Doctoral thesis

HIKING IN THE SCHEDULING LANDSCAPE: EXACT AND APPROXIMATION ALGORITHMS FOR PARALLEL MACHINES

Moritz Buchem

2022

HIKING IN THE SCHEDULING LANDSCAPE: EXACT AND APPROXIMATION ALGORITHMS FOR PARALLEL MACHINES

Dissertation

To obtain the degree of Doctor at Maastricht University,
on the authority of the Rector Magnificus, Prof. dr. P. Habibović,
in accordance with the decision of the Board of Deans,
to be defended in public
on Tuesday 29th of November 2022, at 16.00 hours

by

Moritz Yannik Buchem

This book has been typeset by the author using LaTeX.

Cover	Bregje Jaspers, 2022
Production	ProefschriftMaken || proefschriftmaken.nl
ISBN	978-94-6469-097-2

To my parents and my brother.

Not all those who wonder are lost.

- *Free to J.R.R. Tolkien, The Lord of the Rings*

Acknowledgments

This dissertation is the product of a four year journey as a PhD student and a nine year journey in the city of Maastricht. Taking this journey would not have been possible without my family, friends and colleagues who accompanied and supported me through different parts of this journey.

First of all, I would like to express my deepest gratitude to my promotor and supervisor, Tjark Vredeveld. In the last six years you have been a source of inspiration, guidance, mentorship in research, teaching and (academic) life. Thank you for opening your academic network to me and helping me to grow as a researcher myself. I know that I will always have someone to come to advice for and, hopefully, also some more research adventures together. Thank you for always having a minute (even if a minute turns into hours). I hope that you will find something to do with all this new available time. Simply: Thank you for everything, it means the world to me!

I would also like to say thank you to my co-supervisor Tim Oosterwijk. Thank you for always being available for constructive and thorough feedback on talks or drafts and the great support throughout the whole writing process of this dissertation. Thank you for making MOPD as much fun as possible in my first year of teaching. Furthermore, I would like to give a big thank you to the members of my assessment committee: Stan van Hoesel, André Berger, Nicole Megow, Laura Sanità and Martin Skutella. Thank you a lot for your time to read and approve my dissertation.

Next, I would like to thank everyone else who was a part of my academic life in the past years at Maastricht University. Thank you Karin van den Boorn, Vera Hoekstra and Yolanda Paulissen for always being there and having a friendly open ear when academic bureaucracy seemed like a maze. I would like to express my huge appreciation for all senior staff at the Quantitative Economics department and beyond for an inspiring research and teaching environment. A special thank you to Alain Hecq, André Berger, Lars Rohwedder, Rudolf Müller and Stan van Hoesel for the great teaching together. A special thank you to Lars. I really enjoyed discussing and doing research with you. Thank you for the many things I could learn from you. Also thank you to Alex Grigoriev for the great and fruitful discussions on our work together and beyond. An extra thank you to Christof Defryn for your supervision at the beginning of my PhD and allowing me to follow my own path.

A huge aspect in the past four years was the life as a PhD student which would not have been the same without my fellow PhD students. Elisa, thank you for being there since the beginning: from the first student house parties all the way until the final days as PhD students. Thank you for (almost) always being up for an after work drink and snack when both were strongly needed. You will always be a strong part of my Maastricht memories. Dewi, you were also part of my life in Maastricht for nine years after meeting the first time during the introduction days of the bachelor. Thank you for always spreading a good mood. Adam, thank you for all the fun times in and outside of the office and all the serious and non-serious discussions. Niloufar, thank you for sharing the joy of listening to records and repeatedly teaching me to count in Farsi. Maybe one day I will not need to start at one every few months. Julian, thank you for all the nice discussions and our joint work. Rasmus, thank you for the great company in the office. Ashkan and Arman, thank you for the great company during the last year of my PhD. Aditja, Alex, Benoit, Daniel, Farzaneh, Francesco, Francois, Freija, Joep, Li, Li, Luca, Marie, Niels, Roland, Sean, Shash, Son Tran and Thijs: thank you for being a part of the past four years, the mutual support and for the fun times! To all members of the PhD committee, thank you for the great time planning and attending the social events. To everyone I have mentioned and everyone I might have missed: Thank you and see you soon!

A big part of academic life is working and meeting with people all over the world. I would like to thank José Correa, José Verschae and Andy Wiese for hosting and inviting me to the workshop in Santiago de Chile in 2020. This was my first experience of a research visit and full of great discussions and inspiration. I would like to thank Martin Skutella for inviting me for a research visit to Berlin. Thank you Martin, Daniel Schmidt genannt Waldschmidt and all others from COGA and DISCO for welcoming me with open arms and the great three months (minus quarantine plus Besprechungen). An extra thank you to Daniel for being a great co-author and all the sometimes fruitful and sometimes frustrating research discussions. Thank you, Linda Kleist, for your collaboration, your inspiration and for keeping Daniel and me calm during the more frustrating meetings. Finally, I would like to thank all conferences, workshop and seminar communities that I have had the pleasure experiencing in the past years. The open, warm and inspirational environment you all have created is what makes research fun and productive.

Although this dissertation is the product of the last four years, achieving this would not have been possible without my friends and family. A big thank you to all of my friends who have kept me sane in the last four years. Thank you for all the fun times, all the distractions from research, always being there and for keeping up with me. I wish to express my deepest gratitude for my family for their unconditional love and support. Oma, Steffi, Renate, Sylvie, Lennard und Kathi, vielen Dank für all eure Unterstützung und Begleitung auf meinem Weg. Mama, Papa und Markus, ohne euch wäre Ich nicht dort wo Ich heute bin. Dafür werde Ich euch immer dankbar sein. Danke für all eure Liebe und Unterstützung, all eure Arbeit und Mühen. Danke dafür, dass Ihr immer mein sicherer Hafen seid. Wenn ihr nichts hört, geht's mir gut. Mit viel Liebe, euer Moritz!

Moritz Buchem
Munich
October 24, 2022

Contents

Chapter 1
Introduction

The scarcity of resources is omnipresent in society. This poses many challenges to decision makers in different areas ranging from service industries to manufacturing and transportation industries. One of these challenges is *scheduling*: the allocation of resources to activities over time. Scheduling appears in a diverse range of applications such as setting up educational timetables or planning industrial processes. The *theory of scheduling* is concerned with the modelling and solving of fundamental scheduling problems underlying the challenges appearing in practice [131]. In this thesis, five scheduling problems stemming from theory and practice are investigated.

This introductory chapter establishes important notions and concepts underlying the results in this thesis. In Section 1.1, the notion of *scheduling problems* is formally introduced and different variants of scheduling problems considered in this thesis are discussed. In Section 1.2, we introduce several notions of *computational complexity*. Section 1.3 introduces the concepts of *approximation and online algorithms* used in this thesis. Section 1.4 gives a detailed outline of the remainder of this thesis.

1.1 Scheduling Problems

The Merriam-Webster dictionary defines a schedule as "a procedural plan that indicates the time and sequence of each operation" [119]. Finding such schedules is a challenge occurring in various applications. Imagine the following situation faced by a car rental company. Each week the company has to clean its fleet of cars on a specific day. Two employees are tasked with cleaning the fleet and each of them can clean at most one car at the same time. The time it takes to clean each car is known. Furthermore, each car imposes a cost per unit of time for which it is unavailable (not cleaned yet). To solve this problem, the car cleaning company must find an order in which the cars are cleaned such that the total cost incurred by unavailability is minimized.

Modelling and solving problems of this form is the core challenge of *scheduling problems*, which are a sub-family of combinatorial optimization problems. In combinatorial optimization the goal is to find an optimal solution out of a set of finite or countably infinite possible solutions (we refer to Schrijver [142] or Korte and Vygen [101] for an extensive introduction into combinatorial optimization).

The scheduling problems considered in this thesis are concerned with processing n jobs on a set of m machines under some constraints so as to maximize or minimize a given objective function. We denote the set of jobs by $\mathcal{J} = [n]$, where $[n] = \{1, ..., n\}$. Similarly, we denote the set of machines by $\mathcal{M} = [m]$, where $[m] = \{1, ..., m\}$. In order to classify scheduling problems, Graham et al. [63] introduced the *three-field notation*. Here, a scheduling problem is defined by the *machine environment*, the *job characteristics* and the *optimality criterion* represented as $\alpha|\beta|\gamma$.

The first parameter α denotes the *machine environment*. This defines the nature and implicit restrictions of the machines underlying the scheduling problem. In this work we focus on the *parallel machine* environment, where n jobs need to be processed by one of m parallel machines that may schedule one job at a time. Furthermore, processing job j on machine i takes p_{ij} units of time. If the processing time depends solely on the job itself, we denote it by p_j. We distinguish between the following types of parallel machine environments:

- Single machine environment ($\alpha = 1$): There is a single machine available and, hence, $p_{1j} = p_j$.
- Identical parallel machine environment ($\alpha = P$): There are m machines available and the processing time solely depends on the job itself, i.e., $p_{ij} = p_j$ for all i and j.
- Uniform parallel machine environment ($\alpha = Q$): There are m machines available and the processing time of job j on machine i depends on the job as well as the speed s_i of machine i. Hence, $p_{ij} = p_j / s_i$ for all i and j.
- Unrelated parallel machine environment ($\alpha = R$): There are m machines available and p_{ij} can be arbitrary and job and machine specific.

The number of machines can either be fixed or given as part of the input. For the latter we add an indicating value m to the machine environment, e.g., $\alpha = Pm$ for the setting of identical parallel machines.

The second field β denotes the *job characteristics*. Job characteristics can be of diverse nature. On the one hand job characteristics may involve job-specific information and on the other hand job characteristics may involve relation between jobs or relations between jobs and machines. A scheduling problem can entail multiple job characteristics at once. Two examples of job characteristics considered in this thesis are:

- Release dates: If $r_j \in \beta$, then for every job j a release date r_j denotes earliest point in time at which a job can be processed.

- Due dates: If $d_j \in \beta$, then for every job j a due date d_j defines a point in time at which jobs should be completed and completion before or after this point is subject to incurred penalty costs.

Many other types of job characteristics are studied in the literature (see, e.g., [131]). One of these is *preemption* which allows for interrupting the processing of a job and resuming it later (possibly on different machines). In this thesis, we focus on the *non-preemptive* setting in which jobs are not allowed to be preempted. Note that whenever job characteristics impose no additional constraints, these job characteristics are omitted in the second field, e.g., if all jobs are released at time 0.

The final field γ indicates the *optimality criterion* of the problem. Various types of optimality criteria appear in the literature. In this thesis, the focus lies on optimality criteria concerned with *job completion times* and criteria concerned with *machine loads*.

- *Job completion optimality criteria*: The completion time of a job C_j is defined as the time at which job j has finished processing. The criterion is defined as a function of job completion times $f(C_1, \dots, C_n)$. Two well-studied optimality criteria of this form are: (1) makespan minimization in which the goal is to minimize the completion time of the final job, i.e., $f(C_1, \dots, C_n) = \max_j C_j$, and (2) the sum of (weighted) completion times, i.e., $f(C_1, \dots, C_n) = \sum_j (w_j) C_j$.
- *Machine load optimality criteria*: The load of a machine L_i is defined as the sum of processing times assigned to this machine. The optimality criterion is defined as a function of machine loads $f(L_1, \dots, L_m)$. In absence of any additional restrictions ($\beta = \emptyset$), makespan minimization is equivalent to minimizing the maximum machine load, i.e., $f(L_1, \dots, L_m) = \max_i L_i$. Another example is the Santa Claus problem in which the goal is to maximize the minimum machine load, i.e., $f(L_1, \dots, L_m) = \min_i L_i$.

This dissertation particularly focuses on problems of the types of *deterministic scheduling* and *scheduling with uncertainty*.

1.1.1 Deterministic scheduling

Deterministic scheduling encompasses scheduling problems in which all parameters are known and available without any degree of uncertainty.

In this setting, a *feasible schedule* is an allocation of jobs to machines over time satisfying all constraints. These constraints are implicitly given by the machine environment as well as the job characteristics. The quality of a schedule is measured by the objective given by the optimality criteria (γ). Depending on the type of criterion, an *optimal schedule* is a feasible schedule minimizing or maximizing this criterion.

Recall the situation of the car rental company.

EXAMPLE 1.1: THE CAR CLEANING PROBLEM

Suppose the car rental company has two employees responsible for cleaning the fleet. Furthermore, assume that the whole fleet is located at the station and each car can be cleaned at the beginning of the day.
This problem can be modelled as $P2||\sum_j w_jC_j$ as follows:

- $\alpha = P2$: The fleet needs to be cleaned by two employees with identical cleaning speeds.
- $\gamma = \sum_j w_jC_j$: The objective is to minimize total cost incurred by unavailability of cars.

1.1.2 Scheduling with Uncertainty

In contrast to deterministic scheduling, many real-world applications face factors of uncertainty. For scheduling problems there are many different types of uncertainty. On the one hand, the set of jobs may not be fully available at once and jobs are revealed to the scheduler one-by-one with the number of jobs not known in advance. On the other hand, job specific information such as the processing times, may not be known exactly and are better modelled using random variables rather than deterministic values. Two major models for coping with uncertainty in scheduling are *stochastic scheduling* and *online scheduling*.

In *stochastic scheduling* factors of uncertainty regarding the values of job parameters are modeled. Whereas in deterministic scheduling the processing times of jobs are assumed to be known, in stochastic scheduling these are not certain and are subject to random fluctuations. For example, only probability distributions underlying the processing times could be known rather than the actual processing times. Although Rothkopf [138] considered random processing times in the 1960s, the field of stochastic scheduling only became active field in the 1980s [44].

Online scheduling is concerned with uncertainty regarding the availability of information on job parameters. In offline scheduling all information is available a priori and decisions can be made by taking into account all job parameters. In online scheduling, however, this information is revealed job-by-job and decisions must be made on the go. These decisions can only take into account information that is available so far. In online scheduling one distinguishes between the *online-list* model and the *online-time* model as defined by Pruhs et al. [134]. In the online-list model, job information is made available one by one at time zero and decisions must be made irrevocably once a job is presented to the scheduler. In the online-time model, job information is made available upon the release dates of jobs and decisions can be delayed beyond the release date but decisions concerning a specific job may not be made before this job has been released.

Stochastic online scheduling, as introduced by Chou et al. [34] and Megow et al. [118], combines these two frameworks of uncertainty. Here, job processing times are only known in terms of underlying probability distributions and these are presented in an online manner.

The dynamic nature of uncertainty demands for dynamic solution methods rather than a deterministic schedule. Therefore, the aim is to find a *stochastic online scheduling policy* (SOS policy). Roughly, a stochastic online scheduling policy defines a decision on which job to schedule at any decision moment. We require a stochastic online scheduling policy to be *non-anticipatory*. This means that each decision may only use information revealed up to this moment and no information about future jobs nor about realized processing times of jobs that will be completed in the future [118].

Consider the stochastic online variant of the car cleaning problem.

EXAMPLE 1.2: THE CAR CLEANING PROBLEM WITH UNCERTAINTY

Consider the car cleaning problem when each car is still located at a customer's location and will be returned to the station during the day. In this situation, the rental company may face two sources of uncertainty:

- *Stochastic:* The state of a car depends on various factors such as the duration of the last rental period and the time passed since the last time the car was cleaned. Therefore, the time it takes to clean the car is subject to random fluctuations.
- *Online:* The fleet is still located at customer locations and no information is available before a car returns to the station. Furthermore, a car can only be assigned to an employee and scheduled after it has arrived at the station.

In this SOS setting, the problem is denoted by $P2|r_j|E[\sum_j w_j C_j]$.

1.2 Computational Complexity

Optimization problems differ in terms of how difficult it is to find (optimal) solutions. *Computational complexity theory* is concerned with classifying (optimization) problems according to this difficulty.

In order to solve optimization problems, researchers develop *algorithms*. An exact algorithm for an optimization problem is a computational recipe that gives a step-by-step guideline for finding an optimal solution to any instance of that problem. The computational complexity of a problem is closely related to the *efficiency* of the exact algorithms known for this problem. This efficiency can be measured by the *worst case running time* of the algorithm. Roughly speaking, this denotes the time it takes an algorithm to find the solution in the worst possible case. More formally, the running time of an algorithm is the worst case number of elementary operations needed before the algorithm terminates and returns an optimal solution. We are interested in finding the relation between the running time of an algorithm and the size of the input instance. The size of the input, denoted by $|I|$, is roughly the number of bits needed to encode the instance.

A naive type of an exact algorithm is *complete enumeration*. However, this often takes too much time in theory or practice. Consider for example the car cleaning problem without release dates. Here the total number of feasible schedules for cleaning n cars is 2^n. This means that the time it takes to find the optimal solution grows exponentially with the number of cars needed to be cleaned. Hence, when applying this technique for a large number of cars the rental company may have to wait until the end of the day before knowing in which order to clean the cars.

This motivates the development of *efficient algorithms*. In particular, we are interested in algorithms whose running times can be bounded by a polynomial in the input size.

Definition 1.1 (Polynomial time algorithm)**.** *Let Π be an optimization problem and* ALG *an exact algorithm for Π.* ALG *is said to be a polynomial time algorithm if there exists a polynomial p such that for every instance I of size $|I|$ the running time of* ALG *is bounded from above by $p(|I|)$.*

Consider the following variant of the car cleaning problem for which a polynomial time algorithm exists.

Example 1.3: Car Cleaning Problem with a Single Employee

Suppose that instead of two employees only one employee is available to clean the fleet. This variant of the car cleaning problem can be modelled as $1||\sum_j w_j C_j$. In this setting, cleaning cars according to the famous Smith's rule [150], which sorts the cars in non-increasing order of the unavailability costs compared to the time it takes to clean each car, finds an optimal schedule in polynomial time.

In order to classify optimization problems according to their difficulty, we distinguish between problems for which efficient exact algorithms are known to exist and those for which no efficient exact algorithm is known to exist and is deemed unlikely to exist. Before formalizing this distinction, using the notion of *hardness* of optimization problems, we first need to make a small excursion to the world of *decision problems*.

A *decision problem* is defined as a set of instances where each instance is a either a "yes" or a "no" instance. The aim is to decide whether an instance is a "yes" or a "no" instance. A *certificate* for an instance of a decision problem is used to verify whether this instance is a "yes" instance. An algorithm is said to solve a decision problem if it finds the correct answer for any instance. Decision problems are closely related to optimization problems. In particular, for a minimization (maximization) problem the corresponding decision problem is defined by the instance together with a value k and the question whether there exists a feasible solution with objective value at most (at least) k. Note that an (efficient) algorithm for the optimization problem can be used to solve the corresponding decision problem. Analogously, under some mild restrictions on the objective function, an efficient algorithm for the decision problem combined with a binary search over all possible values of k can be applied as an efficient algorithm for the optimization problem. Hence, the existence of efficient algorithms for an optimization problem is closely linked to the existence of efficient algorithms for the corresponding decision problem.

Towards the classification of decision problems the class $\mathcal{NP}$ has been introduced, see, e.g., Garey and Johnson [59]. This class includes all decision problems for which a "yes"-certificate can be verified in polynomial time in the input size. A subclass of $\mathcal{NP}$ is the class $\mathcal{P}$ which includes all decision problems for which an efficient exact algorithm is known to exist. A famous open problem is to show whether or not $\mathcal{P} = \mathcal{NP}$ and it is strongly believed that $\mathcal{P} \neq \mathcal{NP}$.

While the problems in $\mathcal{P}$ are known to be solvable in polynomial time, for other problems in $\mathcal{NP}$ no such algorithm is known to exist. For some of these problems, it can be shown that these are at least as hard as other problems in the class $\mathcal{NP}$. These are the so-called $\mathcal{NP}$-complete problems. Roughly speaking, the class of $\mathcal{NP}$-complete problems is the collection of the most difficult problems within the class $\mathcal{NP}$. More precisely, these problems can be transformed into any other problem in $\mathcal{NP}$ such that the existence of an efficient algorithm for an $\mathcal{NP}$-complete problem would imply the existence of an efficient algorithm for any problem in $\mathcal{NP}$ and, therefore, imply that $\mathcal{P} = \mathcal{NP}$. If a decision problem is $\mathcal{NP}$-complete even if the numeric values in the input are bounded by a polynomial in the input size, we say that this problem is $\mathcal{NP}$-complete in the *strong* sense.

Whereas it suffices to develop an efficient algorithm to show that a problem is in $\mathcal{P}$, it is not sufficient to fail at developing such an algorithm to prove that a problem is $\mathcal{NP}$-complete. Instead, for this purpose, we rely on *polynomial reductions*. A polynomial reduction is used to show that a problem in $\mathcal{NP}$ with an unknown complexity status is at least as hard as a problem which is known to be $\mathcal{NP}$-complete. In more detail, we consider a decision problem D_1 in $\mathcal{NP}$ and an $\mathcal{NP}$-complete decision problem D_2. If we can transform any instance of D_2 into an equivalent instance of D_1 in polynomial time and any solution to D_1 back into a solution to D_2, then any efficient algorithm for D_1 could be translated into an efficient algorithm for D_2, which is not possible, unless $\mathcal{P} = \mathcal{NP}$. Therefore, D_1 is at least as difficult as D_2 and also $\mathcal{NP}$-complete.

Many decision problems have been shown to be $\mathcal{NP}$-complete over the years. Cook [36] laid the foundation of this by showing that the satisfiability problem is $\mathcal{NP}$-complete. Later, Karp [97] showed $\mathcal{NP}$-completeness of 21 fundamental decision problems. Since then many more problems have shown to be $\mathcal{NP}$-complete, see e.g., Garey and Johnson [59].

Returning to the world of optimization problems, an optimization problem is said to be easy or solvable if the corresponding decision problem is in $\mathcal{P}$. In contrast, an optimization problem is said to be $\mathcal{NP}$-hard if the corresponding decision problem is $\mathcal{NP}$-complete. Furthermore, an optimization problem is said to be strongly $\mathcal{NP}$-hard if the decision problem is $\mathcal{NP}$-complete in the *strong* sense. For the optimization problem this implies that no *pseudo-polynomial time* exact algorithm exists, unless $\mathcal{P} = \mathcal{NP}$. A pseudo-polynomial time algorithm is one that is polynomial if all numeric values in the input are bounded by a polynomial in the input size.

Definition 1.2 (Pseudo-polynomial time algorithm)**.** *Let Π be an optimization problem and ALG an algorithm. ALG is said to be a pseudo-polynomial time algorithm if there exists a polynomial p such that for every instance I of size $|I|$ with maximum numeric value $\langle I \rangle$ the running time of ALG is bounded from above by $p(|I|, \langle I \rangle)$.*

We have seen that for a single employee the car cleaning problem is in $\mathcal{P}$. Let us now consider the setting with two employees or release dates.

Example 1.4: Computational Complexity of the Car Cleaning Problem

The decision version of the car cleaning problem can be formulated as:

"Given an instance I of the car cleaning problem and an integer $k > 0$, does there exist a feasible schedule with total weighted completion time at most k?"

Bruno et al. [22] showed that this decision problem is $\mathcal{NP}$-complete via a reduction from the famous partition problem. This implies that the car cleaning problem with two employees is $\mathcal{NP}$-hard. Thus, no polynomial time algorithm for finding an optimal schedule exists, unless $\mathcal{P} = \mathcal{NP}$.

1.3 Approximation Algorithms

There are many alternative approaches to cope with $\mathcal{NP}$-hard problems. The study of exact algorithms is concerned with the challenge of developing algorithms that work well on realistic instances of the problems but for which no polynomial running time is guaranteed. Another popular direction of research is to consider special cases of $\mathcal{NP}$-hard problems and develop exact efficient algorithms for these.

In contrast to these approaches, the study of *approximation algorithms* is concerned with developing algorithms that find solutions that are provably close to the optimal solution. Two common paradigms of approximation are *multiplicative approximation* and *additive approximation.* For an extensive introduction to approximation algorithms we refer to the books by Williamson and Shmoys [169] or Vazirani [161].

The quality of a *multiplicative approximation algorithm* is measured in relative terms compared to the optimal solution. Formally, a multiplicative approximation algorithm is defined as follows.

Definition 1.3 (Multiplicative approximation algorithm)**.** *Consider a minimization (maximization) problem* Π*. An algorithm* ALG *is a multiplicative* α*-approximation with* $\alpha > 1$ *(*$\alpha < 1$*), if for any instance* I *of* Π*,* ALG *finds a feasible solution with value at most (at least)* α *times the optimal value. Here,* α *is called the multiplicative approximation guarantee of* ALG*.*

Analogously to the concept of multiplicative approximation, the quality of an *additive approximation algorithm* is measured in absolute terms compared to the optimal solution value.

Definition 1.4 (Additive approximation algorithm)**.** *Consider an optimization problem* Π *and let* $\rho(I) > 0$ *be a quantity defined for every instance* I *of* Π*. An algorithm* ALG *is an additive* $\rho(I)$*-approximation if for any instance* I *of* Π*,* ALG *finds a feasible solution such that the absolute difference between the value of this solution and the optimal value is at most* $\rho(I)$*. Here,* $\rho(I)$ *is referred to as the additive approximation guarantee of* ALG*.*

We are particularly interested in finding approximation algorithms for which the running time is bounded by a polynomial in the input size. The ultimate goal in both approximation paradigms is to reach approximation guarantees as small as possible, i.e., α should be as close to 1 as possible or $\rho(I)$ as close to 0 as possible. To formalize this quest, we define the concept of *approximation schemes*.

For the setting of multiplicative approximation algorithms, a *polynomial time approximation scheme (PTAS)* is a family of polynomial time approximation algorithms such that for any approximation guarantee a corresponding approximation algorithm is defined. Formally, a PTAS is defined as follows.

Definition 1.5 (Polynomial time approximation algorithms (PTAS))**.** *Consider a minimization (maximization) problem* Π*. A polynomial time approximation scheme is a family of* $(1+\epsilon)$*-approximation algorithms (*$(1-\epsilon)$*-approximation algorithms) running in polynomial time for any fixed* $\epsilon > 0$*.*

We say that a PTAS is an *efficient polynomial time approximation scheme (EPTAS)* if the running time is given as $f(1/\epsilon)poly(|I|)$, where $f(1/\epsilon)$ is some (not necessarily polynomial) function depending on ϵ and $poly(|I|)$ is a polynomial function of the input size. If $f(1/\epsilon)$ is a polynomial function, we refer to the approximation scheme as a *fully polynomial time approximation scheme (FPTAS)*.

