B. Mertzios, M. Shalom, A. Voloshin, P. W. H. Wong, and S. Zaks. Optimizing busy time on parallel machines. *Theor. Comput. Sci.*, 562:524–541, 2015.

[66] J. Mestre and J. Verschae. A 4-approximation for scheduling on a single machine with general cost function. *CoRR*, abs/1403.0298, 2014.

[67] S. G. Nurre, B. Cavdaroglu, J. E. Mitchell, T. C. Sharkey, and W. A. Wallace. Restoring infrastructure systems: An integrated network design and scheduling (INDS) problem. *European J. Oper. Res.*, 223(3):794 – 806, 2012.

[68] E. W. Parsons and K. C. Sevcik. Multiprocessor scheduling for high-variability service time distributions. In *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, volume 949 of *LNCS*, pages 127–145. Springer, 1995.

[69] M. L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, 2008.

[70] M. Queyranne and M. Sviridenko. A $(2+\varepsilon)$-approximation algorithm for the generalized preemptive open shop problem with minsum objective. *J. Algorithms*, 45(2):202–212, 2002.

[71] A. S. Schulz and M. Skutella. Random-based scheduling: New approximations and LP lower bounds. In *Proceedings of the International Workshop on Randomization and Approximation Techniques in Computer Science (APPROX/RANDOM)*, volume 1269 of *LNCS*, pages 119–133. Springer, 1997.

[72] A. S. Schulz and M. Skutella. Scheduling unrelated machines by randomized rounding. *SIAM J. Discrete Math.*, 15(4):450–469, 2002.

[73] D. B. Shmoys and M. Sozio. Approximation algorithms for 2-stage stochastic scheduling problems. In *Proceedings of the 12th International Conference on Integer Programming and Combinatorial Optimization (IPCO)*, volume 4513 of *LNCS*, pages 145–157. Springer, 2007.

[74] D. B. Shmoys and C. Swamy. An approximation scheme for stochastic linear programming and its application to stochastic integer programs. *J. ACM*, 53(6):978–1012, 2006.

[75] D. B. Shmoys and D. P. Williamson. *The Design of Approximation Algorithms*. Cambridge University Press, New York, 2011.

[76] R. A. Sitters. Approximability of average completion time scheduling on unrelated machines. In *Proceedings of the 16th Annual European Symposium on Algorithms (ESA)*, volume 5193 of *LNCS*, pages 768–779. Springer, 2008.

[77] M. Skutella. List scheduling in order of $\alpha$-points on a single machine. In E. Bampis, K. Jansen, and C. Kenyon, editors, *Efficient Approximation and Online Algorithms*, volume 3484 of *LNCS*, pages 250–291. Springer, 2006.

[78] W. E. Smith. Various optimizers for single-stage production. *Naval Res. Logist. Quart.*, 3:59–66, 1956.

[79] A. J. Soper and V. A. Strusevich. Power of preemption on uniform parallel machines. In *Proceedings of the 17th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, volume 28 of *LIPIcs*, pages 392–402, 2014.

[80] C. Swamy and D. B. Shmoys. Approximation algorithms for 2-stage stochastic optimization problems. *SIGACT News*, 37(1):33–46, 2006.

[81] C. Swamy and D. B. Shmoys. Sampling-based approximation algorithms for multistage stochastic optimization. *SIAM J. Comput.*, 41(4):975–1004, 2012.

[82] F. W. Taylor. *The Principles of Scientific Management*. Harper & Brothers Publishers, New York, London, 1911.

[83] G. Wan and X. Qi. Scheduling with variable time slot costs. *Naval Res. Logist.*, 57:159–171, 2010.

[84] G. Wang, H. Sun, and C. Chu. Preemptive scheduling with availability constraints to minimize total weighted completion times. *Ann. Oper. Res.*, 133:183–192, 2005.

[85] Y. Zhao, X. Qi, and M. Li. On scheduling with non-increasing time slot cost to minimize total weighted completion time. *J. Sched.*, pages 1–9, 2015.