In the setting of additive approximations, approximation schemes can be defined in a similar fashion. Here, an approximation scheme is defined as a family of additive $\epsilon\rho(I)$ approximation algorithms based on a quantity $\rho(I)$ referred to as the approximation parameter. More formally, this is defined as follows.

Definition 1.6 (additive PTAS (add-PTAS))**.** *Consider a minimization problem* Π *and let* $\rho(I) > 0$ *be a quantity defined for any instance* I *of* Π. *A polynomial time additive approximation scheme is a family of additive* $\epsilon\rho(I)$-*approximation algorithms running in polynomial time for any fixed* $\epsilon > 0$.

Similar to PTASs, we say that a add-PTAS is an *additive efficient polynomial time approximation scheme (add-EPTAS)* if the running time is of the form $f(1/\epsilon)poly(|I|)$, where $f(1/\epsilon)$ is some (not necessarily polynomial) function depending on ϵ and $poly(|I|)$ is a polynomial function of the input size. If $f(1/\epsilon)$ is a polynomial function, we refer to the approximation scheme as a *additive fully polynomial time approximation scheme (add-FPTAS)*.

While for some problems the quest of finding polynomial time approximation schemes has been successful, for other problems this has been shown to be unreachable. In the study of the *hardness of approximation* we investigate the limitations of approximation algorithms. An important implication of the complexity of an optimization problem on the hardness of approximation, is that a strongly $\mathcal{NP}$-hard problem does not admit an FPTAS nor an add-FPTAS, unless $\mathcal{P} = \mathcal{NP}$. Hence, the best one can wish for here is the existence of an EPTAS or an add-EPTAS. Moreover, for some optimization problems it has even been shown that no polynomial time (additive) approximation algorithm can achieve better approximation guarantees than some value α^* ($\rho^*(I)$), unless $\mathcal{P} = \mathcal{NP}$.

The study of approximation algorithms and hardness of approximation is an important and active pillar of the research within the field of scheduling. Graham [61] derived a constant multiplicative approximation guarantee for makespan minimization on identical parallel machines which also yields an additive $p_{\max}$ approximation, where $p_{\max}$ is the maximum processing time. For makespan minimization on identical parallel machines with m being a constant, $Pm||C_{\max}$, Sahni [140] introduced a FPTAS. For the case that m is part of the input, $P||C_{\max}$, Hochbaum and Shmoys [80] developed PTAS. This was later improved to an EPTAS, see e.g. Jansen et al. [90]. Due to $P||C_{\max}$ being strongly $\mathcal{NP}$-hard, this is the best one can achieve. A prominent example of a problem which encounters limits of approximation is makespan minimization on unrelated machines, for which Lenstra et al. [107] showed that no polynomial time approximation algorithm can attain a guarantee less than $\frac{3}{2}$, unless $\mathcal{P} = \mathcal{NP}$.

Recall the car cleaning problem. While for a single employee the problem without release dates can be solved in polynomial time, it turns out to be $\mathcal{NP}$-hard for two employees. However, scheduling jobs greedily according to Smith's rule turns out to find a provably good schedule for any instance.

Example 1.5: Approximation algorithms for the Car Cleaning Problem

Adapting Smith's rule [150] to the setting of two employees works as follows: we assign the next job to the employee with the earliest possible starting time for this job. Kawaguchi and Kyan [98] showed that this way we construct a schedule with multiplicative approximation guarantee of $(1+\sqrt{2})/2$-approximation in time $O(n \log n)$.

When uncertainty is taken into account, the notion of approximation algorithms may be adapted. In stochastic online scheduling, where the objective value of a policy is a random variable, we follow the definition of Möhring et al. [122] of the *performance guarantee* of SOS policies. The performance guarantee compares the expected solution value of an SOS policy with an optimal stochastic offline scheduling policy which has access to all stochastic job information a priori.

Definition 1.7 (Performance guarantee in SOS)**.** *Let P be a stochastic online scheduling problem with a minimization (maximization) objective and let $OPT(I)$ be the stochastic offline optimal policy for instance I. A stochastic online scheduling policy Π is said to attain a performance guarantee of c with $c > 1$ ($c < 1$) if for any instance I it finds a solution of expected value at most (at least) $c \cdot E\left[OPT(I)\right]$. We refer to c as the performance guarantee.*

In addition to the performance guarantee, the *asymptotic behavior* of a policy is of interest when assessing the quality of a policy. The *asymptotic optimality* of a policy implies that the performance guarantee goes towards 1 as the instance increases.

Definition 1.8. *Consider a stochastic online scheduling problem with a minimization (maximization) objective and let $OPT(I)$ be the stochastic offline optimal policy for instance I. Let Π be a stochastic online scheduling policy with performance guarantee c. Π is said to be asymptotically optimal if*

$$c \xrightarrow{n\to\infty} 1$$

or equivalently

$$\frac{E\left[\Pi(I)\right] - E\left[OPT(I)\right]}{E\left[OPT(I)\right]} \xrightarrow{n\to\infty} 0,$$

where $n \to \infty$ indicates that the number of jobs in instance I tends to infinity. Here, $E\left[\Pi(I)\right]$ is the expected objective of Π on instance I.

EXAMPLE 1.6: SOS POLICY FOR THE CAR CLEANING PROBLEM

Consider the stochastic online variant of the car cleaning problem in which cars are still located at customer locations and return to the station during the day. Megow et al. [118] consider the following adaptation of Smith's rule to this setting. Whenever a car arrives at the station it is greedily assigned to one of the employees. For each individual employees, a shifted version of Smith's rule is applied. This SOS policy yields a performance guarantee of $1+\max\{1+\frac{\delta}{\alpha}, \alpha+\delta+\frac{\Delta+1}{4}\}$, where α is an arbitrary parameter and δ and Δ are parameters measuring the stochasticity of the input.

1.4 Outline of Thesis

The research underlying this thesis entails the study of various scheduling problems. Here, we summarize the main results and outline the remainder of this thesis.

Chapter 2 is devoted to the study of load balancing problems on identical parallel machines under the additive approximation paradigm. More precisely, we consider three different load balancing problem: the classic scheduling problem of makespan minimization for which the objective is to minimize the maximum machine load, the max-min allocation problem in which we want to maximize the minimum machine load and the envy-minimizing Santa Claus problem where the goal is to minimize the difference between the maximum machine load. The first two problems are well-studied from a multiplicative approximation perspective and (efficient) polynomial time approximation schemes exist [3, 30, 79, 89, 90, 109]. The third problem, however, does not admit any multiplicative approximation guarantee, unless $\mathcal{P} = \mathcal{NP}$, as it is strongly $\mathcal{NP}$-hard to decide whether there exists a solution such that all machines have the same load [58]. For all three problems, we devise additive approximation schemes with $\rho(I) = p_{\max}$ being the maximum processing time of all jobs. In the case of an arbitrary number of machines, we develop an add-PTAS. Hereto, we first introduce a new mixed-integer linear programming relaxation which integrally assigns slots to machines and fractionally assigns jobs to slots. We identify structural properties of (near-)optimal solutions leading to techniques to find such solutions in polynomial time for any $\epsilon > 0$. To complement this, we develop a local-search algorithm inspired by the rounding techniques for the restricted assignment problem [91, 92, 155]. The local search technique considers any feasible solution of the relaxation and returns an integral solution such that the error introduced on each machine load is at most $\epsilon \cdot p_{\max}$. When the number of machines is fixed, we show that the techniques by Sahni [140] and Woeginger [171] for FPTASs, can be adjusted to obtain add-FPTASs.

In *Chapter 3*, we focus on a natural extension of makespan minimization on identical parallel machines with a single common server introduced by Hall et al. [72] and Kravchenko and Werner [103]. In this context, jobs consist of a pre-processing and processing component. While the processing component has to be executed on one of m identical parallel machines the pre-processing component has to be executed by an external server which can serve at most one machine at a time. Jobs must be executed non-preemptively without any interruption between pre-processing and processing. We generalize this problem by considering a third job component (post-processing) which has to be executed non-preemptively and without interruption on the server after the job has been processed on one of the machines. Moreover, we combine this model with the concept of *machine conflicts* defining for each pair of machines if they can access the server in parallel or not. As done before by Chrobak et al. [35] and Höhne and van Stee [81], machine conflicts are represented as a conflict graph where two machines are said to be conflicting if they are adjacent. We refer to this generalized problem as SMC. In this work, we restrict our attention towards the special case where all three job components are equal to 1 for every job (denoted by SMC-UNIT). By establishing a connection between this problem and the task of finding a maximum induced bipartite subgraph of the conflict graph, we prove that SMC-UNIT on m machines does not allow for a $\mathcal{O}(m^{1-\varepsilon})$-approximation algorithm for any $\varepsilon > 0$. Motivated by this inapproximability result, we consider special graph classes. We translate polynomial time (approximation) algorithms for finding maximum induced bipartite subgraphs into approximation algorithms for SMC-UNIT implying approximation results for various graph classes. Regarding the original setting considered in [72, 103], we show that SMC-UNIT can be solved in polynomial time when the conflict graph is complete. Most prominently, we devise a polynomial time algorithm to solve SMC-UNIT on bipartite conflict graphs offering possible insights into obtaining stronger approximation results for other graph classes. Finally, we generalize some of the results to the more general setting of identical jobs by showing how to translate a suitable collection of independent sets of the conflict graph into schedules with a provable approximation guarantee.

Chapter 4 is concerned with a just-in-time scheduling problem with quadratic penalties. Just-in-time scheduling has received a lot of attention and has been studied in various settings [137]. We consider the problem of minimizing the (weighted) squared deviation from a common or distinct job due dates. We particularly focus on the special case where all jobs have equal processing times. In the unweighted setting, we show that the problem can be solved in polynomial time. The algorithmic techniques to achieve this are based on structural insights on optimal schedules on a single machine combined with the optimality of so-called balanced schedules for the identical machine environment. If the job weights are not all the same, we show that the problem can be solved in polynomial time for a single common due date if this due date is sufficiently large to allow jobs to be scheduled (almost) symmetrically around it. If this is not the case, we present an algorithm exponential in the number of machines. Finally, we consider the weighted version on a single machine with a constant number of different due dates and devise an add-FPTAS with $\rho(I) = w_{\max}$ being the maximum job weight.

In *Chapter 5*, we turn our attention to a stochastic online scheduling problem. More precisely, we consider the problem of minimizing total weighted expected completion time on a special case of uniform parallel machines where each machine speed is either 1 or $s > 1$. Following the work of Megow et al. [118] for identical parallel machines and Gupta et al. [67] for unrelated machines, we investigate the performance of fixed assignment policies for the online-list as well as the online-time model. Fixed assignment policies are a class of stochastic online scheduling policies which first irrevocably assign each job to a machine once it is presented to the scheduler and then use a single machine policy for each machine individually. We adapt and refine these policies to the special case of uniform machines with two different speeds. In the online-list model, we adapt the greedy fixed assignment policies [67, 118] to the uniform parallel machine setting by taking into account machine speeds. We derive a performance guarantee depending on the number of (fast) machines and the degree of variation underlying the processing time distributions. Furthermore, we show that this policy for the online-list model is asymptotically optimal under the assumption that the expected processing times and weights are bounded from above and below. For the online-time, model we introduce a random fixed assignment policy and combine it with a single machine policy introduced by Gupta et al. [67]. Finally, we complement the theoretical investigation of the presented policies with a computational study analysing the realized performance of the policies and comparing it to the theoretically expected performance.

Chapter 6 studies a scheduling problem with uncertainty occurring in inland waterway transportation. Inland waterways play a significant role in the transportation of goods [53]. An important aspect of the coordination and management of inland waterways is the presence of locks which are responsible for connecting river segments with different water levels and transport vessels from one segment to the other. We consider a velocity optimization problem from the perspective of a single vessel facing stochastic waiting times at the lock. These waiting times are due to uncertainty of the lock processing time of a second vessel approaching the lock. The goal is to find find a choice of velocities for any point in time such that our vessel crosses two river segments connected by a lock and the total expected fuel consumption is minimized. We formulate a mathematical model and develop two solution techniques. The first technique is a pseudo-polynomial time algorithm which finds a near-optimal solution and the second technique is a simple heuristic based on a fixed arrival time at the lock. To evaluate these solution techniques, we conduct a computational study regarding the efficiency in terms of fuel savings and the simplicity in terms of implementation.

1.5 Publications

The work in this dissertation is based on the following publications.
Published:

- M. Buchem, L. Rohwedder, T. Vredeveld, and A. Wiese. "Additive Approximation Schemes for Load Balancing Problems". In: *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2021
- M. Buchem and T. Vredeveld. "Performance analysis of fixed assignment policies for stochastic online scheduling on uniform parallel machines". In: *Computers & Operations Research* 125 (2021)
- M. Buchem, J. A. P. Golak, and A. Grigoriev. "Vessel velocity decisions in inland waterway transportation under uncertainty". In: *European Journal of Operational Research* 296 (2022), pp. 669–678

Accepted for publication:

- M. Buchem, L. Kleist, and D. Schmidt genannt Waldschmidt. "Scheduling with Machine Conflicts". To appear in: *Proceedings of International Workshop on Approximation and Online Algorithms 2022 (WAOA 2022)*. A pre-published version can be found in [24].

Chapter 2

Additive approximation schemes for load balancing on identical machines

2.1 Introduction

Load balancing problems are among the classical problems in the literature on approximation algorithms. This dates back to the seminal work of Graham [61, 62] on the first approximation algorithms for makespan minimization on identical parallel machines. In this chapter, we particularly focus on a family of load balancing problems on identical parallel machines. Formally, these problems are defined by a set $\mathcal{M}$ of m identical parallel machines and set $\mathcal{J}$ of n jobs. For each job $j \in \mathcal{J}$ a processing time p_j is given and each job must be processed exactly once while a machine may only process one job at a time in a non-preemptive fashion. The task is to find an assignment $\sigma : \mathcal{J} \to \mathcal{M}$ so as to minimize or maximize a given load balancing function $f(L_1, \ldots, L_m)$, where the load of machine i, denoted by L_i, is defined as the sum of the processing times assigned to machine i.

Parts of this chapter have been published in: M. Buchem, L. Rohwedder, T. Vredeveld, and A. Wiese. "Additive Approximation Schemes for Load Balancing Problems". In: *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021).* Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2021

We consider the following three load balancing problems:

i *Makespan minimization*: The aim is to minimize the maximum machine load, i.e., $f(L_1, \ldots, L_m) = \max_{i \in \mathcal{M}} L_i$. Recall that this is denoted by $Pm||C_{\max}$ or $P||C_{\max}$ when the number of machines is constant or part of the input, respectively.

ii *Max-min load*: The aim is to maximize the minimum machine load, i.e., $f(L_1, \ldots, L_m) = \min_{i \in \mathcal{M}} L_i$. This problem is also referred to as the Santa Claus problem in the restricted assignment model [14].

iii *Envy-minimizing Santa Claus problem*: The aim is to minimize the difference between the maximum and the minimum machine load, i.e., $f(L_1, \ldots, L_m) = \max_{i \in \mathcal{M}} L_i - \min_{i \in \mathcal{M}} L_i$. This problem also appears as the minimum envy problem in the area of goods allocation [112].

The problem of finding a schedule which assings the same load to each machine is weakly $\mathcal{NP}$-hard when the number of machines is constant [59] and strongly $\mathcal{NP}$-hard when the number of machines is arbitrary [58]. Thus, the considered load balancing problems exhibit the same complexity. The hardness of these problems has led to a lively history of research on approximation algorithms and approximation schemes.

Previous Work. Makespan minimization is a classical scheduling problems on parallel machines. The work on this problem dates back to the first approximation algorithms introduced by Graham [61, 62]. Graham shows that the so-called LIST-scheduling algorithm, which greedily assigns jobs one-by-one to the machine with the lowest load, yields a multiplicative $(2 - {}^1/_m)$ approximation. In his follow-up work, Graham [62] introduces a variant of LIST-scheduling which assigns jobs in non-increasing order of processing times, the longest processing time first (LPT) rule, and shows that this admits a multiplicative approximation guarantee of $({}^4/_3 - {}^1/_{3m})$. When the number of machines is constant, Sahni [140] developed a fully polynomial time approximation scheme (FPTAS) based on a dynamic programming algorithm and an interval partitioning technique. This technique was later generalized by Woeginger [171]. For the setting where m is part of the input, Hochbaum and Shmoys [80] introduced the first polynomial time approximation scheme (PTAS). Their algorithm is based on the dual approximation approach and runs in time $O((n/\epsilon)^{1/\epsilon^2})$. This result led to a lively chain of research on improving the running time of the approximation scheme towards an efficient polynomial time approximation scheme (EPTAS), see e.g, [3, 30, 79, 89, 90, 109]. The latest improvement is due to Jansen et al. [90]. They design an EPTAS based on structural insights of the configuration-IP running in time $2^{O(1/\epsilon \log^4(1/\epsilon))} + poly(n)$. The configuration-IP (and its relaxation) have been proven to be powerful tools in obtaining approximation algorithms for scheduling and packing problems and date back to the work of Gilmore and Gomory [60]. The result by Jansen et al. [90] settles the gap to the lower bound on the running time of any EPTAS (unless the Exponential Time Hypothesis (ETH) fails) shown by Chen et al. [30].

The objective of maximizing the minimum load was referred to as the *max-min allocation* problem by Chakrabarty [28]. Deuermeyer et al. [45] investigate the performance of LPT for this objective. Based on a minimal counterexample analysis, they show that LPT yields a multiplicative approximation guarantee of $3/4$. Csirik et al. [38] later showed that the exact multiplicative approximation guarantee of LPT is $(3m-1/4m-2)$. When the number of machines is a constant, the techniques of Sahni [140] and Woeginger [171] can be used to construct an FPTAS for this objective on identical machines as well. In case the number of machines is part of the input, Woeginger [170] introduced an (efficient) polynomial time approximation scheme. A similar result is shown by Alon et al. [3] who prove the existence of EPTASes for a general family of scheduling problems with load balancing objectives. Following the same generalization, Jansen et al. [90] prove that their EPTAS for makespan minimization can be extended to the objective of maximizing the minimum machine load. In the restricted assignment setting where each job can only be assigned to a subset of machines, Bansal and Sviridenko [14] coined this problem as the *Santa Claus* problem. In this setting, various approximation algorithms are known (see, e.g., [5, 6, 31, 40, 56, 69, 76]) and the currently best known result is a polynomial time $(4+\epsilon)$ multiplicative approximation due to Davies et al. [40] and Cheng and Mao [31]. Recently, Haxell and Szabó showed that the integrality gap of the configuration-LP is bounded from above by roughly 3.534 [76].

The *envy-minimizing Santa Claus* problem has been considered before by Lipton et al. [112] for the more general case of unrelated machines. For this case they present an algorithm with an additive approximation guarantee of $\max_{i\in\mathcal{M},j\in\mathcal{J}} p_{ij}$. Note that it is strongly NP-hard to decide whether or not the envy is 0 [58]. Therefore, unless $\mathcal{P} = \mathcal{NP}$, there cannot exist any polynomial time approximation algorithm with any (multiplicative) performance guarantee.

While the majority of research has focused on finding multiplicative approximation algorithms (or schemes), additive approximation algorithms have only been considered for a small number of problems in the literature. One of these is the well-studied bin-packing problem where task is to pack a set of items into the least possible number of bins with fixed capacity. Kamarkar and Karp [96] developed the first additive approximation algorithm for this problem with an approximation guarantee of $O(\log^2 \mathrm{OPT})$, where OPT is the number of bins in an optimal solution. Hoberg and Rothvoss [78] improved this by presenting algorithm with an additive approximation guarantee of $O(\log \mathrm{OPT})$ after an initial improvement to $O(\log \mathrm{OPT} \cdot \log\log \mathrm{OPT})$ by Rothvoss [139]. Recently, in follow-up work of the results in this chapter, Inoue and Kobayashi [86] presented an add-PTAS for maximizing Nash social welfare with identical additive valuations. The Nash social welfare is measured as the geometric mean of the received valuations of the agents (or in terms of machine scheduling the geometric mean of the machine loads). Vizing's algorithm [166] yields an additive approximation of 1 for the graph coloring problem.

Our Contributions and Outline. In this chapter, we complement this long history of research by approaching the considered load balancing problems under the additive approximation paradigm. We devise additive approximation schemes with $\rho(I) = p_{\max}$ being the maximum processing time of all jobs. For the makespan minimization and max-min load objectives this gives an improvement over multiplicative approximation schemes whenever the maximum processing time is much smaller than the value of an optimal solution. For the envy-minimizing Santa Claus problem, this demonstrates that additive approximation schemes can lead to non-trivial guarantees even for problems for which no multiplicative guarantee is possible. While a straightforward analysis of LIST-scheduling as well as LPT gives an additive approximation guarantee of $O(p_{\max})$ for all three problems, new techniques are needed to obtain additive approximation schemes.

The main contribution of this chapter are additive polynomial time approximation schemes (add-PTASes) when the number of machines is part of the input as well as additive fully polynomial time approximation schemes (add-FPTASs) when the number of machines is a constant.

In Section 2.2, we take a closer look at the case where m is part of the input and devise add-PTASes for this setting. The approximation schemes for this setting are based on a generalization of the considered load balancing problems. For this generalization, we introduce a new mixed integer linear programming relaxation. In a first step, we derive structural properties of solutions to this relaxation. These structural properties provide insights for developing algorithmic techniques to find these solutions. To complement these techniques we introduce a rounding procedure based on a local search algorithm. This rounding procedure finds an integral solution to the original such that the load of each machine is close to the load of each machine in the solution to the relaxation. In Section 2.3, we develop add-FPTASs following similar techniques as Sahni [140] and Woeginger [171] for fixed m.

2.2 Additive approximation schemes for arbitrary number of machines

In this section, we develop add-PTASes for makespan minimization, the max-min load problem and the envy-minimizing Santa Claus problem when m is part of the input.

Before discussing the details of our algorithms, we first discuss the difficulties that standard techniques to obtain multiplicative approximation schemes face when aiming towards additive approximation guarantees. Two techniques that are commonly used by (E)PTASes (such as the algorithm by Jansen et al. [90]) are the notion of small or large jobs and rounding processing times such that only a constant number of different processing times is considered. For the former, a job j is said to be small if $p_j \leq \epsilon\mathrm{OPT}$ and large otherwise. This leads to the property that in an optimal solution only a constant number of large jobs is assigned to each machine. Therefore, methods based on enumeration or integer programming in constant dimension such as Lenstra's algorithm [108] can be used after rounding or grouping job sizes. The final schedule is obtained by greedily assigning small jobs. From an additive approximation point of view this is problematic if $p_{\max} < \epsilon\mathrm{OPT}$ as all jobs are greedily assigned giving an additive error of $p_{\max} > \epsilon p_{\max}$. Adjusting this approach by defining a job j as a large job if $p_j > \epsilon p_{\max}$ fails since this does not guarantee that there are only a constant number of large jobs assigned to each machine in an optimal solution. The rounding of processing times can be useful as it allows to apply techniques known to solve the problem to optimally for a constant number of different processing times [139]. However, when unrounding the processing times an error of $\epsilon \sum_j p_j > \epsilon p_{\max}$ may be introduced on each machine.

This underlines the need of new (non-trivial) machinery to obtain additive approximation schemes. Hereto, we consider the corresponding feasibility problem which we refer to as the *target load balancing problem*.

Definition 2.1. *In the target load balancing problem we are given a set of jobs $\mathcal{J}$ with a processing time p_j for each $j \in \mathcal{J}$ and a set of machines $\mathcal{M}$ with values ℓ, u. The goal is to find an assignment of jobs to machines such that for every machine $i \in \mathcal{M}$ it holds that $L_i \in [\ell, u]$.*

The target load balancing problem in combination with guesses on the machine lower and upper bounds generalizes the considered load balancing problems. For example, for makespan minimization let $\ell = 0$ and $u = T$ with T being a guess of the maximum machine load. The task is then to find a solution to the corresponding target load balancing problem for the smallest possible T which can be done via binary search. Similarly, the target load balancing problem to generalizes the max-min load and envy-minimizing Santa Claus problem.

In the remainder of this section, we develop a polynomial time algorithm which for any $\epsilon > 0$ either finds an approximate solution to the target load balancing problem such that each machine load lies in the interval $[\ell - \epsilon p_{\max}, u + \epsilon p_{\max}]$ or asserts that no feasible solution exists. This, in combination with guessing possible candidates of ℓ and u yields add-PTASes for makespan minimization, the max-min load problem as well as the envy-minimizing Santa Claus problem. We start by introducing a new mixed integer linear programming relaxation for the target load balancing problem, which we refer to as the *slot-MILP*. The slot-MILP can be regarded as a strengthened variant of the *assignment linear program* which is shown below.

$$\begin{aligned}
\sum_{i \in \mathcal{M}} x_{i,j} &= 1 && \forall j \in \mathcal{J} \\
\ell \le \sum_{j \in \mathcal{J}} p_j x_{i,j} &\le u && \forall i \in \mathcal{M} \\
x_{i,j} &\ge 0 && \forall j \in \mathcal{J}, i \in \mathcal{M}
\end{aligned}$$

As it evenly distributes jobs to machines, the assignment-LP has an integrality in $O(p_{\max})$ which can be observed even for a single job.

To strengthen the assignment-LP, we introduce the additional requirement that for each group of jobs with similar processing times an integral number of jobs is assigned to each machine. In particular, we first group jobs with similar processing times (without rounding) and introduce integer variables denoting the number of slots for each job group on each machine. Then, jobs are fractionally assigned to these slots. More formally, the slot-MILP is defined as follows. Let $\mathcal{I}$ be an instance of the target load balancing problem and $\epsilon > 0$ such that $1/\epsilon \in \mathbb{N}$. We define the slot-MILP for the target load balancing as follows. First, we group jobs according to their processing times into sets $\mathcal{J}_1, \dots, \mathcal{J}_{1/\epsilon}$, where for $k \in [1/\epsilon]$ the set $\mathcal{J}_k$ contains all jobs $j \in \mathcal{J}$ with $p_j \in ((k-1)\epsilon \cdot p_{\max}, k\epsilon \cdot p_{\max}]$. Then, for each machine $i \in \mathcal{M}$ and each job type $k \in [1/\epsilon]$, we introduce an integer variable $y_{i,k}$ denoting the number of slots of type k assigned to i. Additionally, we introduce fractional assignment variables $x_{i,j}$ for every $i \in \mathcal{M}$ and $j \in \mathcal{J}$.

$$
\begin{aligned}
\sum_{i \in \mathcal{M}} x_{i,j} &= 1 && \forall j \in \mathcal{J} \\
\ell \leq \sum_{j \in \mathcal{J}} p_j x_{i,j} &\leq u && \forall i \in \mathcal{M} \\
\sum_{j \in \mathcal{J}_k} x_{i,j} &= y_{i,k} && \forall i \in \mathcal{M}, \forall k \in [1/\epsilon] \\
x_{i,j} &\geq 0 && \forall j \in \mathcal{J}, i \in \mathcal{M} \\
y_{i,k} &\in \mathbb{N}_0 && \forall i \in \mathcal{M}, k \in [1/\epsilon]
\end{aligned}
$$

Since the number of integer variables depends linearly on the number of machines, the application of techniques for MILPs with a constant number of integral variables such as Lenstra's algorithm [108] is not possible. Instead, in Section 2.2.1 we derive a structural property of the slot-MILP which gives insights for algorithmic techniques to find a feasible solution. While this solution is feasible for the slot-MILP it may still give a fractional assignment of jobs to machines. In Section 2.2.2

we introduce a rounding procedure which finds a solution to the slot-MILP with an integral assignment of jobs such that the lower and upper bounds on the machine loads are violated by at most $\epsilon p_{\max}$. This rounding procedure is based on local search techniques introduced for the restricted assignment problem [91, 92, 155] where jobs have restrictions on the machines they can be assigned to. In fact, the successful application of this rounding procedure shows that the slot-MILP is stronger than both the assignment-LP which has an additive integrality gap of up to $p_{\max}$ and the configuration-LP which has an additive integrality gap of at least $\mathrm{OPT}/1023 \geq p_{\max}/1023$ [106]. In Section 2.2.3 we present an alternative structural property of the slot-MILP leading to an improvement in the running time of the add-PTASes.

2.2.1 The first step towards the add-PTAS: Solving the slot-MILP

The first task towards towards an add-PTAS is to either find a solution to the slot-MILP in polynomial time or to assert that there is no feasible solution to the instance of the target load balancing problem. In the following, we derive a structural property of the slot-MILP which allows us to guess the integer slot variables and solve the remaining linear program. To this end, for a given a solution (x, y), we define the slot vector of machine i, denoted by y_i, as the $(1/\epsilon)$-tuple $(y_{i,1}, \ldots, y_{i,1/\epsilon})$. Based on an argument by Eisenbrand and Shmonin [50] for bin-packing, we show that there exists an optimal solution using only a small number of different slot vectors.

Lemma 2.2. *If the slot-MILP is feasible, then there exists a solution (x, y) such that for all $i, i' \in \mathcal{M}$ with $y_i \equiv y_{i'} \bmod 2$ it holds that $y_i = y_{i'}$.*

Proof. Let (x, y) be a solution to the slot-MILP minimizing

$$\sum_{i \in \mathcal{M}} \sum_{k=1}^{1/\epsilon} \|y_{i,k}\|_2. \tag{2.1}$$

Suppose towards contradiction that there exist two machines with similar but not equal slot vectors, i.e., there exist machines i_1, i_2 with $y_{i_1} \equiv y_{i_2} \bmod 2$, but $y_{i_1} \neq y_{i_2}$.

We will construct a new solution (x', y') with a lower value of the potential function (2.1). The idea is to leave all machines other than i_1 and i_2 the same, while evenly distributing all jobs between i_1 and i_2.

Hence, for all j we set $x'_{i,j} = x_{i,j}$ for all $i \notin \{i_1, i_2\}$ and $x'_{i_1,j} = x'_{i_2,j} = (x_{i_1,j} + x_{i_2,j})/2$. Furthermore, for all k we set $y'_{i,k} = y_{i,k}$ for all $i \notin \{i_1, i_2\}$ and $y'_{i_1,k} = y'_{i_2,k} = (y_{i_1,k} + y_{i_2,k})/2$. Now, we check that the solution remains feasible. For every job $j \in \mathcal{J}$, we have

$$\begin{aligned}\sum_{i\in\mathcal{M}} x'_{i,j} &= x'_{i_1,j} + x'_{i_2,j} + \sum_{i\notin\{i_1,i_2\}} x'_{i,j} \\ &= \frac{x_{i_1,j} + x_{i_2,j}}{2} + \frac{x_{i_1,j} + x_{i_2,j}}{2} + \sum_{i\notin\{i_1,i_2\}} x_{i,j} \\ &= \sum_{i\in\mathcal{M}} x_{i,j} = 1.\end{aligned}$$

Hence, all jobs are still fully assigned. Next, consider the machine loads. For all machines $i \notin \{i_1, i_2\}$ the load does not change and, hence, the load of machine i remains within $[\ell, u]$. For i_1 and i_2 the new load according to (x', y') is equal to the average of the loads according to (x, y):

$$\begin{aligned}\sum_{j\in\mathcal{J}} p_j x'_{i_1,j} = \sum_{j\in\mathcal{J}} p_j x'_{i_2,j} &= \sum_{j\in\mathcal{J}} p_j \frac{x_{i_1,j} + x_{i_2,j}}{2} \\ &= \frac{1}{2}\sum_{j\in\mathcal{J}} p_j x_{i_1,j} + \frac{1}{2}\sum_{j\in\mathcal{J}} p_j x_{i_2,j} \\ &\leq \frac{u}{2} + \frac{u}{2} = u,\end{aligned}$$

and

$$\begin{aligned}\sum_{j\in\mathcal{J}} p_j x'_{i_1,j} = \sum_{j\in\mathcal{J}} p_j x'_{i_2,j} &= \sum_{j\in\mathcal{J}} p_j \frac{x_{i_1,j}+x_{i_2,j}}{2} \\ &= \frac{1}{2}\sum_{j\in\mathcal{J}} p_j x_{i_1,j} + \frac{1}{2}\sum_{j\in\mathcal{J}} p_j x_{i_2,j} \\ &\geq \frac{\ell}{2}+\frac{\ell}{2} = \ell.\end{aligned}$$

Next, consider the integrality constraint, i.e., the correct number of jobs must be assigned to every machine with respect to the slot variables. Again, this remains true for all machines $i \notin \{i_1, i_2\}$ since neither the assignment nor the slot variables change. Let $k \in [1/\epsilon]$. Since $y_{i_1,k} \equiv y_{i_2,k}$, we have that $y_{i_1,k} + y_{i_2,k}$ is even. It follows that

$$\sum_{j\in\mathcal{J}_k} x'_{i_1,j} = \sum_{j\in\mathcal{J}_k} x'_{i_2,j} = \frac{y_{i_1}+y_{i_2}}{2}$$

is integral. Thus, (x', y') is a feasible solution to the slot-MILP. Furthermore, since $y_{i,1} \neq y_{i,2}$, the triangle inequality implies the following for at least one k:

$$\|y'_{i_1,k}\|_2 + \|y'_{i_2,k}\|_2 = 2\left\|\frac{y_{i_1,k}+y_{i_2,k}}{2}\right\|_2 < \|y_{i_1,k}\|_2 + \|y_{i_2,k}\|_2.$$

As all other machines remain unchanged, we can conclude that (x', y') has a lower value of the potential function (2.1) than (x, y). □

This structural property allows us to find a solution to the slot-MILP or assert that there is no solution in polynomial time.

Lemma 2.3. *Let $\mathcal{I}$ be an instance of the target load problem. We can find a feasible solution to the corresponding slot-MILP or assert that there is no solution to $\mathcal{I}$ in time $m^{O(2^{1/\epsilon})} \cdot n^{O(1/\epsilon \cdot 2^{1/\epsilon})}$.*

Proof. By Lemma 2.2 we know that there is a feasible solution using at most $2^{1/\epsilon}$ many different slot vectors. Therefore, we may guess all values of $y_{i,k}$ (up to permutations of machines) of the optimal solution as follows. For each of the $2^{1/\epsilon}$ necessary slot vectors we guess:

- The number of machines having this slot vector and
- for each $k \in \{1, ..., 1/\epsilon\}$ we guess the value of $y_{i,k}$ for each machine $i \in \mathcal{M}$ with this slot vector.

Since the machines are identical, it suffices to guess the number of each slot vector, rather than guessing the slot vector of each machine. The total number of guesses is bounded by $m^{O\left(2^{1/\epsilon}\right)} \cdot n^{O\left(1/\epsilon \cdot 2^{1/\epsilon}\right)}$. For each guess, a linear program remains to be solved. This can be done in polynomial time using known techniques [159, 160]. The running time due to solving these linear programs gives an additional factor of $n^{O(1)}$ which is dominated by the double exponential dependency in $1/\epsilon$. If one of the guesses was correct, then the linear program must have a feasible solution. Otherwise, we can assert that there is no feasible solution to $\mathcal{I}$. $\square$

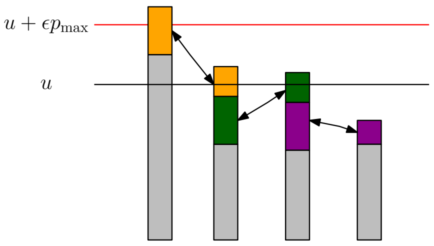
2.2.2 The second step towards the add-PTAS: Rounding the slot-MILP

The algorithm underlying Lemma 2.2 finds a solution to the slot-MILP. However, this is not a feasible solution to the target load balancing problem due to the fractional assignment of jobs. In this section, we describe a rounding procedure which turns the feasible solution to the slot-MILP into an integral assignment of jobs satisfying the initial slot variables. The goal is to round the solution such that the target load intervals are violated by at most $\epsilon \cdot p_{\max}$ for every machine.

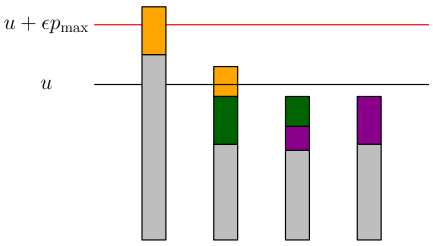
The local search starts with an arbitrary assignment of jobs to machines based on a solution (x, y) to the slot-MILP. Hereto, we can imagine that each machine $i \in \mathcal{M}$ has $y_{i,k}$ slots for the jobs in $\mathcal{J}_k$, for each $k \in \{1, ..., 1/\epsilon\}$. These slots are said to be of *type* k. Due to the feasibility of (x, y), we have $\sum_{i \in \mathcal{M}} y_{i,k} = |\mathcal{J}_k|$ for every type k. Hence, arbitrarily assigning $y_{i,k}$ jobs of type k to each machine i gives a feasible assignment satisfying the slot variables. This initial solution may include machines whose loads are not in $[\ell - \epsilon \cdot p_{\max}, u + \epsilon \cdot p_{\max}]$. We refer to a machine with load less than $\ell - \epsilon \cdot p_{\max}$ as *underloaded* and a machine with load more than $u + \epsilon \cdot p_{\max}$ as *overloaded*. Starting from this initial assignment, we iteratively repair the solution by decreasing (increasing) the load of overloaded (underloaded) machines. This is done in two stages. In the first stage, we repair all overloaded machines while not decreasing the load on any machine too much ensuring that no machine will become underloaded. In the second, stage we repair all remaining underloaded machines and ensure that no machine becomes overloaded. It is important to remark that the rounding procedure is not based on the assignment variables x but only on the slot variables y. The assignment variables are, however, needed to ensure the existence of a fractional solution satisfying the slot variables.

In the following we give a detailed description and analysis of the first stage of the local search procedure. The second stage is described in Appendix 2.A and follows similar ideas and arguments.

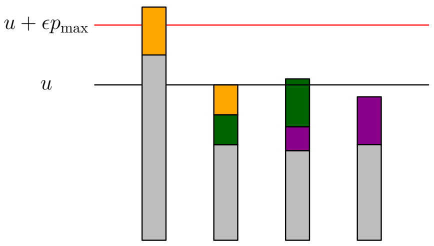
The first stage: Repairing overloaded machines. As mentioned above, the initial assignment of jobs to machines satisfying the load variables may lead to some machines being overloaded. In this stage of the local search, the load on these machines is decreased iteratively while not making non-overloaded machines overloaded. To this end, in each iteration the goal is to swap a job of type k currently assigned to an overloaded machine with a smaller job of type k assigned to a non-overloaded machine. We call the non-overloaded machines for which such a pair can be found *direct neighbors* of overloaded machines. If for some overloaded machines one of its direct neighbors has load at most u, we may execute a swap. This swap decreases the load on the overloaded machine while increasing the load of its direct neighbor. However, since the load of the direct neighbor was at most u, we do not create new overloaded machines. If, however, for all overloaded machines all direct neighbors have load strictly more than u, we must first decrease the load on one of these direct neighbors. Therefore, we again search among all neighbours of these machines to find a swappable pair of jobs. If such a pair exists and the neighboring machine has load at most u we execute the swap and decrease the load of a direct neighbor of an overloaded machine. Otherwise, we repeat the procedure again. With this procedure, we eventually find a sequence of swaps such that the load of the direct neighbor of some overloaded machines is at most u and the desired initial swap can be executed. Figure 2.1 shows a visualization of a successful sequence of such swaps.



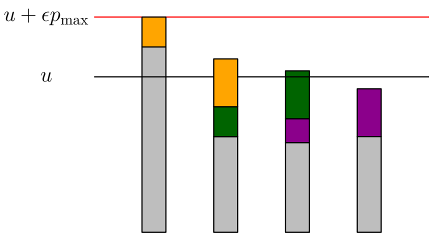
(a) Initial assignment and swaps.



(b) Assignment after first swap.



(c) Assignment after second swap.



(d) Final assignment.

Figure 2.1: Visualization of the first stage of local search procedure.

More formally, the first stage works as follows. As long as there is at least one overloaded machine, let $\mathcal{M}_1$ be the set of all overloaded machines. We define the *direct neighbors* of a machine $i \in \mathcal{M}_1$ as all $i' \in \mathcal{M} \setminus \mathcal{M}_1$ such that for some job type $k \in \{1, ..., 1/\epsilon\}$ there exists a pair j, j' with $p_j > p_{j'}$ such that j is assigned to machine i and j' is assigned to machine i'. We refer to such a pair j and j' as a pair of swappable jobs. Let $\mathcal{M}_2$ be the set of all direct neighbors of machines in $\mathcal{M}_1$. Using the same definition we construct the set $\mathcal{M}_3$ including the direct neighbors of $\mathcal{M}_2$. Repeating this construction, the aim is to find the smallest h such that for some machine in $\mathcal{M}_{h-1}$ there exists a direct neighbor in $\mathcal{M}_h$ with load at most u. For this pair of machines we execute the swap of the corresponding swappable pair of jobs. Note that for set $\mathcal{M}_h$, $h - 1$ can be seen as the distance between a machine in $\mathcal{M}_h$ and an overloaded machine in $\mathcal{M}_1$. Intuitively, we either find a direct neighbor of an overloaded machine with load at most u and execute a swap as to decreasing the load of the overloaded machine or we find a machine closest to an overloaded machine as to eventually decrease the load on a direct neighbor of an overloaded machine. Once a swap is executed, we repeat the procedure. In particular, we do not reuse the constructed sets $\mathcal{M}_1, \ldots, \mathcal{M}_{h+1}$ for the next swap but instead we forget these sets before the next swap starts. Once there are no more overloaded machines, the local search enters its second stage where a similar procedure is used to repair all underloaded machines (see Appendix 2.A). Note that in the first stage no machine becomes underloaded as we only decrease the load of a machine with load at least u. Hence, after decreasing the load this machine has load at least $u - \epsilon p_{\max}$.

In order to show the correctness and running time of the first stage of the local search, we first prove that we can find a swappable pair of jobs in polynomial time in any iteration.

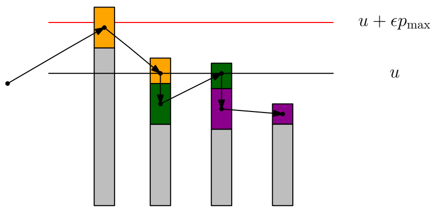
Lemma 2.4. *If there is an overloaded machine, there exists a pair of swappable jobs j, j' and finding such a pair can be done in time $O(n^2)$.*

Proof. Consider an iteration of the first stage of the local search with machine sets $\mathcal{M}_1, \ldots, \mathcal{M}_\ell$. Suppose towards contradiction that there exists no swappable pair of jobs. This means that for all job types k there exists no pair $j, j' \in \mathcal{J}_k$ such that j is assigned to a machine $i \in \mathcal{M}_\ell$ with load more than $u + \epsilon \cdot p_{\max}$ and job j' is assigned to a machine $i' \notin \mathcal{M}_1 \cup \ldots \cup \mathcal{M}_\ell$ and $p'_j < p_j$. If this is the case, the machines in $\mathcal{M}_1 \cup \ldots \cup \mathcal{M}_\ell$ are assigned the smallest jobs of every job type k while each having a load strictly more than u. Thus, the average load of these machines is strictly larger than u implying that not even a fractional assignment of all jobs can guarantee that the load of all these machines is at most u. This gives a contradiction to the feasibility of the initial solution to the slot-MILP.

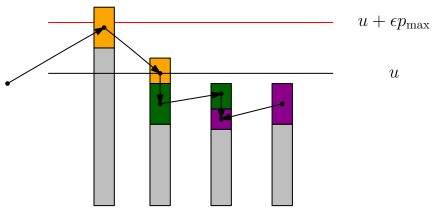
In order to find a pair of jobs to swap we need to consider each pair of jobs needs to be considered at most once. Hence, in the worst case we have a running time of $O(n^2)$ for each iteration. □

To complete the running time analysis of the first stage of the local search procedure it remains to be shown that the number of total iterations can also be bounded by a polynomial in the input size. For this purpose, we give an alternative formulation of the local search as a repeated breadth-first search (BFS).

An alternative formulation of the local search procedure. We construct a weighted, directed graph $G = (V, A)$ representing the current assignment of jobs to machines. We define V containing one special vertex, the source s and one vertex for each slot such that $|V| = n + 1$. Slot vertices are associated with a machine and a job group. Next, we define the set of arcs A. First of all, for all slot vertices $u, v \in V$ associated with the same machine we introduce arcs (u, v) and (v, u) with weight 0, i.e., vertices associated with the same machine form a clique. Secondly, for $v, w \in V$ we introduce an arc (v, w) with weight 1 if: (1) v and w are not associated with the same machine, (2) v and w are associated with the same job type and (3) v is currently assigned a larger job than w. Finally, we introduce arcs (s, v) of weight 0 for all $v \in V$ being associated with an overloaded machine. In each iteration, the algorithm performs a BFS on the graph above starting in the source vertex. Once a slot vertex on a machine with load at most u is reached, the algorithm selects the arc (v, w) over which the machine was reached and swaps the jobs assigned to the slots v and w. Note that the distance between the source vertex and a slot vertex is the number of machines needed to cross before reaching a machine with load at most u. This procedure is repeated until no machine is overloaded. Figure 2.2 illustrates an iteration of the BFS algorithm in which a path from s to a slot vertex on a machine with load at most u is found.



(a) Current assignment and path from s to slot vertex



(b) Assignment after swapping last two jobs on the path

Figure 2.2: Visualization of the breadth-first search

We next show that the distance between the source vertex and the slot vertices does not decrease after a swap.

Lemma 2.5. *In any iteration of the first stage of the local search procedure, the distance from s to any other slot does not decrease by a swap.*

Proof. Note that a swap reverses the direction of an arc. Let (v, w) be the slots whose jobs were swapped. Then, the key idea of the proof is that the distance in the original graph to w is equal to the shortest path to v plus 1. After the swap, this shortest path is eliminated whereas all other paths from s to w remain. Therefore, the distance from s to w will not decrease. Furthermore, the distance from s to v did not change due to the swap.

Formally, we observe how the graph changes when swapping two jobs. Let $d(w)$ denote the distance from s to some slot vertex w before the swap. Clearly, removing arcs from the graph cannot decrease $d(w)$ for any $w \in V$. Furthermore, arcs between slots of the same machine do not change and no new arcs are added from the source, since no overloaded machines are created by a swap. Hence, it is sufficient to look at the possible changes in arcs of weight 1. Although such an arc (v, w) might be added to the graph, we will show that this happens only when $d(w) \leq d(v) + 1$. Therefore, adding such an arc cannot decrease any distances, since the first part of a shortest path using (v, w) could always be replaced by a path to w without this arc.

Now we have to check that these are the only changes made to the graph. Let v, w be the slots in which we exchange the jobs. The size of the job in v decreases while the size of the job in w increases. We only need to look at the incoming and outgoing arcs of v and w, since all other arcs remain the same.

Consider the incoming arcs of v. Since the size of the job in v decreases, there could be new incoming arcs. Let (w', v) be an arc of weight 1 that is added. This means the job in slot w' has a larger size than the job on v. Either w' is a slot on the same machine as w or $(w', w) \in A$ before the swap. The former case implies that $d(w') = d(w) = d(v) + 1$. In the latter case we have $d(v) = d(w) - 1 \leq d(w')$. No outgoing arc from v can be added, since the size of v's job decreases.

For w, the reverse holds. Since the size of its job increases, no incoming arc can be added. As for the outgoing arcs, let (w, w') be an outgoing arc added by the swap. Then either v and w' are slots on the same machine or (v, w') was is in the graph before the swap. In the former case, $d(w') = d(v) = d(w) - 1$. In the latter case, $d(w') \leq d(v) + 1 = d(w)$. □

Using Lemmas 2.4 and 2.5 we can bound the maximum number of swaps necessary in the first stage of the algorithm. Intuitively, this is due to the distance between the source vertex and a slot vertex being equivalent to the distance between the machine corresponding to this slot vertex and a slot on an overloaded machine. Since this distance is bounded and does not decrease after a swap, the claim follows.

Lemma 2.6. *The first stage of the local search terminates after at most $O(n^3)$ swaps.*

Proof. Let $j_1, \ldots, j_n$ be the jobs $\mathcal{J}$ in increasing order of size. Define $d(j_i)$ as the distance from s to the slot to which j_i is assigned. We claim that the following potential function increases with every swap:

$$\sum_{i=1}^{n} i \cdot d(j_i)$$

Since this function is integral and bounded by n^3, this proves the lemma. To prove the claim, let j_k, j_h be the jobs that are swapped. Assume that $k < h$, i.e., $p_{j_k} < p_{j_h}$. Let d, d' be the distance functions before and after the swap. Based on Lemma 2.5 we know that $d'(j_i) \geq d(j_i)$ for all $i \notin \{k, h\}$, $d'(j_h) \geq d(j_k)$ and $d'(j_k) \geq d(j_h)$, since these jobs swapped their slots. Hence, it follows that

$$\begin{aligned} k \cdot d'(j_k) + h \cdot d'(j_h) &\geq k \cdot d(j_h) + h \cdot d(j_k) \\ &= k \cdot d(j_h) + (h-k) \cdot d(j_k) + k \cdot d(j_k) \\ &> k \cdot d(j_h) + (h-k) \cdot d(j_h) + k \cdot d(j_k) \\ &= h \cdot d(j_h) + k \cdot d(j_k). \end{aligned}$$

Hence, the first stage of the local search procedure terminates after at most $O(n^3)$ many iterations. □

Following similar arguments as the proofs for the first stage, the second stage of the local search procedure can be shown to run in polynomial time as well (see Appendix 2.A). Hence, the total running time of the local search procedure is given as follows.

Lemma 2.7. *Given a solution to the slot-MILP, in time $O(n^5)$ we can compute an integral solution to slot-MILP such that $\sum_{j\in\mathcal{J}} p_j x_{i,j} \in [\ell - \epsilon \cdot p_{\max}, u + \epsilon \cdot p_{\max}]$ for each machine $i \in \mathcal{M}$.*

The polynomial time additive approximation scheme for the target load balancing problem follows from Lemmas 2.3 and 2.7.

Theorem 2.8. *There is an algorithm for the target load balancing problem with a running time of $m^{O(2^{1/\epsilon})} \cdot n^{O(1/\epsilon \cdot 2^{1/\epsilon})}$ that computes an integral solution in which the load of each machine $i \in \mathcal{M}$ is in $[\ell - \epsilon \cdot p_{\max}, u + \epsilon \cdot p_{\max}]$, or asserts that there is no feasible solution.*

From this we obtain add-PTASes for the considered load balancing problems based on guessing the machine target load intervals up to multiples of $\epsilon \cdot p_{\max}$. For makespan minimization we set $\ell = 0$ and guess u as a multiple of $\epsilon \cdot p_{\max}$ in the interval $[\frac{1}{m}\sum_{j\in\mathcal{J}} p_j, \frac{1}{m}\sum_{j\in\mathcal{J}} p_j + p_{\max}]$, where the upper bound follows from greedily assigning jobs to machines. For the max-min problem, set $u = \sum_{j\in\mathcal{J}} p_j$ and guess ℓ within the interval $[\frac{1}{m}\sum_{j\in\mathcal{J}} p_j - p_{\max}, \frac{1}{m}\sum_{j\in\mathcal{J}} p_j]$, where the lower bound follows from greedily assigning jobs to machines. For both problems this amounts to a total of at most $f(\epsilon) = 1/\epsilon$ guesses. For the envy-minimizing Santa Claus problem we need to guess both ℓ and u simultaneously from the intervals used for the max-min load problem and makespan minimization, respectively. Hence, a total number of $f(\epsilon) = 1/\epsilon^2$ guesses is needed. This gives the following result.

Corollary 2.9. *There exist additive polynomial time approximation schemes (add-PTASes) for makespan minimization, the max-min load and the envy minimizing Santa Claus problem with running time $f(\epsilon)\cdot m^{O(2^{1/\epsilon})} \cdot n^{O(1/\epsilon \cdot 2^{1/\epsilon})}$.*

2.2.3 Improving the running time of the add-PTAS

The running time of the add-PTASes based on Theorem 2.8 is $f(\epsilon) \cdot m^{O\left(2^{1/\epsilon}\right)} \cdot n^{O\left(1/\epsilon \cdot 2^{1/\epsilon}\right)}$ and double exponential with respect to $1/\epsilon$. In the following, we show how to improve the running time's dependency on $1/\epsilon$. Since the running time of the local search procedure is independent of $1/\epsilon$, this improvement must take place in the first step of the add-PTASes, that is, solving the slot-MILP.

The key idea behind the improvement is to sacrifice the exactness of the solution to the slot-MILP and allow it to slightly violate the target load intervals. To this end, we introduce a weaker version of the target load balancing problem with lower and upper bounds $\ell' = \ell - \delta \cdot p_{\max}$ and $u' = u + \delta \cdot p_{\max}$ for some $\delta > 0$. The corresponding weakened version of the slot-MILP is referred to as slot-MILP'. We next show that we can find a solution to slot-MILP' in time $m^2(\frac{n}{\delta\epsilon})^{O(1/\epsilon)}$ or assert that no solution to the slot-MILP exists.

First, we derive an alternative structural property of the slot-MILP. Given a solution (x, y), we denote by $z_{i,k}$ the average size of jobs of type k assigned to machine i such that

$$z_{i,k} \cdot y_{i,k} = \sum_{j \in \mathcal{J}_k} p_j x_{i,j}.$$

This allows us to freely choose the value of $z_{i,k}$ when $y_{i,k} = 0$. This is important for the upcoming structural property. Furthermore, we refer to as $z_{i,k} \cdot y_{i,k}$ as the total size of slots for type k on machine i.

Using these definitions, we show that there exists a solution the slot-MILP such that for every job type k the average job sizes are non-decreasing with respect to the machines. Furthermore, for any type k and for every prefix of machines the total size of the jobs of type k assigned to these machines is at least the total sum of processing times of the smallest jobs which could be integrally assigned to the slots on these machines. For each integer n', let $\mathcal{J}_k^{\min}(n') \subseteq \mathcal{J}_k$ be the n' smallest jobs in $\mathcal{J}_k$.

Lemma 2.10. *If the slot-MILP is feasible, then there is a solution (x, y) and a corresponding vector $\{z_{i,k}\}_{i\in\mathcal{M},k\in[1/\epsilon]}$ such that*

$$\sum_{\ell'=1}^{\ell} y_{\ell,k} z_{\ell,k} \geq \sum_{j\in\mathcal{J}_k^{\min}(y_{1,k}+\cdots+y_{\ell,k})} p_j \qquad \begin{array}{l}\forall k \in [1/\epsilon]\\ \forall \ell \in \mathcal{M}\end{array} \tag{2.2}$$

$$\sum_{i\in\mathcal{M}} y_{i,k} z_{i,k} = \sum_{j\in\mathcal{J}_k} p_j \qquad \forall k \in [1/\epsilon] \tag{2.3}$$

$$z_{\ell,k} \leq z_{\ell+1,k} \qquad \begin{array}{l}\forall k \in [1/\epsilon]\\ \forall \ell \in \mathcal{M} \setminus \{m\}.\end{array} \tag{2.4}$$

Proof. Let (x, y) be a feasible solution to the slot-MILP with corresponding vector of average job sizes $\{z_{i,k}\}_{i\in\mathcal{M},k\in[1/\epsilon]}$. Then, Conditions (2.2) and (2.3) follow directly from the feasibility of (x, y) and the definition of the average job sizes. Therefore, it remains to be shown that we can find an ordering of machines such that Condition (2.4) holds. Since we are considering identical machines this ordering corresponds to an index rearrangement.

Let $\hat{z}_1 \leq \cdots \leq \hat{z}_{\bar{n}}$ be an ordering of the $\bar{n} = |\{(i, k) : y_{ik} > 0\}|$ values $z_{i,k}$ for all $i \in \mathcal{M}, k \in [1/\epsilon]$ with $y_{ik} > 0$. Assume that (x, y) is the solution maximizing the following potential function

$$\sum_{i=1}^{\bar{n}} n^{2(m/\epsilon-i)} \hat{z}_i. \tag{2.5}$$

We will now show that we can find an ordering of the machines such that Condition (2.4) holds, or otherwise unless (x, y) does not maximize (2.5). Thus, giving a proof by contradiction.

Let i be the machine minimizing $\sum_{k=1}^{1/\epsilon} z_{i,k}$. Then, for all other machines i' one of the following two conditions must be true: (1) $z_{i,k} \leq z_{i',k}$ for all $k \in [1/\epsilon]$ or (2) $z_{i,k} > z_{i',k}$ for some k. If for all machines i'

condition (1) holds, we relabel machine i as machine 1. Otherwise, we consider a machine $i' \neq i$ such that for some k we have

$$z_{i,k} > z_{i',k} \tag{2.6}$$

Then, as i minimizes $\sum_{k=1}^{1/\epsilon} z_{i,k}$ we know that there must exist $\overline{k} \neq k$ with

$$z_{i,\overline{k}} < z_{i',\overline{k}} \tag{2.7}$$

Due to the free choice of the value of the average size of jobs assigned, whenever the number of slots for a certain job type on a machine is 0, we know that $y_{i,k}, y_{i,\overline{k}}, y_{i',k}, y_{i',\overline{k}} > 0$.

We now gradually exchange jobs of $\mathcal{J}_k$ and $\mathcal{J}_{\overline{k}}$ between i and i' without changing the total load on either of the machines. By (2.6), there must exist a pair $j, j' \in \mathcal{J}_k$ with $x_{i,j} > 0$, $x_{i',j'} > 0$, and $p_j > p_{j'}$. Conversely, by (2.7), there are $\overline{j}, \overline{j}' \in \mathcal{J}_{\overline{k}}$ with $x_{i,\overline{j}} > 0$, $x_{i',\overline{j}'} > 0$, and $p_{\overline{j}} < p_{\overline{j}'}$. We now change the solution using some $\delta, \overline{\delta} > 0$ in the following way:

$$\begin{aligned}
x_{i,j'} &\leftarrow x_{i,j'} + \delta & \qquad x_{i,\overline{j}'} &\leftarrow x_{i,\overline{j}'} + \overline{\delta} \\
x_{i,j} &\leftarrow x_{i,j} - \delta & \qquad x_{i,\overline{j}} &\leftarrow x_{i,\overline{j}} - \overline{\delta} \\
x_{i',j'} &\leftarrow x_{i',j'} - \delta & \qquad x_{i',\overline{j}'} &\leftarrow x_{i',\overline{j}'} - \overline{\delta} \\
x_{i',j} &\leftarrow x_{i',j} + \delta & \qquad x_{i',\overline{j}} &\leftarrow x_{i',\overline{j}} + \overline{\delta}
\end{aligned}$$

Clearly, for δ and $\overline{\delta}$ sufficiently small each assignment variable remains non-negative. Moreover, each job remains fully assigned and the sum of the assignment variables of jobs of $\mathcal{J}_k$ and $\mathcal{J}_{\overline{k}}$ assigned to i and i' remains the same.

By setting $\overline{\delta} = \delta(p_j - p_{j'})/(p_{\overline{j}'} - p_{\overline{j}})$ the load over each of the two machines stays the same. Furthermore, as $p_j > p_{j'}$ and $p_{\overline{j}'} > p_{\overline{j}}$ we have that $\delta, \overline{\delta} > 0$. We choose δ maximal such that all x variables remain non-negative and the inequalities (2.6) and (2.7) still hold or turn to equality. Hence, we decreased $z_{i,k}$ by $\frac{\delta(p_j - p_j')}{y_{i,k}}$ and $z_{i',\overline{k}}$ by $\frac{\delta(p_j - p_{j'})}{y_{i,\overline{k}}}$.

At the same time we increased $z_{i,\bar{k}}$ by $\frac{\delta(p_j - p_{j'})}{y_{i,\bar{k}}}$ and $z_{i',k}$ by $\frac{\delta(p_j - p'_j)}{y_{i',k}}$. Since $z_{i',k}$ and $z_{i,\bar{k}}$ (the respective smaller z-variables for i and i' that we change) increase by at least $\frac{\delta(p_j - p'_j)}{n}$ and $z_{i,k}$ and $z_{i',\bar{k}}$ decrease by at most $\delta(p_j - p'_j)$, we have that (2.5) increases. This gives a contradiction.

Iteratively repeating this argument with the assumption that machines $[i_0]$ are correctly sorted for some $i_0 \in \mathcal{M}$, we have that there exists a solution (x, y) with vector $\{z_{i,k}\}_{i \in \mathcal{M}, k \in [1/\epsilon]}$ such that Condition (2.4) holds. □

This structural property of the slot-MILP allows us to derive a result for the weakened variant, the slot-MILP′, in which we allow machine loads to violate the bounds by at most $\delta p_{\max}$. Specifically, we show that if there exist vectors $\{y_{i,k}, z_{i,k}\}_{i \in \mathcal{M}, k \in [1/\epsilon]}$ such that Conditions (2.2) and (2.4) of Lemma 2.10 are satisfied, then there exists a solution to slot-MILP′ and this solution can be constructed in polynomial time.

Lemma 2.11. *Suppose that we are given slot and average load vectors vectors $\{y_{i,k}, z_{i,k}\}_{i \in \mathcal{M}, k \in [1/\epsilon]}$ such that conditions* (2.2) *and* (2.4) *hold. Moreover, assume that for each $i \in \mathcal{M}$ it holds that*

$$\ell \leq \sum_{k=1}^{1/\epsilon} y_{i,k} z_{i,k} \leq u + \delta p_{\max} \tag{2.8}$$

and for each $k \in [1/\epsilon]$ we have that the following holds

$$\sum_{j \in \mathcal{J}_k} p_j \leq \sum_{i \in \mathcal{M}} y_{i,k} z_{i,k} \leq \sum_{j \in \mathcal{J}_k} p_j + \delta\epsilon \cdot p_{\max}. \tag{2.9}$$

Then we can compute a vector $\{x_{i,j}\}_{i \in \mathcal{M}, j \in \mathcal{J}}$ in time $O(mn^2)$ such that (x, y) is a solution to slot-MILP′.

Proof. Let the slot and average size vectors $\{y_{i,k}, z_{i,k}\}_{i \in \mathcal{M}, k \in [1/\epsilon]}$ be given. We now construct a solution (x, y) to slot-MILP′.

First, we construct an assignment vector which satisfies the following for all $i \in \mathcal{M}$ and $k \in [1/\epsilon]$:

$$\sum_{j \in \mathcal{J}_k} p_j x_{i,j} \le y_{i,k} z_{i,k} \le u + \delta p_{\max}. \tag{2.10}$$

To this end, we make use of Condition (2.2) of Lemma 2.10. Based on this property we iteratively construct the assignment vectors for each job type k independently. First, we completely assign the smallest jobs $\mathcal{J}_k^{\min}(y_{1,k})$ to machine 1, then $\mathcal{J}_k^{\min}(y_{1,k} + y_{2,k}) \setminus \mathcal{J}_k^{\min}(y_{1,k})$ and so on. Since this assignment does not necessarily satisfy the desired property (2.10), we need to repair the property for $i = 2, \dots, m$ since machine 1 clearly satisfies (2.10) because of (2.2).

Let $i \in \{2, \dots, m-1\}$ such that all machines $1, \dots, i$ satisfy (2.10). We may assume that, when repairing machine i, machines $1, \dots, i$ contain only $\mathcal{J}_k^{\min}(y_{1,k} + \cdots + y_{i,k})$ since we do not consider jobs currently assigned to machines $i+1, \dots, m$. If machine i satisfies (2.10)) we continue with $i+1$. If this is not the case, we know that there exists a job $j \in \mathcal{J}_k$ with $p_j > z_{i,k}$ and $x_{i,j} > 0$. Moreover, because of Condition (2.2) we have

$$\sum_{i'=1}^{i} \sum_{j \in \mathcal{J}_k^{\min}(y_{1,k}+\cdots+y_{i,k})} p_j x_{i',j} = \sum_{j \in \mathcal{J}_k^{\min}(y_{1,k}+\cdots+y_{i,k})} p_j \le \sum_{i'=1}^{i} y_{i',k} z_{i',k}. \tag{2.11}$$

Since i violates (2.10) there must be some $i' < i$ satisfying (2.10) with strict inequality. In particular, there is a job j' with $p_{j'} < z_{i',k} \le z_{i,k} < p_j$ and $x_{i',j'} > 0$. We now choose an $\alpha > 0$ and exchange j' and j between i and i' as follows:

$$\begin{aligned} x_{i,j'} &\leftarrow x_{i,j'} + \alpha, & x_{i',j'} &\leftarrow x_{i',j'} - \alpha, \\ x_{i,j} &\leftarrow x_{i,j'} - \alpha, & x_{i',j'} &\leftarrow x_{i',j'} + \alpha. \end{aligned}$$

Clearly, the solution remains feasible since only jobs j and j' are touched. We choose α maximal such that either i' satisfies (2.10) with equality, i satisfies (2.10), $x_{i,j} = 0$, or $x_{i',j'} = 0$. This choice of α ensures that each pair of jobs will be exchanged at most once. This procedure is repeated until i satisfies (2.10). As the procedure is repeated for all i and possibly has to check all pairs of every job type in each exchange we have a running time of $O(mn^2)$.

It remains to be shown that for all i and k the following is true

$$\sum_{j \in \mathcal{J}_k} p_j x_{i,j} \geq y_{i,k} z_{i,k} - \delta\epsilon \cdot p_{\max}. \tag{2.12}$$

Suppose towards contradiction that for some machine i' (2.12) does not hold. Then, due to (2.10) and condition (2.3), we have that

$$\sum_{j \in \mathcal{J}_k} \sum_{i \in \mathcal{M}} x_{i,j} p_j < \sum_{i \in \mathcal{M}} y_{i,k} z_{i,k} - \delta\epsilon \cdot p_{\max} < \sum_{j \in \mathcal{J}_k} p_j. \tag{2.13}$$

This, however, contradicts the fact that all jobs must be fully assigned. Hence, we know that the constructed assignment vector satisfies (2.10) and (2.12). Therefore, (x, y) is a feasible solution to slot-MILP'. $\square$

Based on Lemmas 2.10 and 2.12, we now introduce a dynamic programming algorithm to find a solution to slot-MILP'. This, in combination with the local search procedure described in Section 2.2.2, leads to faster polynomial time additive approximation schemes for the considered load balancing problems.

The main idea of the dynamic program is to compute slot and average load vectors $\{y_{i,k}, z_{i,k}\}_{i \in \mathcal{M}, k \in [1/\epsilon]}$ that satisfy the conditions due to Lemma 2.11 such that we can compute a solution to slot-MILP' in polynomial time. The key insight towards the dynamic program is that when considering machine i in the ordering, we only need to remember the number of previously assigned jobs from each set $\mathcal{J}_k$, the current left hand side of inequality (2.2) for each k and the vector $\{z_{i-1,k}\}_{k\in[1/\epsilon]}$ of the previously considered machine $i-1$. Then, at each iteration we guess the vectors $\{y_{i,k}, z_{i,k}\}_{i\in\mathcal{M},k\in[1/\epsilon]}$ such that the new solution consisting of the guess for machine i and the remembered solution for machine i' satisfies the conditions stated in Lemma 2.12. If none of the guesses satisfies these conditions the DP cell corresponding to this iteration remains empty.

More specifically, we introduce a dynamic programming table with one cell for each combination of

- a value $i \in \mathcal{M}$ indicating the number of machines that have already been considered,
- a vector $\{z_{i,k}\}_{k\in[1/\epsilon]}$ where $z_{i,k} \in \left\{0, \frac{\delta\epsilon}{n}p_{\max}, \frac{2\delta\epsilon}{n}p_{\max}, ..., p_{\max}\right\}$ for $k \in [1/\epsilon]$. The vector $z_{i,k}$ corresponds to the average loads on the currently considered machine,
- a vector $\{y_{i,k}\}_{k\in[1/\epsilon]}$ where $y_{i,k} \in [n_k]$ for each $k \in [1/\epsilon]$. The vector $y_{i,k}$ corresponds to the number of jobs on the currently considered machine and
- for each $k \in [1/\epsilon]$
  - a value $n'_k \in \{0, ..., |\mathcal{J}_k|\}$ indicating the number of slots for jobs of $\mathcal{J}_k$ that have already been assigned to machines,
  - a value S_k with $S_k \in \left\{0, \frac{\delta\epsilon}{n}p_{\max}, \frac{2\delta\epsilon}{n}p_{\max}, ..., np_{\max}\right\}$ which corresponds to the value $\sum_{i'=1}^{i} y_{i',k}z_{i',k}$.

Each cell corresponds to the subproblem of checking whether there is a solution using machines $[i]$ such that machine i is assigned $y_{i,k}$ jobs of each type k with average load $z_{i,k}$ and for each type k a total number of n'_k slots is assigned of a total volume of S_k. If such a solution exists the corresponding cell is a true-cell. Otherwise, it is a false-cell. The dimension of the dynamic program is given by $m^2 \cdot (\frac{n}{\delta\epsilon})^{O(1/\epsilon)}$.

To recursively solve the dynamic program we proceed as follows. When considering cell

$$\left(i, \{z_{i,k}\}_{k\in[1/\epsilon]}, \{y_{i,k}\}_{k\in[1/\epsilon]}, \left\{n'_k\right\}_{k\in[1/\epsilon]}, \{S_k\}_{k\in[1/\epsilon]}\right)$$

the dynamic program checks whether there exist vectors $\{\tilde{z}_{i-1,k}\}_{k\in[1/\epsilon]}$ and $\{\tilde{y}_{i-1,k}\}_{k\in[1/\epsilon]}$ such that the entry of the dynamic program is true in cell

$$\left(i-1, \{\tilde{z}_{i-1,k}\}_{k\in[1/\epsilon]}, \{\tilde{y}_{i-1,k}\}_{k\in[1/\epsilon]}, \left\{\tilde{n}'_k\right\}_{k\in[1/\epsilon]}, \left\{\tilde{S}_k\right\}_{k\in[1/\epsilon]}\right),$$

where $\tilde{n}'_k = n'_k - y_{i,k}$ for each $k \in [1/\epsilon]$, and $\tilde{S}_k = S_k - y_{i,k}z_{i,k}$ for each $k \in [1/\epsilon]$. If this is the case, we must verify if for every $k \in [1/\epsilon]$ the currently total assigned load satisfies the right-hand side of Condition (2.9) and the average slot vector is non-decreasing in the machines:

$$S_k \leq \delta\epsilon \cdot p_{\max} + \sum_{j\in\mathcal{J}_k} p_j \tag{2.14}$$

$$z_{i,k} \geq \tilde{z}_{i-1,k}. \tag{2.15}$$

If these conditions are true, then there exists a solution for the cell

$$\left(i, \{z_{i,k}\}_{k\in[1/\epsilon]}, \{y_{i,k}\}_{k\in[1/\epsilon]}, \left\{n'_k\right\}_{k\in[1/\epsilon]}, \{S_k\}_{k\in[1/\epsilon]}\right)$$

such that the order of the average loads as well as the upper bound of the total assigned load is satisfied. To fill each individual cell in this procedure takes $(\frac{n}{\delta\epsilon})^{O(1/\epsilon)}$.

Finally, we consider the base case. For each possible combination of vectors $\{z_{m,k}\}_{k\in[1/\epsilon]}$ $\{y_{m,k}\}_{k\in[1/\epsilon]}$ we check whether there exists a solution for the dynamic programming cell

$$\left(m, \{z_{m,k}\}_{k\in[1/\epsilon]}, \{y_{m,k}\}_{k\in[1/\epsilon]}, \left\{n'_k\right\}_{k\in[1/\epsilon]}, \{S_k\}_{k\in[1/\epsilon]}\right),$$

where all jobs are assigned and for every $k \in [1/\epsilon]$ we have that

$$\sum_{j \in \mathcal{J}_k} p_j \leq S_k \leq \sum_{j \in \mathcal{J}_k} p_j + \delta\epsilon \cdot p_{\max}.$$

If such a solution exists we can use standard backward recursion to find the corresponding slot and average size vectors for all machines. Given this solution, we know by Lemma 2.11 that there is a solution to the slot-MILP′ and the corresponding assignment variables can be found in polynomial time. If there is no such solution we assert that there is no solution to the original relaxation, i.e., to slot-MILP.

Lemma 2.12. *For each $\delta > 0$ there is an algorithm with a running time of $m^2(\frac{n}{\delta\epsilon})^{O(1/\epsilon)}$ which either finds a δ-approximate solution to the slot-MILP (and thus a feasible solution to slot-MILP′) or asserts that the slot-MILP is infeasible.*

Proof. The running time follows from the dimension of the dynamic program and the time it takes to validate each specific cell. This amounts to a total running time of $m^2(\frac{n}{\delta\epsilon})^{O(1/\epsilon)}$.

Regarding the correctness of the dynamic program two observations are crucial. Firstly, due to conditions (2.14) and (2.15) and the way we check whether a solution corresponding to a DP cell exists we have that there exists a solution for machine i if and only if there is a solution for machine $i-1$. Hence, a solution can be found via backward recursion. Secondly, if for some $\{z_{m,k}\}_{k\in[1/\epsilon]}$ and $\{y_{m,k}\}_{k\in[1/\epsilon]}$ there is a solution for the cell

$$\left(m, \{z_{m,k}\}_{k\in[1/\epsilon]}, \{y_{m,k}\}_{k\in[1/\epsilon]}, \{n'_k\}_{k\in[1/\epsilon]}, \{S_k\}_{k\in[1/\epsilon]}\right),$$

Lemma 2.11 implies that there exists a feasible solution to the slot-MILP′. If no such solution exists, this implies that there is also no solution satisfying Lemma 2.10. Thus, there is no solution to the slot-MILP. □

In combination with Lemma 2.6 and choosing $\delta := \epsilon$, this dynamic program yields a significant improvement on the running time of the add-PTASes.

Theorem 2.13. *There is an algorithm for the target load balancing problem with a running time of $m^2 n^{O(1/\epsilon)}$ that computes a solution in which the load of each machine $i \in \mathcal{M}$ is in $[\ell - \epsilon \cdot p_{\max}, u + \epsilon \cdot p_{\max}]$, or asserts that there is no feasible solution.*

Corollary 2.14. *There exist additive polynomial time approximation schemes (add-PTASes) for makespan minimization, the max-min load problem and the envy-minimizing Santa Claus problem with a running time of $f(\epsilon) \cdot m^2 n^{O(1/\epsilon)}$.*

2.3 Improvement for constant number of machines

In this section, we consider the three load balancing objectives in the setting where the number of machines is a constant rather than part of the input. Here, we present additive fully polynomial time approximation schemes (add-FPTASes). These approximation schemes are based on an adjustment of the techniques introduced by Sahni [140] and Woeginger [171].

For completeness, we first present the dynamic program introduced by Sahni [140]. The idea of this dynamic program is to enumerate over all possible load vectors of the form $(L_1, L_2, ..., L_m)$. Hereto, the dynamic program iteratively constructs the set S_j including all unique load vectors possible when considering a subset of jobs $[j]$ for $j \in \mathcal{J}$. In a final step, the optimal solution can be found in S_n for each of the problems by finding the load vector optimizing the corresponding objective function. The running time of the dynamic program can be bounded by $O(nP^m)$, where $P = \sum_{j \in \mathcal{J}} p_j$.

Algorithm 2.1 Dynamic Program for load balancing on identical parallel machines for constant m.

```
1: Initialization: $S_0 = \{(0, ..., 0)\}$;
2: for $j = 1, ..., n$ do
3:     $S_j = \emptyset$
4:     for $(L_1, ..., L_m) \in S_{j-1}$ do
5:         for $\ell = 1, ..., m$ do
6:             $S_j = S_j \cup \{(L_1, ..., L_\ell + p_i, ..., L_m)\}$
7:         end for
8:     end for
9: end for
```

Following similar ideas as in [140, 171], we introduce a truncated dynamic program based on interval partitioning (see Algorithm 2.2).

Algorithm 2.2 Truncated dynamic program for load balancing on identical parallel machines with constant m.

```
1: Initialization:
2: $\hat{S}_0 = \{(0, \cdots, 0)\}$.
3: $I_0 = \{0\}, I_\ell = \big((\ell - 1)\frac{P\epsilon}{n^2}, \ell\frac{P\epsilon}{n^2}\big]$ for $1 \leq \ell \leq \lceil \frac{n^2}{\epsilon} \rceil$.
4: for $j = 1, \ldots, n$ do
5:     $\hat{S}_j = \emptyset$.
6:     for $(L_1, \ldots, L_m) \in \hat{S}_{j-1}$ do
7:         for $\ell = 1, \cdots, m$ do
8:             Let $k_1, \ldots, k_\ell^p, \ldots, k_m$ be such that
9: $(L_1, \ldots, L_\ell + p_j, \ldots, L_m) \in I_{k_1} \times \cdots \times I_{k_\ell^p} \times \cdots \times I_{k_m}$.
10:            if $\hat{S}_j \cap I_{k_1} \times \cdots \times I_{k_\ell^p} \times \cdots \times I_{k_m} = \emptyset$ then
11:                $\hat{S}_j = \hat{S}_j \cup \{L_1, \cdots, L_\ell + p_j, \cdots, L_m\}$.
12:            end if
13:        end for
14:    end for
15: end for
```

Intuitively, the main idea is to decrease the number of unique load vectors that are considered while ensuring that for every possible load vector a *representative* with similar individual loads is found. Hereto, we partition the set of possible values for each L_i into a polynomial number of sub-intervals (line 3 of Algorithm 2.2). Then, for each combination of sub-intervals we only store one representative of the possible load vectors (line 9 of Algorithm 2.2).

We show that the truncated dynamic program is indeed the basis of a fully polynomial time additive approximation scheme. To this end, we first prove that for any possible load vector found by the dynamic program, the truncated dynamic program includes a load vector such that on each machine the load deviates by at most $\epsilon p_{\max}$. The proof follows along the same lines as the results in [140, 171]. We still include it for completeness.

Lemma 2.15. *Let $\mathcal{I}$ be an instance of a load balancing problem on parallel identical machines with jobs $\mathcal{J}$. Consider the sets of load vectors $S_0, \ldots S_m$ and $\hat{S}_0, \ldots, \hat{S}_m$ computed by the dynamic program and the truncated dynamic program, respectively. Then the following holds: For every $j = 0, \ldots, n$ and $(x_1, \ldots, x_m) \in S_j$ there exists $(\hat{x}_1, ..., \hat{x}_m) \in \hat{S}_j$ such that*

$$x_h - j\frac{P\epsilon}{n^2} \leq \hat{x}_h \leq x_h + j\frac{P\epsilon}{n^2} \quad \forall h \in \mathcal{M}.$$

Proof. We prove the claim by induction on j. Clearly, for $j = 0$ we have that $S_0 = \hat{S}_0$ and the claim is true. Suppose the claim holds for some j and let $(x_1, \ldots, x_m) \in S_{j+1}$. Then, we must distinguish two cases:

- Case 1: If $(x_1, \ldots, x_m) \in S_{j+1} \cap S_j$, we have that there exists $(\hat{x}_1, ..., \hat{x}_m)) \in \hat{S}_j$ such that the statement holds and $(\hat{x}_1, ..., \hat{x}_m) \in \hat{S}_j \cap \hat{S}_{j+1}$. So the claim holds for $j + 1$ as well.
- Case 2: If $(x_1, \ldots, x_m) \in S_{j+1} \backslash S_j$, then we know that $\exists(y_1, ..., y_m) \in S_j$ such that for some $\ell \in \{1, ..., m\}$ we have that $y_\ell = x_\ell - p_{j+1}$ and $y_h = x_h$ for $h \neq \ell$.

We now show that the claim is true for $\ell = 1$; for all other ℓ the same argument holds. By assumption, the claim is true for i and, thus, there exists $(\hat{y}_1, ..., \hat{y}_m) \in \hat{S}_i$ such that $y_h - i\frac{P\epsilon}{n^2} \leq \hat{y}_h \leq y_h + i\frac{P\epsilon}{n^2}$, for $h = 1, \ldots, m$. If, in the $(j+1)$-th iteration, Algorithm 2.2 adds $(\hat{y}_1 + p_{j+1}, ..., \hat{y}_m)$ to $\hat{S}_{j+1}$, then we know that the following inequalities need to be true:

$$\hat{y}_1 + p_{j+1} \leq y_1 + j\frac{P\epsilon}{n^2} + p_{j+1} = x_1 + (j+1)\frac{P\epsilon}{n^2},$$

$$\hat{y}_1 + p_{j+1} \geq y_1 - j\frac{P\epsilon}{n^2} + p_{j+1} = x_1 - (j+1)\frac{P\epsilon}{n^2}.$$

Furthermore, for $h = 2, \ldots, m$, we have that $y_h = x_h$ and thus it follows that $x_h - (j+1)\frac{P\epsilon}{n^2} \leq \hat{y}_h \leq x_h + (j+1)\frac{P\epsilon}{n^2}$.

If $(\hat{y}_1 + p_{j+1}, ..., \hat{y}_m)$ is not added to $\hat{S}_{j+1}$, then there already exists a representative of the intervals corresponding to this solution in $\hat{S}_{j+1}$. Hence, there exists $(z_1, \ldots, z_m) \in \hat{S}_{j+1}$ such that $\hat{y}_1 + p_{j+1} - \frac{P\epsilon}{n^2} \leq z_1 \leq \hat{y}_1 + p_{j+1} + \frac{P\epsilon}{n^2}$ and for $h = 2, \ldots, m$ we have $\hat{y}_h - \frac{P\epsilon}{n^2} \leq z_h \leq \hat{y}_h + \frac{P\epsilon}{n^2}$. In this case we have

$$z_1 \leq \hat{y}_1 + \frac{P\epsilon}{n^2} + p_{j+1} \leq y_1 + (j+1)\frac{P\epsilon}{n^2} + p_{j+1} \leq x_1 + (j+1)\frac{P\epsilon}{n^2},$$

$$z_1 \geq \hat{y}_1 - \frac{P\epsilon}{n^2} + p_{j+1} \geq y_1 - (j+1)\frac{P\epsilon}{n^2} + p_{j+1} \geq x_1 - (j+1)\frac{P\epsilon}{n^2}.$$

For all $h \in \{2, ..., m\}$ we have:

$$z_h \leq \hat{y}_h + \frac{P\epsilon}{n^2} \leq x_h + (j+1)\frac{P\epsilon}{n^2}$$

$$z_h \geq \hat{y}_h - \frac{P\epsilon}{n^2} \geq x_h - (j+1)\frac{P\epsilon}{n^2}$$

Following the same argument for $\ell \in \{2, ..., m\}$, the claim holds. $\square$

Lemma 2.15 together with the fact that for each $\hat{S}_j$ we have that $|\hat{S}_j|\leq \frac{n^{2m}}{\epsilon^m}$ and the truncated dynamic program taking n iterations, we obtain a fully polynomial time additive approximation scheme. Note that, once Algorithm 2.2 has been executed, the best solution can be found in time $O(n^{2m}/\epsilon^m)$ for any of the considered load balancing problems by checking set $\hat{S}_n$ with respect to the corresponding objective function.

Theorem 2.16. *There exists a fully polynomial time additive approximation scheme for makespan minimization, the max-min load problem and the envy-minimizing Santa Claus problem with a running time of $O(\frac{n^{2m+1}}{\epsilon^m})$.*

Remark 2.17. The dynamic program can be truncated for any $\rho(I)$ which is polynomial in the number of jobs and machines such that the load on each machine varies by at most $\epsilon\rho(I)$. This implies an add-FPTAS with respect to any such $\rho(I)$.

Appendix

2.A Details of second stage of local search procedure: Repairing underloaded machines

In the following, we give a complete description and analysis of the second stage of the local search. The second stage starts with the current solution after finishing the first stage and performs an analogous procedure to repair underloaded machines, i.e., all machines whose load is currently lower than $\ell - \epsilon p_{\max}$.

This is done in an iterative manner by increasing the load of these machines. For this purpose, we define $\mathcal{M}_1$ to be the set of all underloaded machines $i \in \mathcal{M}$. We define the *direct neighbors* of a machine $i \in \mathcal{M}_1$ as all $i' \in \mathcal{M} \setminus \mathcal{M}_1$ such that for some job type $k \in [1/\epsilon]$ there exists a pair of jobs j, j' with $p_j < p_{j'}$ such that j is assigned to machine i and j' is assigned to machine i'. This pair is referred to as a pair of swappable jobs. Let $\mathcal{M}_2$ be the set of all direct neighbors of machines in $\mathcal{M}_1$. Using the same idea we construct the set $\mathcal{M}_3$ of all direct neighbours of $\mathcal{M}_2$ and repeat this procedure until we find the smallest h such that there exists a machine in $\mathcal{M}_{h-1}$ with a direct neighbor in $\mathcal{M}_h$ whose load is at least ℓ. Given this pair of machines we execute the swap of the corresponding swappable pair of jobs. Similarly to the first stage, the distance between an underloaded machine and a machine in $\mathcal{M}_h$ can be seen as $h - 1$. Intuitively, we either immediately increase the load of an underloaded machine or we try to find the closest machine with a load at least ℓ and use it to increase the load of a machine that is one set closer to an underloaded machine. Then, eventually, we can execute a swap between an underloaded machine and one of its direct neighbors. Once a swap is executed, we repeat the procedure by starting over with $\mathcal{M}_1$. Once there are no more underloaded machines, the local search procedure terminates. Since we only increase the load of machines with load at most ℓ no machines become overloaded. The second stage algorithm terminates once it has found an assignment in which no machine is underloaded anymore. Thus, in the final assignment all machine loads are within $[\ell - \epsilon \cdot p_{\max}, u + \epsilon \cdot p_{\max}]$.

Similarly to the first stage of the local search, we can show that there always exists a pair of swappable jobs as long as there is at least one underloaded machine in the second stage.

Lemma 2.18. *If there is at least one underloaded machine, there exists a pair of swappable jobs j, j' and finding such a pair can be done in time $O(n^2)$.*

Proof. Suppose towards contradiction that we can not find a swappable pair of jobs in some iteration. Then, for all job types k there exists no pair $j, j' \in \mathcal{J}_k$ such that j is assigned to a machine $i \in \mathcal{M}_\ell$ with load less than $\ell - \epsilon \cdot p_{\max}$ and job j' is assigned to a machine $i' \notin \mathcal{M}_1 \cup \ldots \cup \mathcal{M}_\ell$ and $p'_j > p_j$. If this is the case, the machines in $\mathcal{M}_1 \cup \ldots \cup \mathcal{M}_\ell$ are assigned the largest jobs of every job type k while each having a load strictly less than ℓ. Thus, the average load of these machines is strictly less than ℓ implying that not even a fractional assignment of all jobs can guarantee that the load of all these machines is at least ℓ. This gives a contradiction to the feasibility of the initial solution to the slot-MILP.

To find a pair of jobs to swap we need to consider each pair of jobs needs to be considered at most once. Hence, in the worst case we have a running time of $O(n^2)$ for each iteration. □

To finalize the proof of the running time of the second stage of the local search, we again provide an alternative formulation as a BFS algorithm. We can define the graph G in a similar fashion. The key difference is the nature of the arcs. We add an arc (v, w) for to slot vertices $v, w \in V$ if: (1) v and w are not associated with the same machine, (2) v and w are associated with the same job type and (3) v is currently assigned a smaller job than w. Furthermore, we introduce arcs (s, v) of weight 0 for all $v \in V$ being associated with an underloaded machine. The remaining graph remains the same as for the first stage.

In each iteration the algorithm starts at the source vertex and searches for a path towards a vertex v on a machine with load at least ℓ to perform a swap of the jobs associated with the last two vertices along the path.

Lemma 2.19. *In any iteration of the second stage of the local search procedure, the distance from s to any other slot does not decrease by a swap.*

Proof. Let $d(w)$ denote the distance from s to slot w before a swap. Clearly, removing arcs does not decrease the distance from s to any vertex. Arcs between slots of the same machine do not change and we do not add new arcs from the source to a machine since we only decrease the load on machines with load at least ℓ and, hence, do not decrease any machine load below $\ell - \epsilon \cdot p_{\max}$. So we only have to consider the changes in arcs of weight 1. Such an arc (u, v) might be added to the graph but only when $d(v) \leq d(u) + 1$. Adding this arc will not decrease any distances, since the first part of a shortest path using (u, v) can always be replaced by a path to v not using (u, v). We have to check that these are the only changes made to the graph.

Let u and v be the slots in which we exchange the jobs. This implies that the size of the job assigned to slot u increases and the size of the job assigned to slot v decreases. Next we look at the incoming arcs of u and outgoing arcs of v (all others remain the same). As the size of the job assigned to slot u increases there could be a new incoming arc from slots with smaller jobs. Let (w, u) be such an arc. Then either slot w is on the same machine as v or (w, v) was an arc with weight 1 before the swap. The former case implies that $d(w) = d(v) = d(u) + 1$ and in the latter case $d(u) = d(v) - 1 \leq d(w)$. In both cases we have that $d(u) \leq d(w) + 1$ which satisfies the property above. Next consider the new outgoing arcs of v. Let (v, w) be such an arc. Then the size of the job assigned to w is larger than the job assigned to v which was originally assigned to u. So, either u and w are on the same machine and $d(w) = d(u) = d(v) - 1$ or (u, w) was an arc of weight 1 in the original graph and $d(w) \leq d(u) + 1 = d(v)$. □

Lemma 2.20. *The second stage of the local search terminates after at most $O(n^3)$ swaps.*

Proof. Let $j_1, \dots, j_n$ be the jobs in $\mathcal{J}$ in decreasing order of size. We claim that the potential

$$\sum_{i=1}^{n} i \cdot d(j_i)$$

increases with every swap. Let j_k and j_h be the jobs that were swapped and assume that $k < h$, i.e. $p_{j_k} > p_{j_h}$. This implies that the arc from the slot of j_h to j_k was deleted and new arcs were constructed as described above. Let d and d' be the distances before and after the swaps, respectively. Based on Lemma 2.19 we have that $d'(j_i) \geq d(j_i)$ for all $i \notin \{h, k\}$, $d'(j_h) \geq d(j_k)$ and $d'(j_k) \geq d(j_h)$. Furthermore, we know $d(j_k) > d(j_h)$. From this it follows that

$$\begin{aligned} kd'(j_k) + hd'(j_h) &\geq kd(j_h) + hd(j_k) \\ &= kd(j_h) + (h-k) \cdot d(j_k) + kd(j_k) \\ &> kd(j_h) + (h-k) \cdot d(j_h) + kd(j_k) \\ &= hd(j_h) + kd(j_k) \end{aligned}$$

This concludes the proof. □

Chapter 3

Makespan minimization on identical parallel machines with machine conflicts

3.1 Introduction

In practice, scheduling problems on parallel machines appear with various additional restrictions on the machine environment. One such restriction plays a role whenever jobs need to be pre- and/or post-processed on a common server before and after being processed on one of the parallel machines. This arises in manufacturing and or logistical processes in which machines are served by a single robot or automated guided vehicle which is responsible for loading and unloading machines and can only serve one machine at a time [72, 93]. Another example where machines are served by a common server is where jobs must be moved into and out furnaces before and after heating processes [172].

This chapter is adapted from: M. Buchem, L. Kleist, and D. Schmidt genannt Waldschmidt. "Scheduling with Machine Conflicts". In: *CoRR* abs/2102.08231 (2021). The joint work has been accepted for publication in the *Proceedings of International Workshop on Approximation and Online Algorithms 2022 (WAOA 2022)*. The results will also appear in the doctoral thesis of Daniel Schmidt genannt Waldschmidt.

In this chapter, we consider a natural generalization of the parallel machine scheduling with a single common server introduced by Hall et al. [72] and Kravchenko and Werner [103] by relaxing the assumption that only one one of the machines can access the server at a time. We do so by extending this problem with the notion of *machine conflicts*. We say that a pair of machines cannot be served simultaneously if the two machines are in conflict with each other. Conflicts of this type occur in environments with spatial limitations or resource restrictions regarding the server. In the following, we give a formal definition. We then proceed to give an overview of related work and the contributions of this chapter.

Problem Definition. An instance of SCHEDULINGWITHMACHINECONFLICTS (SMC) is defined by a set of jobs $\mathcal{J}$ (with $n := |\mathcal{J}|$) and a set of machines $\mathcal{M}$ (with $m := |\mathcal{M}|$) with a conflict graph $G = (\mathcal{M}, E)$ where a pair of machines $i, i' \in \mathcal{M}$ is said to be *in conflict* with each other if and only if $\{i, i'\} \in E$. Each job is associated with three processing components: a pre-processing time $\overleftarrow{b_j}$, a processing time p_j and a post-processing time $\overrightarrow{b_j}$. The pre- and post-processing of jobs is done externally on a single server and the processing of jobs is done on one of the parallel machines. We also refer to the pre- and post-processing time of a job as its first and second *blocking time*. The total duration of a job is referred to as its *system time* and defined as $q_j = \overleftarrow{b_j} + p_j + \overrightarrow{b_j}$. The order of the three job components must be maintained and jobs may not be interrupted at any point, i.e., a feasible schedule must be non-preemptive. Furthermore, a schedule Π must be *conflict free*:

- For each point in time, every machine processes at most one job,
- for every edge $\{i, i'\} \in E$ and two jobs $j, j' \in \mathcal{J}$ assigned to machines i and i', respectively, the intervals of the blocking times of j and j' do not overlap interiorly in time. In other words, for every edge $\{i, i'\} \in E$ the server pre- or post-processes at most one job assigned to one of the two machines at the same time.

Given a schedule Π the completion time of a job is defined as the time at which the server has finished post-processing this job. The goal is to find a conflict free schedule of minimum makespan, i.e., the maximum completion time of all jobs. We denote the optimal makespan by OPT. SMC is a generalization of the classical problem of makespan minimization on parallel machines in two ways: the introduction of pre- and post-processing and the introduction of a conflict graph. More specifically, makespan minimization on identical parallel machines is a special case of SMC where either $\overleftarrow{b_j} = \overrightarrow{b_j} = 0$ for all jobs or $E = \emptyset$.

Related Work. *Parallel machine scheduling with a common server* was introduced by Hall et al. [72] and Kravchenko and Werner [103]. In its original setting, jobs must be processed non-preemptively with a single server being responsible for pre-processing of jobs. Moreover, the server can only process one job at a time. Specifically, this problem is equivalent to SMC where G is a complete graph and $\overrightarrow{b_j} = 0$ for all jobs j. Since this problem is a generalization of the classical makespan minimization problem on identical parallel machines, it is known to be weakly $\mathcal{NP}$-hard for a fixed number of machines and strongly $\mathcal{NP}$-hard for an arbitrary number of machines [59].

To investigate the computational complexity of the parallel machine scheduling problem with a common server, multiple special cases have been considered. Hall et al. [72] take a closer look at the special case of unit serving times ($\overleftarrow{b_j} = 1\,\forall j$) on two identical parallel machines and show that it is weakly $\mathcal{NP}$-hard. Furthermore, they also show that the special case with identical serving times ($\overleftarrow{b_j} = \overleftarrow{b}\,\forall j$) is strongly $\mathcal{NP}$-hard even on two identical parallel machines [72]. Kravchenko and Werner [103] prove that the case with unit serving times is strongly $\mathcal{NP}$-hard when the number of machines is arbitrary. Brucker et al. [21] show that the special case with identical processing times ($p_j = p\,\forall j$) is weakly $\mathcal{NP}$-hard even for a fixed number of machines.

From an algorithmic point-of-view multiple special cases have been considered as well. Kravchenko and Werner [103] introduce a pseudo-polynomial time algorithm for the case of $\overleftarrow{b_j} = 1\,\forall j$ and $m = 2$. Abdekhodaee and Wirth [1] develop an exact polynomial time algorithm for the special case where for any pair of jobs j and j' we have $p_j \leq s_{j'}$. Furthermore, the special case with unit processing times, i.e., $p_j = 1 \, \forall j$, can be solved in linear time [72]. Xie et al. [172] extend this model to a single server responsible for both pre- and post-processing of jobs. This problem is equivalent to SMC with K_m as the conflict graph. They show that if preemption is allowed between the processing and post-processing of jobs the longest processing time rule introduced by Graham [62] yields an approximation guarantee of $3/2 - 1/2m$. Jiang et al. [93] consider this problem on two machines in the fully non-preemptive setting and derive approximation guarantees of $12/7$ and $4/3$ for list scheduling and the longest processing time rule, respectively.

Other scheduling models which take into account external pre- and/or post-processing of jobs are the *master-slave scheduling problem* introduced by Kern and Nawijn [99] and termed by Sahni [141] and *scheduling with segmented self-suspension* with a single self-suspension segment introduced by Rajkumar et al. [136]. In the master-slave scheduling problem a job consists of three components: two processing segments which must be executed in the right order on a master machine and one intermediate segment for which a sufficient amount of slave machines is available. The aim is to minimize the makespan. In scheduling with segmented self-suspension a job consists of two computation segments and a suspension interval. The second computation segment of a job can only start after the length of the suspension interval has passed since the completion of the first computation segment. An overview of the relation between the two problems and approximation algorithms for special cases were shown by Chen et al. [30].

Machine conflicts were considered by Chrobak et al. [35] for the online problem of buffer minimization in a multiprocessor environment. Here, conflicting processors are not allowed to process jobs at the same time. The problem is shown not to admit any non-trivial approximation guarantees for general conflict graphs in the offline setting [35]. In the online setting, approximation algorithms for various special graph classes have been developed [35, 81].

The notion of conflicts in parallel machine scheduling has been considered in terms of *job conflicts* with two types of restrictions:

- conflicting jobs are not allowed to be scheduled on the same machine (JC-INTRA), or
- conflicting jobs are not allowed to be scheduled concurrently on different machines (JC-INTER).

Bodlaender and Jansen [17] show that JC-INTRA is $\mathcal{NP}$-hard even for unit time jobs on bipartite conflict graphs. For bipartite graphs and graphs with bounded treewidth, Bodlaender et al. [19] develop approximation algorithms. Das and Wiese [39] consider JC-INTRA where the conflict graph is a collection of cliques and introduce a PTAS.

Different types of problems fall under the classification of JC-INTER: mutual exclusion scheduling problem (MES), scheduling with conflicts and scheduling with agreements where an agreement graph is the complement of a conflict graph. In mutual exclusion scheduling (MES), unit processing jobs have to be scheduled on identical parallel machines such that two conflicting jobs may not be scheduled concurrently. Baker and Coffman [11] showed that MES is $\mathcal{NP}$-hard for general graphs and even for $m = 3$ with unit processing time jobs. The computational complexity of MES has been further investigated for various graph classes, see, e.g, [11, 16, 18, 168, 57, 74]. In the case of bipartite graphs, for example, MES remains $\mathcal{NP}$-hard [18], while it becomes polynomial time solvable for a fixed number of machines [11, 18, 74]. Even et al. [54] consider JC-INTER for non unit jobs on two machines with few job types. They develop an approximation algorithm when $p_j \in \{1, 2, 3\}$ and show that the problem is APX-hard when $p_j \in \{1, 2, 3, 4\}$ even for bipartite conflict graphs [54]. For complements of bipartite graphs, Bendraouche and Boudhar [15] showed $\mathcal{NP}$-hardness with few job types. Mohabeddine and Boudhar [121] showed that JC-INTER is $\mathcal{NP}$-hard on complements of trees and solvable in polynomial time for complements of caterpillars or cycles.

Our Contributions. In this chapter, we mainly focus on the special case of SMC with unit jobs ($\overleftarrow{b_j} = p_j = \overrightarrow{b_j} = 1\,\forall j$), denoted by SMC-Unit, and its generalization with identical jobs ($\overleftarrow{b_j} = \overleftarrow{b}, p_j = p, \overrightarrow{b_j} = \overrightarrow{b}$ $\forall j$) which we denote by SMC-Id.

In Section 3.2, we introduce some preliminary notions and graph theoretical concepts used in the remainder of the chapter. In Section 3.3 we use these concepts to establish a relationship between SMC-Unit and finding a maximum induced bipartite subgraph of the conflict graph. Based on this connection, we derive an inapproximability result for SMC-Unit on general conflict graphs. Complementary, we develop exact and approximation algorithms for different special graph classes. In Section 3.4.1 we develop approximation algorithms for graph classes for which a maximum induced bipartite subgraph or an approximation of it can be found in polynomial time. In Section 3.4.2 we present exact polynomial time algorithms for various special graph families. Most prominently, we present a polynomial time algorithm when the conflict graph is bipartite. Additionally, we show that the problem can be solved in polynomial time on complete conflict graphs. In Section 3.5, we turn our attention towards the more general case of SMC-Id and generalize the approximation algorithms developed in Section 3.4.1.

3.2 Preliminary Notions

In the following, we formally define the concepts of *maximum independent sets* and *maximum induced bipartite subgraphs*. These are later used to construct schedules as well as deriving a lower bound on the optimal makespan.

A *maximum independent set* of a graph G is a subset of the vertices such that there is no edge between any two vertices in the subset. More formally, a maximum independent set is defined as follows.

Definition 3.1 ((Maximum) independent set). *Let $G = (V, E)$ be a graph. A set of vertices $I_1 \subseteq V$ is an independent set if for any pair of vertices u, v we have that $\{u, v\} \notin E$. An independent set of maximum cardinality is referred to as a maximum independent set and we denote by $\alpha_1(G)$ the size of a maximum independent set of G.*

Similarly, a *maximum induced bipartite subgraph* of G is an induced subgraph whose vertex set consists of two disjoint independent sets.

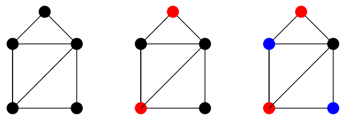
Definition 3.2 ((Maximum) induced bipartite subgraph). *Let $G = (V, E)$ be a graph. Two disjoint independent sets of vertices $I_1, I_2 \subseteq V$ form an induced bipartite subgraph of G. We say that I_1 and I_2 form a maximum induced bipartite subgraph if the cardinality of their union is maximized. We denote by $\alpha_2(G)$ the size of a maximum induced bipartite subgraph of G.*

We use the concepts of maximum independent sets and maximum induced bipartite subgraphs to define a class of subschedules which we later use for both algorithmic purposes as well as proving the inapproximability for SMC-UNIT on general conflict graphs.

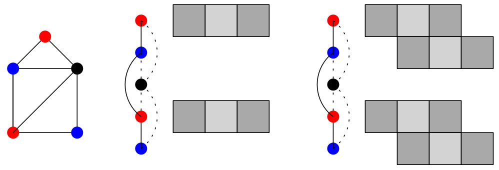
Definition 3.3 (1-Pattern). *Consider an instance $\mathcal{I}$ of* SMC-UNIT *and let $G = (V, E)$ be a conflict graph. Let I_1 be an independent set of G. A partial schedule of length 3 starting at time t is called a* 1-pattern on I_1 *if on each machine $i \in I_1$, there is one job starting at time t. If I_1 is a maximum independent set, we refer to the corresponding 1-pattern as an A-pattern.*

Definition 3.4 (2-Pattern). *Consider an instance $\mathcal{I}$ of* SMC-UNIT *and let $G = (V, E)$ be a conflict graph. Let I_1, I_2 be the vertex sets of an induced bipartite subgraph of G. A partial schedule of length 4 starting at time t is called a* 2-pattern on I_1 and I_2 *if on each machine $i \in I_1$, there is one job starting at time t and on each machine $i \in I_2$ there is one job starting at time $t + 1$. If I_1 and I_2 form a maximum induced bipartite subgraph, we refer to the corresponding 1-pattern as a B-pattern.*

A visualization of A- and B-patterns is given in Figure 3.2.1. Here, machines are represented by vertices of the conflict graph and the patterns are depicted as Gantt charts over time. In Figure 3.2.1b, the pre- and post-processing time of a job is visualized in dark gray and the processing time in light gray.



(a) Example of a conflict graph for 5 machines: maximum independent set (middle) and maximum induced bipartite subgraph (right).



(b) A-Pattern on maximum independent set I_1 (middle) and B-pattern using vertex set of maximum induced bipartite subgraph (right).

Figure 3.2.1: Example of maximum independent set and maximum induced bipartite subgraphs and corresponding 1- and 2-Patterns.

3.3 Unit Jobs: Inapproximability for general conflict graphs

In the following, we prove an inapproximability result by establishing a relation between SMC-UNIT and the problem of finding a maximum induced bipartite subgraph. More formally, we show the following result.

Theorem 3.5. *For any $\varepsilon > 0$, there exists no $\mathcal{O}(m^{1-\varepsilon})$-approximation for* SMC-UNIT *on general conflict graphs, unless $\mathcal{P} = \mathcal{NP}$.*

To prove this result, we first derive a lower bound on the optimal makespan for SMC-UNIT. To this end, we derive a lower bound on any feasible schedule by establishing a connection between a conflict free schedule for SMC-UNIT and an induced bipartite subgraph of the conflict graph. More specifically, given a contlict-free schedule Π, we extract an induced bipartite subgraph of G such that the makespan of Π is bounded from below by a function of n and the size of the extracted induced bipartite subgraph. Here, we make use that for unit jobs we can restrict ourselves to schedules with integer starting times.

Lemma 3.6. *Let Π be a feasible schedule of an instance of* SMC-UNIT *with conflict graph G. Then, we can extract an induced bipartite subgraph of G of size β_2^Π in time $O(n)$ such that the makespan of Π is bounded from below as*

$$\|\Pi\| \geq 3 \cdot \lceil n/\beta_2^\Pi \rceil .$$

Proof. Let Π be a feasible schedule of an instance of SMC-UNIT with conflict graph G. Let $t \in \mathbb{Z}$ and consider the interval $[t, t+3)$. We first show that for any such interval, the machines starting a job in this interval form an induced bipartite subgraph of G. Denote by $\mathcal{M}_i$ the machines starting a job at time $t+i$ for $i = 0, 1, 2$. Clearly, no machine can start 2 jobs in this interval and, hence, the sets $\mathcal{M}_0, \mathcal{M}_1$ and $\mathcal{M}_2$ are disjoint. Furthermore, the sets $\mathcal{M}_0$ and $\mathcal{M}_2$ form an independent set since the post-processing time of jobs on machines $\mathcal{M}_0$ fully overlap with the pre-processing time of jobs on machines $\mathcal{M}_2$. Since the machines in $\mathcal{M}_1$ form an independent set as well, we have that $\mathcal{M}_0, \mathcal{M}_1$ and $\mathcal{M}_2$ form the vertex set of an induced bipartite subgraph (see Figure 3.3.1). Using this argument for all such intervals for any $t \in \mathbb{Z}$, we define β_2^Π as the largest of the induced bipartite subgraphs we can extract over all intervals. As the makespan of any schedule is at most $3n$, this can be done in time $O(n)$.

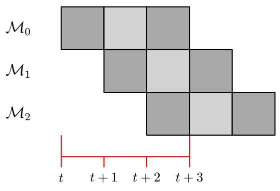


Figure 3.3.1: Visualization of upper bound of jobs starting in interval $[t, t+3)$.

It remains to be shown that we can bound the makespan of Π as a function of n and β_2^{Π}. Hereto, we divide Π into intervals of length 3 starting at time 0. By the argument above, at most β_2^{Π} jobs start in each interval. This means that the number of necessary intervals of length 3 to process all jobs is at least $\lfloor n/\beta_2^{\Pi} \rfloor$. Moreover, if β_2^{Π} does not divide n, at least one job of length 3 starts after time $3\lfloor n/\beta_2^{\Pi} \rfloor$. Therefore, $\|\Pi\| \geq 3 \cdot \lceil n/\beta_2^{\Pi} \rceil$. $\square$

As for any feasible schedule Π the size of the extracted induced bipartite subgraph is at most the size of a maximum induced bipartite subgraph in G, the following lower bound holds for the optimal makespan.

Corollary 3.7. *Let Π be a feasible schedule of an instance of* SMC-UNIT *with conflict graph G. Then, the optimal makespan for scheduling n jobs is bounded from below by*

$$\mathrm{OPT} \geq \frac{3n}{\alpha_2(G)}.$$

Using Lemma 3.6 and Corollary 3.7, we prove Theorem 3.5.

Proof of Theorem 3.5. Consider an instance of SMC-UNIT with n jobs and conflict graph G such that $\alpha_2(G) \geq n$. Suppose for the sake of contradiction that for some $\kappa > 0$ and some $\varepsilon > 0$ there exists a polynomial time algorithm ALG with multiplicative approximation guarantee $\kappa m^{1-\varepsilon}$. Then we know that the makespan of ALG is bounded from above by

$$\|\Pi\| \leq (\kappa m^{1-\varepsilon}) \cdot \mathrm{OPT}.$$

Furthermore, by Lemma 3.6 we know that

$$\|\Pi\| \geq 3\frac{n}{\beta_2^{\Pi}},$$

where β_2^{Π} is the induced bipartite subgraph extracted from Π. Moreover, constructing a schedule containing only B-patterns on a maximum induced bipartite subgraph of G gives an upper bound on the optimal makespan of

$$\mathrm{OPT} \leq 4\left\lceil \frac{n}{\alpha_2(G)} \right\rceil \leq 8\frac{n}{\alpha_2(G)},$$

where the second inequality follows from $\alpha_2(G) \geq n$. Combining these bounds yields

$$\beta_2^{\Pi} \geq \frac{3n}{\|\Pi\|} \geq \frac{3n}{\kappa m^{1-\varepsilon} \cdot \mathrm{OPT}} \geq \frac{1}{\kappa m^{1-\varepsilon}} \cdot \frac{3}{8} \cdot \alpha_2(G).$$

As β_2^{Π} can be computed in time $O(n)$, this yields a polynomial time $\mathcal{O}(m^{\varepsilon-1})$-approximation algorithm for the problem of finding a maximum induced bipartite subgraph contradicting [75, 116, 174]. $\square$

3.4 Unit Jobs: Exact and approximation algorithms for special cases

Motivated by the inapproximability result for SMC-UNIT for general conflict graphs, we closer investigate special graph classes.

In the following, we first show how to utilize B-patterns (2-patterns) to translate maximum (approximate) induced bipartite subgraphs into exact (approximation) algorithms for SMC-UNIT (Section 3.4.1). We then turn our attention towards finding exact polynomial time algorithms for specific graph classes (Section 3.4.2).

3.4.1 From (maximum) induced bipartite subgraphs to (near)-optimal schedules

Whereas the inapproximability result holds for general graph classes, we now show that if the maximum induced bipartite subgraph of a conflict graph is given, then constructing a schedule using only B-patterns on this maximum induced bipartite subgraph leads to a schedule with a constant multiplicative approximation guarantee. Thus, this implies a polynomial time approximation algorithm for any conflict graph class for which we can find a maximum induced bipartite subgraph in polynomial time.

Theorem 3.8. *Consider an instance of* SMC-UNIT *with n jobs on a conflict graph G. Let I_1 and I_2 be the vertex sets of a maximum induced bipartite subgraph of G. Then we can find a schedule Π in polynomial time such that*

$$\|\Pi\| \leq \frac{4}{3}\text{OPT}.$$

Proof. We construct Π by using only $B-$patterns on I_1 and I_2 (and possibly deleting some of the last jobs to ensure that exactly n jobs are processed). This yields a makespan of at most

$$\|\Pi\| \leq 4\left\lceil \frac{n}{\alpha_2(G)} \right\rceil.$$

Then, by Lemma 3.7 it follows that

$$\|\Pi\| \leq \frac{4}{3}3\left\lceil \frac{n}{\alpha_2(G)} \right\rceil \leq \frac{4}{3}\text{OPT}. \qquad \square$$

Similarly, we can derive a polynomial time approximation algorithm for SMC-UNIT for graph classes for which a polynomial time approximation algorithm for finding the maximum induced bipartite subgraph exists.

Theorem 3.9. *Consider an instance of* SMC-UNIT *with n jobs on a conflict graph G. Let I_1 and I_2 be the vertex sets of an approximate induced bipartite subgraph of G with size $\beta_2(G) \geq \gamma\alpha_2(G)$, where $\gamma < 1$. Then we can construct a schedule* Π *in polynomial time such that*

$$\|\Pi\| \leq \frac{8}{3\gamma}\mathrm{OPT}.$$

Proof. In order to construct Π, we use 2-patterns on I_1 and I_2. Note that in the final 2-pattern it might be necessary to delete some jobs such that the correct number of jobs is processed. The makespan of Π is then bounded from above by

$$\|\Pi\| \leq 4\left\lceil \frac{n}{\beta_2(G)} \right\rceil.$$

To obtain the approximation guarantee we must distinguish two cases. First, if $n \leq \beta_2(G)$ (and $n \leq \alpha_2(G)$), then

$$\|\Pi\| \leq \frac{4}{3} \cdot 3 \leq \frac{4}{3}\mathrm{OPT}.$$

Otherwise, if $n > \beta_2(G)$ then we have that

$$\left\lceil \frac{n}{\beta_2(G)} \right\rceil \leq 2\frac{n}{\beta_2(G)} \leq \frac{2}{\gamma}\frac{n}{\alpha_2(G)}.$$

Hence, this yields the following upper bound on the makespan of Π:

$$\|\Pi\| \leq \frac{4}{3}\frac{2}{\gamma}\frac{3n}{\alpha_2(G)} \leq \frac{8}{3\gamma}\mathrm{OPT}. \qquad \square$$

3.4.2 Unit Jobs: Exact algorithms for special graph classes

In the following, we improve upon the approximation algorithms presented in the previous subsection and develop exact polynomial time algorithms for two special graph classes: *complete graphs* and *bipartite graphs*.

SMC-UNIT with complete conflict graphs

Complete conflict graphs are of special interest since SMC with complete conflict graphs is equivalent to the problem of scheduling on identical parallel machines with a single server responsible for pre- and post-processing introduced by Xie et al. [172] and Jiang et al. [93]. We show that for SMC-UNIT on complete conflict graphs there exists an optimal schedule using only two distinct machines. Hence, SMC-UNIT on a complete conflict graph is equivalent to SMC-UNIT with a single edge as a conflict graph.

Lemma 3.10. *Consider an instance of* SMC-UNIT *with n jobs, $m \geq 2$ machines and a complete graph K_m as a conflict graph. An optimal schedule can be computed in time $O(\log n)$ and it coincides with an optimal schedule with respect to conflict graph K_2 of makespan $4\lfloor n/2 \rfloor + 3(n \mod 2)$.*

Proof. Let Π be a feasible schedule for SMC-UNIT with n jobs, $m \geq 2$ machines and a complete graph K_m as a conflict graph. We will first refine the lower bound of Lemma 3.7. To this end, consider some time $t \geq 0$ and let $\mathcal{M}_i$ be the set of machines starting a job at time $t + i$ for $i = 0, 1, 2, 3$. Then, as the conflict graph is complete we know that each of these sets is at most of size 1. Furthermore, if $\mathcal{M}_0 \neq \emptyset$, then $\mathcal{M}_2 = \emptyset$ as otherwise the post-processing time of the job starting at time 0 overlaps with the pre-processing time of the job starting at time 2. The reverse holds if $\mathcal{M}_2 \neq \emptyset$ and, therefore, $|\mathcal{M}_0 \cup \mathcal{M}_2| \leq 1$. Following a similar idea, the same can be shown for $\mathcal{M}_1$ and $\mathcal{M}_3$. Thus, in an interval of length 4 at most 2 jobs start, yielding a lower bound of

$$\text{OPT} \geq 4 \left\lfloor \frac{n}{2} \right\rfloor + 3(n \mod 2).$$

We now argue that constructing a schedule Π consisting only of B-patterns and (possibly) one final A-pattern matches this lower bound. If $n \mod 2 = 0$, we need exactly $n/2$ B-patterns to process all jobs and if $n \mod 2 = 0$ one additional job is processed on either of the two machines starting at time $4\lfloor n/2 \rfloor$. Hence, Π has a makespan of

$$4 \left\lfloor \frac{n}{2} \right\rfloor + 3(n \mod 2).$$

Thus, an optimal schedule can be found in time $O(\log n)$. □

SMC-UNIT with bipartite conflict graphs

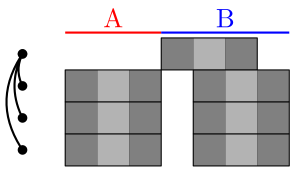
Bipartite conflict graphs are of special interest for SMC-UNIT since for any conflict graph and a feasible schedule the set of machines processing a job at the same time form (at most) an induced bipartite subgraph of the conflict graph. Therefore, optimal schedules on bipartite conflict graphs offer insights into local optimality criteria for schedules on general graph classes.

Observation 3.11. *Consider an instance* SMC-UNIT *on a graph G and a feasible schedule* Π. *At any point in time t, the set of machines that are busy processing a job form the vertex set of an induced bipartite subgraph of G.*

Observe that the approximation algorithms underlying Theorems 3.8 and 3.9 follow the idea of using (maximum) induced bipartite subgraphs as well and use a near optimal schedule on these subgraphs. Therefore, an improvement for bipartite graphs may open up possibilities for improvement on other graph classes.

In the following, we develop a polynomial time algorithm for SMC-UNIT on bipartite conflict graphs. This algorithm is based on a divide-and-conquer technique by exploiting structural properties of optimal schedules on subgraphs of the conflict graph. Therefore, we must first make a small excursion to *stars*.

A *star* is a complete bipartite graph $S_\ell := K_{1,\ell}$ on $\ell+1 \geq 2$ vertices. For $\ell \geq 2$, S_ℓ has a unique *center* and ℓ *leaves*. When $\ell = 1$, we can freely choose any of the two vertices to be the center. We next show that optimal schedules on stars consist of A- and B-patterns as defined in Section 3.2. Note that we relax the notation in the sense that we do not specify the set(s) used for the A- or B-pattern but refer to the patterns being scheduled on the star. Furthermore, we define an *AB-schedule* on a graph G as a schedule consisting of A- and B-patterns only. An example of an AB-schedule is shown in Figure 3.4.1a. Note that an AB-schedule may process more jobs than necessary which can be fixed by deleting some jobs and forming at most one incomplete pattern using not all machines necessary.



(a) An AB-schedule on S_3 consisting of one A-pattern followed by one B-pattern.

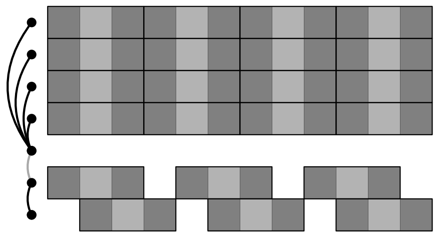

(b) Optimal schedule on a tree with 7 machines and 22 jobs with makespan 12.

Figure 3.4.1: AB-schedules.

We next show, that for any star we can restrict ourselves to AB-schedules to find an optimal schedule for SMC-UNIT.

Lemma 3.12. *For any instance of* SMC-UNIT *with* $G = S_\ell$ *for* $\ell = m - 1$, *there exists an AB-schedule that is optimal.*

Proof. For S_1 the statement follows from Lemma 3.10. Consider a star S_ℓ with $\ell \geq 2$ and let Π^* be an optimal schedule. We now construct an AB-schedule Π with the same makespan. First, let i be the leaf that is assigned the maximum number of jobs. Copying this schedule for all other leaves does not increase the makespan and yields a schedule Π where all leaves follow the same subschedule. Note that this process may increase the number of jobs scheduled which can be fixed by deleting the additional jobs. Finally, for each group of jobs scheduled on the leaves two cases remain. Either a job j being processed on the center of the star overlaps with this block of jobs or not. In the first case, we have a B-pattern. In the second case we have an A-pattern. Thus, Π is an optimal schedule and an AB-schedule. □

The existence of an optimal AB-schedule enables us to solve SMC-UNIT on stars in polynomial time.

Lemma 3.13. *For any instance of* SMC-UNIT *on a star S_ℓ and n jobs, an optimal schedule can be computed in time linear in* $\log n$ *and* $|S|$.

Proof. To prove this statement, we make a case distinction on ℓ.

Case 1. Let $\ell \leq 2$. In this case, there exists an optimal AB-schedule which uses at most 2 A-patterns since any 3 A-patterns processing 3ℓ jobs in time 9 can be replaced with 2 B-patterns scheduling $2\ell + 2 \geq 3\ell$ jobs in time 8. Therefore, an optimal AB-schedule can be found by enumerating over the three possibilities using k A-patterns with $k = 0, 1, 2$ and computing the necessary amount of B-patterns needed to process at least n jobs in total.

Case 2. Let $\ell \geq 3$. In this case, there exists an optimal AB-schedule which uses at most 2 B-patterns since any 3 B-patterns processing $3\ell+3$ jobs in time 12 can be replaced with 4 A-patterns scheduling $4\ell \geq 3\ell+3$ jobs in time 12. Therefore, an optimal AB-schedule can be found by enumerating over the three possibilities using k B-patterns with $k = 0, 1, 2$ and computing the necessary amount of A-patterns needed to process at least n jobs in total. □

While we can restrict ourselves to AB-schedules when considering stars as conflict graphs, this restriction does not hold for all bipartite graphs. In fact, we cannot generalize this statement even to trees as seen in Figure 3.4.1b. However, an interesting observation regarding the optimal schedule in Figure 3.4.1b is that it is comprised of two AB-schedules on the top star and bottom star obtained when deleting the gray edge connecting the two stars. This insight (in combination with the optimality of AB-schedules on stars) is the foundation of our divide-and-conquer algorithm.

The key idea of the next step towards our polynomial time algorithm is to derive a structural property of optimal schedules on bipartite graphs based on a spanning subgraph H of G for which optimal AB-schedules with respect to H are also feasible with respect to G and, therefore, optimal with respect to G. More specifically, we identify a class of subgraphs consisting of stars such that the AB-schedules yielding optimality on each individual star can be feasibly combined into an optimal schedule on the whole subgraph. To clearly verify this feasibility we encode it using certain vertex colorings. We now give more formal definitions of the specific type of subgraph and the vertex colorings.

Definition 3.14 (Star forest of G)**.** *Consider a conflict graph G and subgraph H. We say that H is a star forest of G if each connected component of H is a star and H contains all vertices of G.*

Based on a star forest of the conflict graph, the idea behind the algorithm is that an optimal schedule on the individual stars can be feasibly patched together such that it is also optimal on G. This is where the vertex colorings come into play. Intuitively, we want to define specific vertex subsets of H, such that specific sequences of A and B patterns on the machines corresponding to these vertices can be feasibly scheduled in parallel.

Definition 3.15 (I, II, III-colorings). *Let G be a conflict and H be a star forest of G. Then, we define* I, II, III-*colorings as follows:*

- *A vertex subset A_1 is a* I-coloring *of (G, H) if it is a maximum independent set of both G and H.*
- *Two disjoint vertex subsets A_2, B_2 form a II-coloring of (G, H), if no vertex of A_2 is adjacent to another vertex from $A_2 \cup B_2$ in G and the following properties hold: (i) for each $S = S_\ell$ with $\ell \geq 3$, A_2 contains all leaves of S, (ii) for each $S = S_1$, B_2 contains both the vertices of S and (iii) for each $S = S_2$, either A_2 contains both leaves of S or B_2 contains all vertices of S.*
- *Two disjoint vertex subsets A_3, B_3 are a* III-coloring *of (G, H), if no vertex of A_3 is adjacent to another vertex from $A_3 \cup B_3$ in G and the following properties hold: (i) for each $S = S_\ell$ with $\ell \geq 4$, A_3 contains all leaves of S, (ii) B_3 contains all vertices of each S_1 and each S_2, and (iii) for each $S = S_3$, either A_3 contains all leaves of S or B_3 contains all vertices of S.*

These vertex colorings allow us to combine certain sequences of A- and B-patterns across all components of a star forest. A I-coloring allows us to schedule one A-pattern in parallel on all stars using the set S_1. This yields a feasible schedule for G, see Figure 3.4.2 (left). Similarly, a II-pattern allows us to combine 3 A-patterns on stars whose leaves are contained in A_2 with 2 B-patterns on stars which are fully contained in B_2, see Figure 3.4.2 (middle). Finally, a III-coloring implies a feasible combination of 4 A-patterns on stars whose leaves are in A_3 and 3 B-patterns on stars whose vertex set is fully contained in B_3, see Figure 3.4.2 (right).

We next show how to use the concepts of star forests and I, II, III-colorings to extend the structural insights on stars to general bipartite graphs. We do so by showing that an optimal schedule on a star forest admitting I, II, III-colorings can be changed into a feasible schedule on the conflict graph without increasing the makespan.

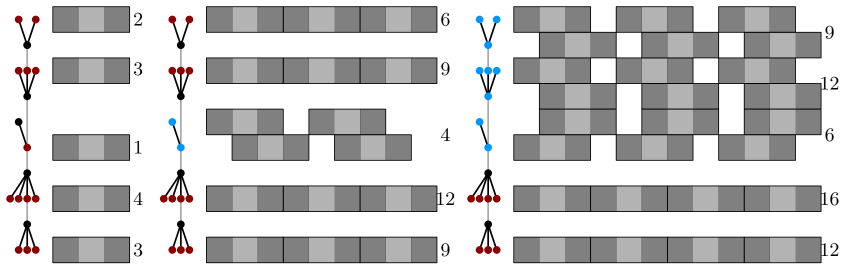


Figure 3.4.2: A graph and a star forest with a I-, II-, and III-coloring and the corresponding schedules and the number of jobs being processed on each star for each schedule. Vertices in A_i are colored in red and vertices in B_i in blue.

Lemma 3.16. *Consider an instance $\mathcal{I}$ of* SMC-UNIT *with a connected bipartite conflict graph G on at least two vertices. Let H be a star forest of G. Given a I-coloring A_1, a II-coloring (A_2, B_2) and a III-coloring (A_3, B_3) of (G, H), an optimal schedule for $\mathcal{I}$ can be computed in polynomial time.*

Proof. Let OPT be the makespan of an optimal schedule for instance $\mathcal{I}$ of SMC-UNIT on conflict graph G. By Lemma 3.12, there exists an optimal schedule for the same instance considering the star forest H as the conflict graph which consists of AB-schedules on the individual components. We denote this schedule by Π. As H is a subgraph of G, we have $\|\Pi\| \leq$ OPT. In the following, we construct a schedule Π^* which is feasible with respect to G and whose makespan is equal to the makespan of Π (and, therefore, Π^* is an optimal schedule with respect to G). We construct Π^* via a case distinction. First, we consider the case of small optimal makespan values and prove the following claim.

Claim 3.17. If $\|\Pi\| \leq 20$, then there exists an optimal AB-schedule Π^* on H that is feasible for G (according to Table 3.4.1).

To prove this claim, first observe that $\|\Pi\| \geq 3$ and that there is no AB-schedule with $\|\Pi\| = 5$. Therefore, we must consider the cases where $\|\Pi\| \in [20] \setminus \{1, 2, 5\}$. We now show that for each of these cases there exists a combination of sequences of A- and/or B-patterns across all components of H such that this yields a feasible schedule on G processing at least as many jobs in the same time. We explain this according to Table 3.4.1. Hereto, we focus on four types of stars in H: S_1, S_2, S_3, S_ℓ with $\ell \geq 4$. For each type of star and every possible makespan $|\Pi|$ (of at most 20), the key idea is to choose an AB-schedule which processes the maximum number of jobs in time $\|\Pi\|$ and is feasible to combine with the AB-schedules on the other components. To ensure feasibility, some AB-schedules are modified for S_1 and S_2 components; these are marked an asterisk in Table 3.4.1. We now describe the necessary modifications in more detail.

For S_1 components and $\|\Pi\| \equiv 2 \pmod 4$, we use a sequence of 2 A-patterns combined with $(\lfloor \|\Pi\|/4 \rfloor - 1)$ B-patterns instead of a sequence of $\lfloor \|\Pi\|/4 \rfloor$ B-patterns. As 2 A-patterns and 1 B-pattern differ in length 2, the modified schedule on each S_1 component finishes in time $\|\Pi\|$. Furthermore, the modified schedule on S_1 processes at least as many jobs as the subschedule of Π on S_1 since we replace one B-pattern scheduling two jobs with two A-patterns scheduling one job each.

For S_2 and for $\|\Pi\| \equiv 1 \pmod 4$, we allow a sequence of 3 A-patterns and $(\lfloor \|\Pi\|/4 \rfloor - 2)$ B-patterns besides the optimal schedule which uses $\lfloor \|\Pi\|/4 \rfloor$ B-patterns. More specifically, we replace two B-patterns with three A-patterns increasing the length of the schedule on S_2 by one leading to a length of $\|\Pi\|$. Furthermore, since both three A-patterns and two B-patterns contain six jobs, the modified schedule processes at least as many jobs as the optimal schedule.

$\|\Pi\|$	S_1	S_2	S_3	S_ℓ, $\ell \geq 4$
1	-	-	-	-
2	-	-	-	-
3	A	A	A	A
4	B	B	B	B
5	-	-	-	-
6	2A*	2A	2A	2A
7	A,B	A,B	A,B	A,B
8	2B	2B	2B	2B
9	2B	3A or 2B*	3A	3A
10	2A,B*	2A,B	2A,B	2A,B
11	A,2B	A,2B	A,2B	A,2B
12	3B	3B	4A or 3B	4A
13	B,2B	B,(3A or 2B)*	B,3A	B,3A
14	2A,2B*	2A,2B	2A,2B	2A,2B
15	A,3B	A,3B	A,(4A or 3B)	A,4A
16	B,3B	B,3B	B,(4A or 3B)	B,4A
17	2B,2B	2B,(3A or 2B)*	2B,3A	2B,3A
18	2A,3B*	2A,3B	2A,(4A or 3B)	2A,4A
19	A,B,3B	A,B,3B	A,B,(4A or 3B)	A,B,4A
20	2B,3B	2B,3B	2B,(4A or 3B)	2B,4A

Table 3.4.1: AB-schedules on the stars of H based on Corollary 3.13 and modification *. The number before A and B indicates the number of A- and B-patterns.

We now construct Π^* by choosing for each component the AB-schedule corresponding to makespan $\|\Pi\|$ in Table 3.4.1. Then, each single A-pattern is scheduled on A_1 and each individual B-pattern is scheduled on the whole set of machines $\mathcal{M}$. A sequence of 3 consecutive A-patterns is scheduled on A_2 in parallel to a sequence of consecutive 2 B-patterns on B_2. Similarly, a sequence of 4 consecutive A-patterns is scheduled on A_3 in parallel to a sequence of 3 consecutive B-patterns on B_3. Next, we show that this construction of Π^* is feasible with respect to G using the definition of the I, II, III-colorings.

By definition of A_1, scheduling an A-pattern on A_1 yields a feasible schedule with respect to G. This is used for the single A-patterns used when $\|\Pi\| \in \{3, 6, 10, 11, 14, 15, 18, 19\}$. Furthermore, scheduling a B-pattern on $\mathcal{M}$ is feasible since G is bipartite. This is used for the single B-patterns used if $\|\Pi\| \in \{4, 7, 8, 10, 11, 13, 14, 16, 17, 19, 20\}$. By the II-coloring (A_2, B_2), we can feasibly combine a sequence of three A-patterns on A_2 with two B-patterns on B_2 which is necessary when $\|\Pi\| \in \{9, 13, 17\}$. Finally, the III-coloring (A_3, B_3) allows us to combine a sequence of four A-patterns on A_3 with a sequence of 3 B-patterns on B_3. This is used if $\|\Pi\| \in \{12, 15, 16, 18, 19, 20\}$. We refer to Figure 3.4.2 for visualizations of the used feasibility arguments. This proves Claim 3.17.

We now show how to construct Π^* for larger makespan values.

Claim 3.18. If $|\Pi| \geq 21$, there exists an optimal schedule that is comprised of blocks of length 12 and one last block of length at most 20.

Observe that we may assume that the difference of the makespans of any two components of H is at most 3. Suppose towards contradiction, that there exists a pair of components S^a and S^b such that the makespan of S^a exceeds the makespan of S^b by at least 4. Then, we can decrease the difference by deleting the last job on S^a and inserting it after the last job on S^b. As a consequence, the length of the schedule Π on each component is at least 18 and Π schedules at least four A-patterns or three B-patterns both of length 12 on each connected component. Thus, if $\|\Pi\| \geq 21$, we can modify Π by using 4 A-patterns on A_3 and 3 B-patterns on B_3, yielding a feasible subschedule by definition of a III-coloring. Repeating this procedure until the remaining makespan is at most 20, in combination with Claim 3.17, proves Claim 3.18.

Finally, to find Π^* in polynomial time, we need to compute a schedule for each possible value $r \in [20] \setminus \{1, 2, 5\}$ by choosing the combinations obtained in Table 3.4.1 and filling up Π^* with the necessary number of blocks of length 12 needed to process all jobs. Then, the schedule with minimum makespan is an optimal schedule with respect to G. $\square$

As a consequence of Lemma 3.16, the remaining puzzle piece to obtain a polynomial time algorithm to solve SMC-UNIT on bipartite graphs is to find a star forest H and I, II, III-colorings in polynomial time. In the following, we present a four phase method to compute such a star forest and corresponding I,II,III-colorings in four phases.

In the first phase, we find an initial star forest H based on a maximum independent set and a maximum matching of G. This initial star forest admits a feasible I-coloring by choosing the maximum independent set as the leaves of the star forest giving the I-coloring A_1. However, modifications might be necessary to ensure the existence of II- and III-colorings. These modifications are done in the second and third phase of the algorithm. In the final phase, all colorings are computed based on the final star forest.

The modifications of the initial star forest in the second and third phase of the algorithms are necessary due to the possible existence of so-called alternating paths in the initial star forest. Intuitively, an alternating path is a connected sequence of star components of H such that there exists no feasible II- or III-coloring of the stars along this sequence. More formally, we define alternating paths as follows. In this definition, $C_i \simeq S_\ell$ indicates that component C_i is a star with ℓ leaves.

Definition 3.19 (Alternating paths.). *Let $H = (V, E')$ be a star forest of a bipartite graph $G = (V, E)$. Let $C_1, \ldots, C_k$ be distinct stars of H and P be a path in G on the vertices $v_1, v_2 \ldots, v_{2k-1}$ with the following properties:*

- *for even i, v_i is a leaf of star $C_{i/2+1}$*
- *for odd i, v_i is the center of star $C_{(i+1)/2}$*
- *edge $\{v_i, v_{i+1}\} \in E'$ if and only if i is even.*

We say P is an alternating path of type II *if $C_1 \simeq S_1$, $C_i \simeq S_2$ for all $i = 2, \ldots, k-1$ and $C_k \simeq S_\ell$ with $\ell \geq 3$. We say P is an* alternating path of type III *if $C_1 \simeq S_2$, $C_i \simeq S_3$ for all $i = 2, \ldots, k-1$ and $C_k \simeq S_\ell$ with $\ell \geq 4$.*

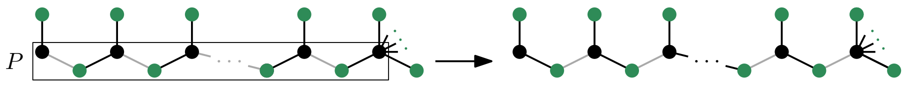


Figure 3.4.3: Alternating path P of type II. Black edges belong to H, gray edges to $G \setminus H$.

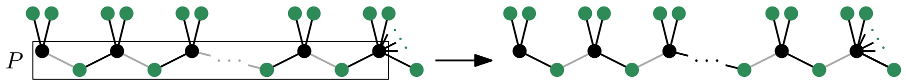


Figure 3.4.4: Alternating path of type III. Black edges belong to H, gray edges to $G \setminus H$.

Figures 3.4.3 and 3.4.4 depict an illustration of alternating paths of type II and III, respectively (on the left of each figure). The reason why these alternating paths are problematic is the following. Along an alternating path of type II, there exists no feasible II-coloring (A_2, B_2) since B_2 must contain all nodes of the first star C_1. This implies that for all intermediate stars, B_2 must contain all vertices. Finally, A_2 must contain the leaves of the final star C_k. Therefore, there exists an adjacent pair of vertices $v_{2k-3} \in B_2$ (B_3) and $v_{2k-2} \in A_2$ (A_3). Another option is to choose all leaves of the intermediate stars to be in A_2 leading to an adjacent pair of vertices $v_1 \in B_2$ and $v_2 \in A_2$. For an alternating path of type III the same issue holds where for any way of coloring the stars along the path leads to an adjacent pair of vertices in A_3 and B_3.

However, in both cases we can resolve this issue by removing the alternating path. This can be done by swapping edges along it, i.e., reassign the leaves on the path from star C_i to star C_{i-1} with $i \geq 2$. This operation is visualized in Figures 3.4.3 and 3.4.4 (right). The second and third phase of our algorithm are based on this operation. During these phases it is important that the operation maintains the leaves of the star forest and, therefore, maintains the I-coloring A_1.

Observation 3.20. *Let $H = (V, E')$ be a star forest of G containing an alternating path P of type II or III. Then, $H' := (V, E' \Delta P)$ is a star forest with the same set of leaves as H, where Δ denotes the symmetric difference.*

In the second phase, we repeat this procedure until no more alternating paths of type II exists. Then, in the third phase we do the same for alternating paths of type III while not creating any new alternating paths of type II. A detailed description of the first three phases is given in Algorithm 3.1.

Algorithm 3.1 Computing a star forest and I,II,III-colorings.

```
1:  Input: Connected bipartite graph $G = (V, E)$ with $|V| \geq 2$.
2:  Output: Star forest $H$ and I,II,III-colorings $A_1$, $(A_2, B_2)$, $(A_3, B_3)$.
    Phase 1 – Initial star forest
3:  Compute a maximum matching $M$ and a maximum independent set $I$ of $G$.
4:  Set $V(M) = \{u, v \in V : \{u, v\} \in M\}$.
5:  Set $U := V \setminus I$ (vertex cover).
6:  Set $E' := M$ and $V' := V \setminus V(M)$.
7:  while $\exists\, v \in V'$ do
8:      Find $u \in U$ such that $\{u, v\} \in E$.
9:      Add $\{u, v\}$ to $E'$ and delete $v$ from $V'$.
10: end while
    Phase 2 – Removing alternating paths of type II
11: while $\exists$ alternating path $P$ of type II do
12:     $E' = E' \Delta P$.
13: end while
    Phase 3 – Removing alternating paths of type III
14: while $\exists$ alternating path $P$ of type III do
15:     $E' = E' \Delta P$.
16: end while
```

In the final phase of the algorithm, we compute the I,II,III-colorings based on the star forest returned after Phases 1-3. To compute the I-coloring, we choose the maximum independent set of G which corresponds to the set of leaves in H. To construct the II-coloring, we first add the vertices of all stars with at least three leaves to A_2 since for these stars we want to use a sequence of three A-patterns using their leaves. Then, we add all vertices of stars with one leaf to B_2 to use a sequence of two B-patterns on these stars. We then consider all stars with two vertices and add their leaves to A_2 if the center is adjacent to a vertex in A_2. Otherwise, we add all vertices of such a star to B_2. Finally, a similar procedure is needed for the III-coloring. Algorithm 3.2 gives a detailed description of the final phase of the algorithm.

Algorithm 3.2 Computing a star forest and I,II,III-colorings (cont'd).

Phase 4 – Computing the colorings

```
16: $H := (V, E')$.
17: $A_1 := I$.
18: For each $S_\ell$ in $H$ with $\ell \geq 3$, add leaves of $S_\ell$ to $A_2$.
19: For each $S_1$ in $H$, add vertices of $S_1$ to $B_2$.
20: **while** $\exists$ $S_2$ in $H$ such that its center is adjacent to a vertex of $A_2$ in $G$ **do**
21:     Add leaves of $S_2$ to $A_2$.
22: **end while**
23: For each $S_2$ in $H$ with $V(S_2) \cap A_2 = \emptyset$, add vertices of $S_2$ to $B_2$.
24: For each $S_\ell$ in $H$ with $\ell \geq 4$, add leaves of $S_\ell$ to $A_3$.
25: For each $S_\ell$ in $H$ with $\ell \in \{1, 2\}$, add vertices of $S_\ell$ to $B_3$.
26: **while** $\exists$ some $S_3$ in $H$ such that center $v$ of $S_3$ is adjacent to some $w \in A_3$ in $G$ **do**
27:     Add leaves of $S_3$ to $A_3$.
28: **end while**
29: For each $S_3$ in $H$ with $V(S_3) \cap A_3 = \emptyset$, add vertices of all $S_3$ to $B_3$.
30: **return** $H, A_1, (A_2, B_2), (A_3, B_3)$
```

We next show that the four phases terminate in polynomial time and return a star forest with corresponding feasible I,II,III-colorings.

Lemma 3.21. *Algorithms 3.1 and 3.2 compute a star forest H of G and I-,II- and III- colorings $A_1, (A_2, B_2)$ and (A_3, B_3) of (G, H), respectively, in time polynomial in the size of G.*

Proof. We prove the statement in three steps. First, we show that the algorithm is well-defined and that the final star forest H (defined in line 1 of Algorithm 3.2) is indeed a star forest. To this end, let M be a maximum matching and I be a maximum independent set of G. Both M and I can be found in polynomial time via the maximum flow algorithm to compute a maximum matching in bipartite graphs [101, Theorem 10.5]. The complement of a maximum independent set, $U := V \setminus I$, is a minimum vertex cover and by Kőnig's Theorem [100] we know that $|U| = |M|$. More specifically, every edge in the maximum matching M contains exactly one vertex of the vertex cover U and every vertex $v \in V \setminus V(M)$ is not contained in U. Thus, for every $v \in V'$, there exists $u \in U$ such that $\{u, v\} \in E$ and line 7 of Phase 1 is well-defined. Furthermore, note that every edge of E' is incident to exactly one vertex $u \in U$. Hence, every vertex in U is the center of a star (on at least two vertices). Hence, $H = (V, E')$ is a star forest after Phase 1. Finally, by Observation 3.20 modifying the star forest $H = (V, E')$ in each iteration of Phases 2 and 3 results in a new star forest. Thus, the final subgraph $H = (V, E')$ is a star forest.

Next, we will show that the whole algorithm runs in polynomial time in the size of G. First, observe that in each iteration of Phase 2 (Phase 3), some S_1 (S_2) receives a new leaf while no new S_1 (S_2) is created. Hence, we can bound the number of iterations needed in Phase 2 (Phase 3) by the number of S_1's (S_2's) before Phase 2 (Phase 3). In each iteration of Phase 2 (Phase 3) an alternating path of type II (type III) can be found in polynomial time with a breadth-first-search method starting from a fixed S_1 (S_2). Finally, since Phase 4 runs in polynomial time in G as well since the number of stars is polynomial. Thus, the whole algorithm terminates in polynomial time.

Next, we show that Phase 4 of the algorithm computes feasible I,II,III-colorings. To this end, we first show that no alternating path of type II is created in Phase 3. Consider the star forest $H = (V, E')$ at the beginning of some iteration of Phase 3 and suppose that H does not contain an alternating path of type II. Let $C_1, \dots C_k$ be the stars of the alternating path P of type III in H. Now, we construct the new star forest H' and let let $C'_1, \dots C'_k$ denote the corresponding stars in H', i.e., the center of C'_i is equivalent to the center of C_i. Observe that $C'_i \simeq C_i \simeq S_3$ for all $i \in \{2, \dots, k-1\}$. Moreover, $C'_1 = S_3$ and $C'_k = S_\ell$ for some $\ell \geq 3$. Suppose towards contradiction that H' contains an alternating path P_2 of type II. Then, P_2 and P must intersect. If this is not the case, P_2 is also contained in H. More specifically, we know that P_2 ends in some star C'_i with $i \in \{1, \dots, k\}$. If $i > 1$, then P_2 and P share the center of C'_i. Hence, H contains P_2 as well. This contradicts the fact H contains no alternating path of type II. Otherwise, if P_2 ends in C'_1, then H contains an alternating path of type II which ends in C_2 and contains $C_1 \simeq S_2$. Again this is a contradiction. Because Phase II terminates once the star forest contains no alternating path of type II, this proves that no such path is created in Phase 3. Thus, after Phase 3, the star forest contains no alternating paths of any kind. We use this fact now to show that the computed vertex colorings are feasible.

By construction, $A_1 := I$ is a maximum independent set of G. Since the vertex cover U constitutes the centers, we have that A_1 is also a maximum independent set of H. Since this property is maintained in Phases 2 and 3 (due to Observation 3.20), A_1 is a feasible I-coloring.

Regarding the type II-coloring, the algorithm ensures that A_2 contains the leaves of all S_ℓ with $\ell \geq 3$ and that B_2 contains all vertices of S_1. Therefore, we only have to consider the S_2 components of the star forest. If possible, the algorithm adds the leaves of S_2 components to A_2. This is the case if the center of an S_2 component is adjacent to a vertex in A_2. All vertices of remaining S_2 components are added to B_2.

Thus, (A_2, B_2) fulfills properties (i), (ii) and (iii) of a II-coloring (see Definition 3.15).

It only remains to be shown that there is no vertex in A_2 which is adjacent to another vertex in $A_2 \cup B_2$. Since only leaves are added to A_2, we have that $A_2 \subset I$ and no two vertices in A_2 are adjacent to each other in G. Now, suppose towards contradiction that there exists a pair of vertices $a \in A_2$ and $b \in B_2$ such that they are adjacent in G. Denote by S^a and S^b be the star containing a and b, respectively. By construction of A_2, a is one of the leaves of S^a. Furthermore, we have $b \in U$ since, otherwise, both a and b are in A_1 which contradicts the fact that A_1 is a maximum independent set of G. If $S^b \simeq S_2$, the algorithm ensures that all leaves of S^b are contained in A_2, and hence, $b \notin B_2$. Thus, $S^b \simeq S_1$. Next, we distinguish based on the size of S^a.

Case 1: If $S^a \simeq S_\ell, \ell \geq 3$, then there exists an alternating path of type II starting in b and ending in the center of S^a via vertex a, see Figure 3.4.5 (left). This implies a contradiction.

Case 2: If $S^a \simeq S_2$, then there must exist a star S^c with at least 2 leaves such that one of its leaves is adjacent to a. If S^c has exactly 2 leaves, there exists another star S^d with a leaf adjacent to the center of S^d, see Figure 3.4.5 (right). This argument can be repeated, such that the containment of $a \in A_2$ is traced back to a star with at least 3 leaves. This, however, yields an alternating path of type II starting in b and ending in this final star. Again, this is a contradiction.

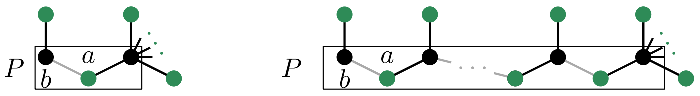


Figure 3.4.5: If vertices $a \in A_2$ and $b \in B_2$ are adjacent in G, there exists an alternating path of type II.

Similar arguments as above can be used to show that (A_3, B_3) is a feasible III-coloring or, otherwise, an alternating path of type III exists. $\square$

Putting all pieces together, Lemmas 3.10, 3.16 and 3.21 lead to a polynomial time algorithm to optimally solve SMC-UNIT on bipartite conflict graphs.

Theorem 3.22. SMC-UNIT *on bipartite conflict graphs can be solved to optimality in polynomial time.*

3.5 Generalizations to identical jobs

In this section, we consider SMC with identical jobs, i.e., ($\overleftarrow{b_j} = \overleftarrow{b}, p_j = p$, $\overrightarrow{b_j} = \overrightarrow{b}$, $\forall j$), denoted by SMC-ID. Since SMC-UNIT is a special case of SMC-ID, the inapproximability derived in Theorem 3.5 holds for SMC-ID on general conflict graphs. In the following, we generalize the approximation results shown for SMC-UNIT in Section 3.4.1 to SMC-ID. Hereto, we distinguish between *short blocking times* and *long blocking times*.

3.5.1 Identical jobs with long blocking times

We refer to long blocking times if $\max\{\overleftarrow{b}, \overrightarrow{b}\} > p$. In this setting, it is not feasible to schedule jobs in parallel on conflicting machines since the larger of the two blocking times cannot be executed on one machine in the time interval in which the other machine executes the processing time of a job. This leads to the following lower bound on the optimal makespan.

Lemma 3.23. *Consider an instance of* SMC-ID *with long blocking times on a conflict graph G and n jobs. Let $\alpha_1(G)$ be the size of a maximum independent set of G, then the optimal makespan is at least*

$$\mathrm{OPT} \geq q \left\lceil \frac{n}{\alpha_1(G)} \right\rceil ,$$

where $q = \overleftarrow{b} + p + \overrightarrow{b}$.

Proof. Consider a half-open interval of length q. Then on each machine at most one job starts processing in this interval. Furthermore, due to the long blocking times, all machines starting a job in this interval form an independent set in G, since for any distinct starting times in this interval the jobs overlap. Therefore, to process all jobs at least $\lfloor \frac{n}{\alpha_1(G)} \rfloor$ of these intervals are needed and if $\alpha_1(G)$ does not divide n, at least one more job must start earliest at time $\lfloor \frac{n}{\alpha_1(G)} \rfloor$. □

Based on this lower bound, we can show that if we can find a maximum independent set of the conflict graph in polynomial time, then constructing a schedule containing only A-patterns on this maximum independent set minimizes the makespan.

Lemma 3.24. *Consider an instance of* SMC-ID *with long blocking times, conflict graph G and n jobs. If a maximum independent set of G of size $\alpha_1(G)$ can be found in polynomial time, then an optimal schedule can be computed in polynomial time.*

Proof. Given a maximum independent set I_1 of size $\alpha_1(G)$, constructing a schedule Π consisting only of A-patterns on I_1 yields a makespan of

$$\|\Pi\| = q \left\lceil \frac{n}{\alpha_1(G)} \right\rceil .$$

Thus, Π is optimal and can be computed in polynomial time by evenly distributing jobs over I_1. □

Similarly, we can translate a polynomial time algorithm computing an approximate independent set of G into an approximation algorithm for SMC-ID with long blocking times.

Lemma 3.25. *Consider an instance of* SMC-ID *with long blocking times, conflict graph G and n jobs. If an independent set of G of size $\beta_1(G) \geq 1/\gamma\alpha_1(G)$ with $\gamma > 1$ can be found in polynomial time, then we can compute a schedule Π with makespan*

$$\|\Pi\| \leq \lceil\gamma\rceil \mathrm{OPT}.$$

Proof. Let I_1 be the approximate independent set. Let Π be the schedule obtained by evenly distributing jobs over I_1, i.e., only using 1-Patterns on I_1. Then,

$$\|\Pi\| = q\left\lceil\frac{n}{\beta_1(G)}\right\rceil \leq q\left\lceil\frac{\gamma \cdot n}{\alpha_1}\right\rceil \leq q\lceil\gamma\rceil \cdot \left\lceil\frac{n}{\alpha_1}\right\rceil = \lceil\gamma\rceil \cdot \mathrm{OPT}.$$

As Π can be constructed in polynomial time this concludes the proof. □

3.5.2 Identical jobs with short blocking times

In the remainder we consider SMC-ID where $\max\{\overleftarrow{b}, \overrightarrow{b}\} \leq p$. In contrast to long blocking times, such short blocking times allow for conflicting machines to work on jobs in parallel. By symmetry, we focus on the case where $\overleftarrow{b} \geq \overrightarrow{b}$. Similar to the case of unit jobs, induced subgraphs play a special role for this setting.

Definition 3.26 (c-colorable induced subgraph)**.** *Let $G = (V, E)$ be a graph and $I_1, \ldots I_c$ be a collection of c disjoint independent sets for some integer value c. Then, $I_1, \ldots, I_c$ induce a c-colorable subgraph of G. We say that this is a maximum induced c-colorable subgraph if the cardinality of the union of $I_1, \ldots I_c$ is maximized. We denote by $\alpha_c(G)$ the size of a maximum induced c-colorable subgraph.*

We first derive a lower bound on the optimal makespan. To this end, we define two quantities k and λ such that we can prove that in an interval of length λ at most $\alpha_{k+1}(G)$. Formally, we define k and λ as

$$k := \lfloor p/\overleftarrow{b} \rfloor \text{ and } \lambda := (k+1)\overleftarrow{b} + \begin{cases} \overrightarrow{b} & \text{if } p/\overleftarrow{b} \in \mathbb{N} \\ 0 & \text{otherwise.} \end{cases}$$

Defining λ as above is necessary to achieve this bound rather than having one additional independent set of machines.

Lemma 3.27. *Consider an instance of* SMC-ID *with short blocking times on conflict graph G and n jobs. Then, the optimal makespan is at least*

$$\mathrm{OPT} \geq \lambda \cdot \left\lceil \frac{n}{\alpha_{k+1}(G)} \right\rceil ,$$

where $\alpha_{k+1}(G)$ is the size of a maximum induced $k+1$-colorable subgraph and k and λ are as defined above.

Proof. We first show that for either of the two cases, the maximum number of jobs starting in a half-open interval of length λ is at most the size of a maximum induced $k+1$-colorable subgraph. To this end, let $I = [t, t+\lambda)$ be an arbitrary interval. Since I has length $\lambda \leq q$ and is half-open, at most one job starts on each machine within I.

Now we partition the interval as follows. First, we choose an interval $I_0 = [t, t+\lambda-(k+1)\overrightarrow{b}]$. Note that I_0 is of length $\overrightarrow{b}$ if $p/\overleftarrow{b} \in \mathbb{N}$ and length 0 otherwise. Then, we take $k+1$ disjoint, left-closed and right-open intervals each of length $\overleftarrow{b}$ denoted by $I_1, \dots, I_{k+1}$. In both cases, the length of the intervals $I_0, \dots, I_{k+1}$ add to λ. Denote by V_ℓ the set of machines starting a job in interval I_ℓ. First, note that for each ℓ the jobs processed on a machine in V_ℓ block a point in time arbitrarily close to the right end of I_ℓ. Thus, each V_ℓ is an independent set. See Figure 3.5.1 (left) for a visualization. Furthermore, if $p/\overleftarrow{b} \notin \mathbb{N}$, then $V_0 = \emptyset$ since the interval I_0 has length 0. Therefore, in this case the sets $V_1, \dots V_{k+1}$ induce a $k+1$-colorable subgraph of G.

It remains to be shown that the same holds when $p/\overleftarrow{b} \in \mathbb{N}$. Hereto, we show that $V_0 \cup V_{k+1}$ is an independent set. Consider a point in time t' which is arbitrarily close to the right end of the whole interval I. Then, the post-processing time of a job j processed on a machine V_0 starts in the interval $[t + q - \overrightarrow{b}, t + q)$. This implies that part of the post-processing time of j lies before $t + q$ and the post-processing time does not end before $t + q$. Therefore, the post-processing time of job j blocks time t'. Additionally, the pre-processing time of any job j' processed on a machine in V_{k+1} blocks time t' as well. This argument is visualized in Figure 3.5.1 (right). Consequently, $V_0 \cup V_{k+1}$ and in total $k + 1$ disjoint independent sets of machines start jobs in interval I. Thus, also in this case the number jobs starting in interval I is bounded from above by $\alpha_{k+1}(G)$.

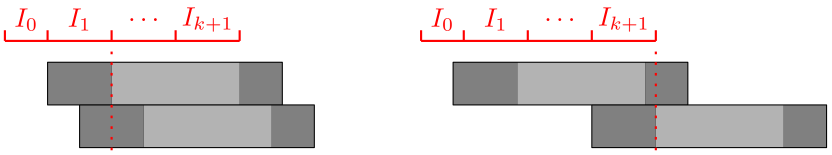


Figure 3.5.1: Illustration for the proof of Lemma 3.27: (left) two jobs starting in I_1 and (right) a job starting in I_0 and a job starting in I_{k+1}.

To finalize the proof, observe that in either of the two cases we need at least $\lfloor n/\alpha_{k+1}(G) \rfloor$ intervals of length λ to process all jobs and if $n \bmod \alpha_{k+1}(G) > 0$ at least one job of length $q \geq \lambda$ needs to be processed on top of these. This concludes the proof. $\square$

Next, to generalize the approximation algorithms introduced for SMC-UNIT in Section 3.4.1, we generalize the idea of 1- and 2-patterns.

Definition 3.28 (c-pattern). *Consider an instance of* SMC-ID *with short blocking times with conflict graph G. Let $c \in \mathbb{N}_{\geq 1}$ with $c \leq \lfloor p/\overleftarrow{b} \rfloor + 1$ and let $\mathcal{I} = (I_1, I_2, \dots, I_c)$ be a c-tuple of disjoint independent sets of G. A partial schedule of length $q + (c - 1) \cdot \overleftarrow{b}$ starting at time t is called a c-pattern on $\mathcal{I}$ if on each machine i in I_k with $k \in \{1, \dots, c\}$, there is one job starting at time $t + (k - 1)\overleftarrow{b}$.*

In the remainder, let $k = \lfloor p/\overleftarrow{b} \rfloor$. First, we show that if we are given a maximum induced $(k+1)$-colorable subgraph, we can translate this into a polynomial time approximation algorithm for SMC-ID with short blocking times.

Theorem 3.29. *Consider an instance of* SMC-ID *with short blocking times, conflict graph G and n jobs. If we can find a maximum induced $(k+1)$-colorable subgraph of G in polynomial time, then we can construct a schedule Π in polynomial time with approximation guarantee $(q + k\overleftarrow{b})/\lambda$, where $(q + k\overleftarrow{b})/\lambda < 2 + 1/(k+1) < 2.5$ and if $p/\overleftarrow{b} \in \mathbb{N}$ the approximation guarantee improves to $(q + k\overleftarrow{b})/\lambda < 1 + p/q < 2$.*

Proof. Let Π be a schedule consisting of $\lceil n/\alpha_{k+1}(G) \rceil$ many $(k+1)$-patterns on the vertex set corresponding to the given maximum induced $(k+1)$-colorable subgraph of G. We leave all other machines idle. Then, Π has makespan

$$\|\Pi\| \leq (q + k\overleftarrow{b}) \cdot \left\lceil \frac{n}{\alpha_{k+1}} \right\rceil .$$

Based on the lower bound of the optimal makespan (Lemma 3.27), we obtain

$$\frac{\|\Pi\|}{\text{OPT}} \leq \frac{q + k\overleftarrow{b}}{\lambda}.$$

If $p/\overleftarrow{b} \in \mathbb{N}$, this yields an approximation guarantee of

$$\frac{q+p}{q} = 1 + \frac{p}{q} < 2.$$

Otherwise, we obtain an approximation guarantee of

$$\frac{(k+1)\overleftarrow{b} + p + \overrightarrow{b}}{(k+1)\overleftarrow{b}} = 1 + \frac{p}{(k+1)\overleftarrow{b}} + \frac{\overrightarrow{b}}{(k+1)\overleftarrow{b}} < 2 + \frac{1}{k+1} \leq 2.5. \quad \square$$

We conclude this chapter, by showing how to use a polynomial time approximation algorithm for finding an induced $k+1$-colorable subgraph to obtain a polynomial time approximation algorithm.

Theorem 3.30. *Consider an instance of* SMC-ID *with short blocking times, conflict graph G and n jobs. If we can find an induced $(k+1)$-colorable subgraph of G of size $\beta_{k+1}(G) \geq 1/\gamma\alpha_{k+1}(G)$ with $\gamma > 1$ in polynomial time, then we can construct a schedule Π in polynomial time with approximation guarantee $2\gamma \cdot (q + k\overleftarrow{b})/\lambda$, where $2\gamma \cdot (q + k\overleftarrow{b})/\lambda < 5\gamma$ and if $p/\overleftarrow{b} \in \mathbb{N}$, the approximation guarantee improves to $2\gamma \cdot (q + k\overleftarrow{b}/\lambda) < 4\gamma$.*

Proof. Let $\mathcal{I}$ be the vertex set of the approximate induced $(k+1)$-colorable subgraph. We construct a schedule Π by using $\lceil n/\beta_{k+1}(G) \rceil$ many $(k+1)$-patterns on $\mathcal{I}$ yielding a makespan of

$$\|\Pi\| \leq (q + k\overleftarrow{b}) \cdot \left\lceil \frac{n}{\beta_{k+1}(G)} \right\rceil .$$

We now distinguish between $n \leq \beta_{k+1}(G)$ and $n > \beta_{k+1}(G)$. In the first case we also have that $n \leq \alpha_{k+1}(G)$. This yields

$$\frac{\|\Pi\|}{\text{OPT}} \leq \frac{(q + k\overleftarrow{b}) \cdot \left\lceil \frac{n}{\beta_{k+1}(G)} \right\rceil}{\lambda \cdot \left\lceil \frac{n}{\alpha_{k+1}(G)} \right\rceil} = \frac{(q + k\overleftarrow{b})}{\lambda}.$$

Due to the approximation guarantees derived in Theorem 3.29, we obtain an upper bound on the performance guarantee of 2 if $p/\overleftarrow{b} \in \mathbb{N}$ and 2.5 if $p/\overleftarrow{b} \notin \mathbb{N}$. If $n > \beta_{k+1}(G)$, we obtain $\lceil n/\beta_{k+1}(G) \rceil \leq 2 \cdot n/\beta_{k+1}(G) \leq 2\gamma \cdot n/\alpha_{k+1}(G)$. Consequently, it holds that

$$\frac{\|\Pi\|}{\text{OPT}} \leq \frac{(q + k\overleftarrow{b}) \cdot 2\gamma \cdot \frac{n}{\alpha_{k+1}(G)}}{\lambda \cdot \frac{n}{\alpha_{k+1}(G)}} \leq 2\gamma \cdot \frac{(q + k\overleftarrow{b})}{\lambda}.$$

Again, the upper bounds on the performance guarantee of 4γ if $p/\overleftarrow{b} \in \mathbb{N}$ and 5γ if $p/\overleftarrow{b} \notin \mathbb{N}$ follow from Theorem 3.29. □

Chapter 4

Just-in-time scheduling with quadratic penalties and unit jobs

4.1 Introduction

Scheduling problems with earliness and tardiness penalties have been extensively studied being motivated by the *Just-in-Time* philosophy introduced by Toyota [153] in which jobs are desired to finish as close as possible to their due date. Starting with the work of Sidney [147] who considered the task of scheduling jobs on a single machine with penalties if jobs start and complete outside of a pre-determined job specific interval, scheduling problems with earliness and tardiness penalties have attracted a lot of research. For an extensive overview we refer to the surveys in [12, 137].

In this chapter, we consider a scheduling problem with (weighted) quadratic earliness and tardiness penalties. More specifically, we consider the problem of minimizing the total (weighted) squared deviation problem for jobs with equal processing times.

This chapter is based on joint work with Camiel Koopmans and Tjark Vredeveld.

Problem Definition. We are given a set $\mathcal{J}$ of n jobs each of which must be processed on one of m identical machines. Each job j is associated with a due date d_j and a strictly positive weight w_j. We assume that the time to fully process a job j, p_j, is equal for each job, and by scaling we may assume that $p_j = 1$. Let $D = \{d^1, \ldots, d^k\}$ denote the set of due dates where k is the number of distinct due dates. We assume that the due dates are ordered in increasing value, i.e., $d^1 < d^2 < \ldots < d^k$.

For a feasible schedule σ, denote by $C_j(\sigma)$ (or just C_j when σ is clear from the context) the completion time of job j. The objective is given by $\mathsf{WSD}(\sigma) = \sum_j w_j(C_j(\sigma) - d_j)^2$. The goal is to to find a schedule σ^* that minimizes $\mathsf{WSD}(\sigma^*)$.

In the three-field notation introduced by Graham et al. [63] (see Chapter 1), this problem can be denoted by $P|p_j = 1, d_j \in D|\sum_j w_j(C_j - d_j)^2$.

Related Work. Various models have been considered in the literature for scheduling on parallel machines with *earliness and tardiness penalties*. For an extensive overview we refer to the recent survey by Rolim and Nagano [137] and the survey by Baker and Scudder [12]. In the following, we highlight some of the results.

Minimizing the total (or equivalently mean) squared deviation around a common due date on a single machine was first studied by Bagchi et al. [9]. They show that in the unrestricted setting where the due date is sufficiently large as to not impose further restrictions on the schedule, minimizing the mean squared deviation is equivalent to minimizing the completion time variance introduced by Merten and Muller [120]. Minimizing the completion time variance on a single machine was shown to be $\mathcal{NP}$-hard by Kubiak [104] implying $\mathcal{NP}$-hardness for minimizing the mean squared deviation on a single machine. Bagchi et al. [8] extended the unit weight version they considered in [9] to the setting in which the earliness penalties have a different weight than the lateness penalties. In both works, exact approaches are developed based on enumeration via branching procedures. De et al. [41, 42] as well as Srirangacharyulu and Srinivasan [151, 152] followed up on the work of Bagchi et al. [8, 9] on minimizing the mean squared deviation of jobs' completion times from a common due date by providing alternative exact approaches. Eilon and Chowdhurry [49] showed that for the unit weight case and a single due date an optimal schedule needs to be *V-shaped*, that is, the early jobs are scheduled in order of non-increasing processing times and the late jobs are scheduled in order of non-decreasing processing times. Alidaee [2] considers the generalized problem for a single machine and develops polynomial time algorithm for multiple special cases when job weights are proportional to processing times.

The majority of the literature focuses on linear earliness and tardiness penalties which are used to penalize the absolute deviation from a due date. The problem of minimizing the total absolute deviation from a common due date on a single machine was first considered by Kanet [95]. In particular, Kanet [95] considers the unrestricted setting where the due date is at least as large as the sum of the processing times and develops a polynomial time algorithm based on the idea of V-shaped schedules which process jobs around the due date with jobs before the due date being processed in non-increasing order before the due date and non-decreasing order after the due date [95]. Bagchi et al. [10] extend on this by presenting an algorithm which finds all optimal schedules in time $O(n \log n)$. Sundararaghavan and Ahmed [154] generalize these algorithms to identical parallel machines by proving the existence of an optimal schedule which evenly distributes jobs over machines. Applying the algorithms by Kanet [95] or Bagchi et al. [10] solves the problem to optimality. Hall and Posner [73] show that minimizing the total weighted absolute deviation is $\mathcal{NP}$-hard even in the unrestricted setting on a single machine and present a dynamic programming algorithm running in pseudo-polynomial time. Kovalyov and Kubiak [102] develop a fully polynomial time approximation scheme for this problem. For the restricted setting in which the common due date is smaller than the sum of the processing times, Hall et al. [71] and Hoogeveen and van de Velde [84] prove $\mathcal{NP}$-hardness even for the unweighted case on a single machine and present pseudo-polynomial time algorithms. In this context, Hoogeveen et al. [82] develop a multiplicative $4/3$-approximation algorithm. The special case of unit processing times was considered by Mosheiov and Yovel [123] who developed a polynomial time exact algorithm to minimize the total weighted absolute deviation on identical parallel machines. Beyond a common due date, Hoogeveen and van de Velde [83] consider the problem with almost equal due dates and present a pseudo polynomial time algorithm for minimizing the total weighted absolute deviation around almost equal due dates. Huynh and Tuong [157] generalize the setting to having k distinct due dates and introduce a polynomial time algorithm when k is a constant.

Our contribution. We consider various special cases of the considered problem. In Section 4.2, we focus on the unweighted setting. For this setting, we derive a structural property which allows us to solve the problem on m parallel machines by evenly distributing jobs to machines and solving m individual single machine problems. This yields an algorithm with running time $O\left(mk + k\log k\right)$. In Section 4.3, we turn our attention towards the weighted setting with a single due date. We show that optimal schedules follow an *inverse v-shape* and jobs are again evenly distributed among machines. Based on this we devise an algorithm to find the optimal solution running time $O(n \log n)$ when the due date is sufficiently large. If, however, the due date is small and imposes additional constraints on the scheduling decisions, we can find an optimal solution in time $O\left(m^2 \cdot 2^m \cdot n + n\log n\right)$. Finally, in Section 4.4, we focus on the case with k distinct due dates and a single machine. When k is a constant, we provide an add-FPTAS with approximation parameter $\rho(I) = w_{\max} = \max_j w_j$.

4.2 Unit weights

We first turn our attention towards the special case of unit weights. In this setting, an instance can be represented by the k distinct due dates $d^1 < \ldots < d^k$ and the number of jobs n_ℓ having due date d^ℓ. We first derive structural properties of optimal schedules on identical parallel machines. These lead to the insight that the problem on m machines can be solved by first distributing jobs evenly among machines and then solving the problem on each machine individually. We complement this insight by developing a polynomial time algorithm for the single machine setting.

4.2.1 Structural properties

Before deriving the needed structural properties, we first make some necessary observations on optimal schedules. Using a simple exchange argument we can show that jobs are processed in order of non-decreasing due dates. This holds even for jobs being processed on different machines.

Observation 4.1. *For any optimal schedule σ^*, it holds that, for every pair of jobs $j \neq j'$:*

$$d_j < d_{j'} \implies C_j \leq C_{j'}.$$

Furthermore, jobs with the same due date are consecutively scheduled without any idle time between; otherwise decreasing the idle time by shifting the schedule towards each other decreases the objective.

Observation 4.2. *In any optimal schedule σ^*, jobs with the same due date assigned to the same machine are consecutively scheduled without idle time between any pair of jobs.*

This property leads to the following definition.

Definition 4.3. *For a given instance with jobs $\mathcal{J}$, we say that a subset of jobs B forms a block if in an optimal schedule B must be scheduled consecutively on the same machine without idle time between any pair of jobs in B.*

We next restrict our attention towards the class of so-called *balanced* schedules which evenly distribute jobs of each due date among machines.

Definition 4.4. *Let σ be a feasible schedule. We say that σ is* balanced *if for any due date d^ℓ, the number of jobs with due date d^ℓ to be processed on any machine is either $\lfloor \frac{n_\ell}{m} \rfloor$ or $\lceil \frac{n_\ell}{m} \rceil$.*

It turns out that to find an optimal schedule, we can restrict ourselves to the class of balanced schedules which is shown in the following lemma.

Lemma 4.5. *There exists an optimal schedule for the unit weight problem on identical parallel machines among all balanced schedules.*

Proof. We prove the statement using a potential function argument. Let $n_i^\ell(\sigma)$ denote the number of jobs with due date d^ℓ assigned to machine i by schedule σ. Let σ^* be an optimal schedule minimizing

$$\sum_{\ell=1}^{k} \sum_{i=1}^{m} n_i^\ell(\sigma)^2. \tag{4.1}$$

Suppose towards contradiction that σ^* is not balanced. Then we know that for some due date d^ℓ there exists a pair of machines i, i' such that machine i processes at least two jobs more with due date d^ℓ than machine i' does. Let j_f and j_l be the first and last job with due date d^ℓ to be processed on machine i, respectively. By Observation 4.2 we know that all jobs processed between these two jobs on machine i have the same due date. Moreover, by Observation 4.1, we can distinguish between two cases.

Case 1: If no job with due date d^ℓ starts processing on machine i' before C_{j_f}, we change σ^* keeping all else the same and exchanging the partial schedules starting at time C_{j_f} between i and i'. This case is visualized in Figure 4.2.1.

Case 2: If machine i' starts processing a job with due date d^ℓ before time C_{j_f}, we know that machine i' has finished processing all jobs with due date d^ℓ before time S_{j_l} (see Figure 4.2.2). We exchange the partial schedule starting at time S_{j_l} between i and i'.

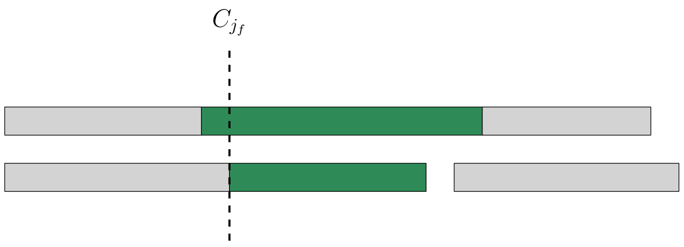


Figure 4.2.1: **Case 1:** Late start on machine i'.

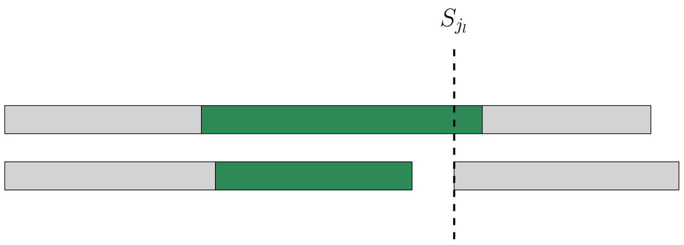


Figure 4.2.2: **Case 2:** Early completion on machine i'.

In either case, the completion time does not change for any job and, thus, σ^* remains optimal. Furthermore, Equation 4.1 changes by

$$\begin{aligned}(n_i^\ell(\sigma^*) - 1)^2 - n_i^\ell(\sigma^*)^2 + (n_{i'}^\ell(\sigma^*) + 1)^2 - n_{i'}^\ell(\sigma^*)^2 \\ = 2(n_{i'}^\ell(\sigma^*) - n_i^\ell(\sigma^*)) + 2 < 0,\end{aligned}$$

contradicting the fact that σ^* initially minimized Equation 4.1. □

The previous lemma allows us to restrict ourselves to balanced schedules when finding optimal schedules. However, since the number of jobs with each distinct due date is not necessarily divisible by m, many balanced schedules exist. We now present a refined structural property which allows us to further restrict our attention.

To this end, we construct a balanced schedule by first assigning $\lfloor n^\ell/m \rfloor$ jobs with due date d^ℓ to each machine and then distributing the jobs that have not been assigned yet over the machines. Let $\mathcal{J}_r = \{j_1, \dots, j_r\}$ be the set of remaining jobs, where $r = \sum_{\ell=1}^{k} n_\ell - m\lfloor \frac{n_\ell}{m} \rfloor$. Assume w.l.o.g. that the remaining jobs are sorted in non-decreasing order of due dates. We now show that there exists an optimal balanced schedule which assigns the remaining jobs to machines in an alternating fashion.

Lemma 4.6. *There exists an optimal balanced schedule σ^*, where machine i processes the remaining jobs in $\mathcal{J}_i = \{j_a \,:\, 1 \le a \le r,\ i \equiv a \mod m\}$, for $i = 1, 2, \dots, m$.*

Proof. Suppose towards contradiction that the described schedule σ^* is not optimal and consider an optimal balanced schedule σ. Let ℓ be such that the remaining jobs $j_1, \dots, j_\ell$ are scheduled in σ as in σ^*, but $j_{\ell+1}$ is scheduled on machine i, whereas $\ell + 1 \mod m = i'$ and $i' \neq i$. Suppose σ maximizes ℓ over all optimal balanced schedules.

We now construct a new schedule σ' such that $j_{\ell+1}$ is assigned to machine i'. To this end, let $\mathcal{J}_{\ell+1}(i)$ denote the set of jobs on machine i with due date $d_{j_{\ell+1}}$. Similarly, let $\mathcal{J}_{\ell+1}(i)$ denote the set of jobs on machine i' with due date $d_{j_{\ell+1}}$. Then $|\mathcal{J}_{\ell+1}|= |\mathcal{J}'_{\ell+1}|+1$, as otherwise i' also has a remaining job with the same due date and an index re-arrangement of this job would yield schedule σ'. Based on this, we may distinguish between two cases:

Case 1: The jobs in $\mathcal{J}'_{\ell+1}$ start processing earlier on machine i' than the jobs in $\mathcal{J}_{\ell+1}$ on machine i. Assume w.l.o.g. that $j_{\ell+1}$ is the last scheduled job in $\mathcal{J}_{\ell+1}$. Then, since $|\mathcal{J}_{\ell+1}|= |\mathcal{J}'_{\ell+1}|+1$, no job in $\mathcal{J}'_{\ell+1}$ is scheduled at time $S_{j_{\ell+1}}$. Thus, by Property 4.1, also no job with a different due date is scheduled on machine i' at $S_{j_{\ell+1}}$. Therefore, we can construct the desired schedule σ' by swapping the partial schedules on i and i' from $S_{j_{\ell+1}}$ onward without changing the objective value as the completion times remain the same for all jobs, contradicting the maximality of ℓ for σ.

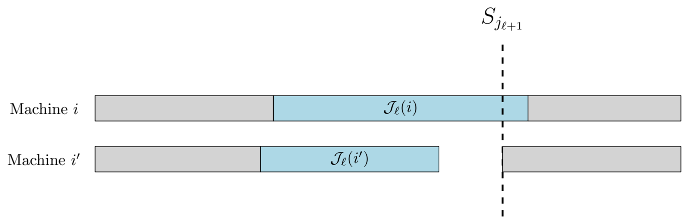


Figure 4.2.3: Visualization of **Case 1**.

Case 2: The jobs in $\mathcal{J}'_{\ell+1}$ start processing later on machine i' than the jobs in $\mathcal{J}_{\ell+1}$ on machine i. In this case, we must again distinguish between two possibilities. Assume w.l.o.g. that $j_{\ell+1}$ is the first job in the set $\mathcal{J}_{\ell+1}$ on machine i.

Case 2a: Suppose that no job j' is being processed on machine i' at $S_{j_{\ell+1}}$ on machine i. We swap the partial schedules from $S_{j_{\ell+1}}$ onward between i and i'. Again, we obtain a schedule σ' which contradicts the maximality of ℓ for σ.

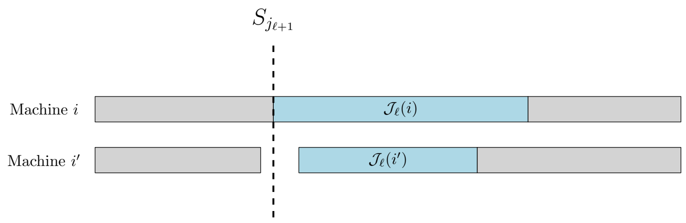


Figure 4.2.4: Visualization of **Case 2a**.

Case 2b: Suppose that a job j' is being processed at time $S_{j_{\ell+1}}$ on machine i'.

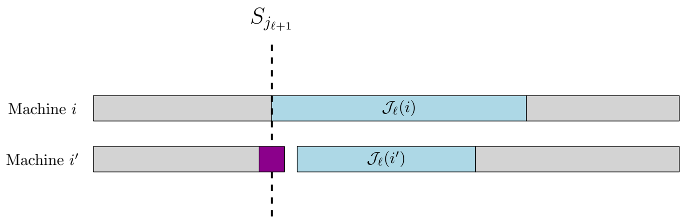


Figure 4.2.5: Visualization of **Case 2b**.

As σ is balanced, there is at most one remaining job with due date $d_{j'}$ assigned to machine i and i' each. We now show that both machines are assigned exactly the same number of jobs with due date $d_{j'}$. To this end, consider jobs $j_p, j_s \in \mathcal{J}_r$ such that j_p is assigned to machine i where p is the largest index among the remaining jobs assigned to machine i smaller than $\ell+1$ and s is the largest index among the remaining jobs assigned to i' smaller than p. By the order of the remaining jobs we know that $d_{j_s} \leq d_{j_p}$ and, therefore, by Observation 4.1:

$$d_{j'} = d_{j_s'} \implies d_{j'} = d_{j_p}.$$

We now prove that the reverse is also true. Suppose towards contradiction that j' and j_p share the same due date, i.e., $d_{j_p} = d_{j'}$ and that j_s has a different due date, i.e, $d_{j_s} \neq d_{j_p}$. This implies that machine i processes one more job with due date $d_{j'}$ than machine i' and completes processing these jobs before machine i' does. Suppose that j_p is the first job processed on machine i with due date $d_{j'}$. By Observation 4.1, no job is being processed on machine i' at time C_{j_p}. Thus, we can obtain the desired schedule σ' by exchanging the partial schedules from $C_{j_{prev}}$ onward. This step is visualized in Figure 4.2.6.

By swapping the partial schedules, we do not increase the objective while introducing idle time between two jobs with the same due date. By Observation 4.2, this schedule is not optimal implying that σ^* is not optimal. A contradiction.

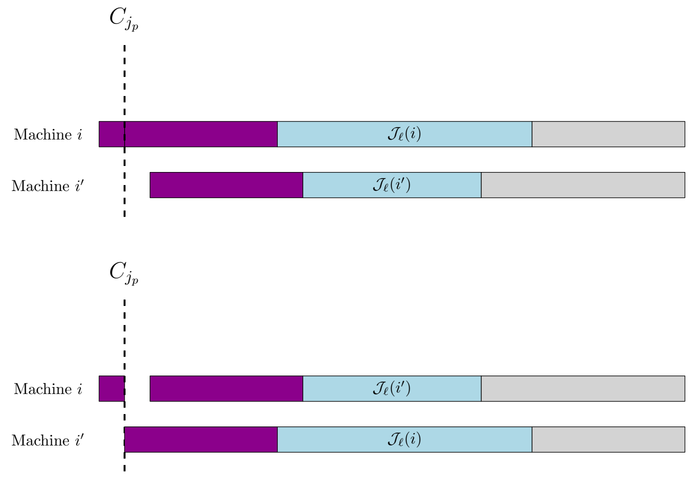


Figure 4.2.6: Visualization of contradiction for **Case 2b**.

Thus, i and i' must process equally many jobs with due date $d_{j'}$. We now use this to complete the argument. Again, we must consider Case 2a and Case 2b for the jobs with due date $d_{j'}$ processed on machines i and i'. If we are in Case 2a, we swap the partial schedules between i and i' from the start time of the first job of this due date onward. By swapping back the remaining jobs with due date $d_{j'}$ we have constructed σ'; contradicting the maximality of ℓ. If, however, we find ourselves in Case 2b, we repeat the argument above until eventually reaching Case 2a. This argument terminates since there are only a finite number of due dates to consider and we will eventually reach Case 2a or the first overall job on the two machines and swap the partial schedules or the full schedules. Then, swapping back the remaining jobs to satisfy the desired order gives σ' and the contradiction. $\square$

4.2.2 Finding optimal balanced schedules in polynomial time

By Lemma 4.6, we can restrict ourselves to a balanced schedule which assigns the remaining jobs to machines in an alternating manner. It remains to find an optimal schedule for each individual machine once this assignment is known. We now show how to use the notion of blocks to find an optimal schedule on a single machine. Towards this, we first independently compute the optimal starting time $\hat{t}_\ell$ of each individual block of jobs B_ℓ of size n_B^ℓ with common due date d^ℓ with $\ell = 1, \ldots, k$. By Observation 4.2 we know that in an optimal schedule each block must be scheduled consecutively without idle time. Therefore, if block B_ℓ starts processing at time t, the sum of squared deviations can be written as $\sum_{j=1}^{n'}(t_1 + j - d_\ell)^2$, which is minimal for

$$\hat{t}_\ell = \max\left\{d^\ell - \frac{n_\ell + 1}{2}, 0\right\}. \tag{4.2}$$

Given all individual starting times, we must repair potential conflicts between consecutive blocks which occur, whenever two consecutive blocks overlap, i.e., $\hat{t}_{\ell+1} \leq \hat{t}_\ell + n_\ell$. To overcome this, we introduce a *block merging* technique. Hereto, we consider the blocks in increasing order of due dates and check whether the optimal starting time of the current block is overlapping with the previous block. If this is the case, we merge the two blocks and adjust the optimal starting time of the new block. For example, when merging blocks B_1 and $B_2$with n_1 and n_2 jobs, we form a new block B_1' containing all jobs of B_1 and B_2 and starting at time

$$\hat{t}_{B_1'} = \max\left\{\frac{n_1}{n_1+n_2}\hat{t}_{B_1} + \frac{n_2}{n_1+n_2}(\hat{t}_{B_2} - n_a), 0\right\}. \tag{4.3}$$

This starting time is optimal for block B_1' as it minimizes

$$\sum_{j=1}^{n_1}(t + j - d_1)^2 + \sum_{j=1}^{n_2}(t + n_1 + j - d_2)^2.$$

We repeat this procedure until we reach the final block. The complete merging technique is described by Algorithm 4.1. The running time of the algorithm is split in the initialization and the merging phase. The initialization consists of computing the original k optimal starting times which takes $\mathcal{O}(k)$ time. During the merging phase, we merge blocks at most $k-1$ times as after this the schedule consists of a single block and no more merging is possible. Merging two blocks can be done in time $O(1)$ leading to a total running time of $O(k)$.

Algorithm 4.1 Block merging algorithm on a single machine.

```
 1: **Input:** Due dates $D = \{d^1, \ldots, d^k\}$ and number of jobs with each
    due date $n_1, \ldots n_k$.
    *Phase 1 – Initialization*
 2: For each block $B_\ell$ compute $\hat{t}_\ell$ with $\ell = 1, \ldots, k$ according to Equa-
    tion 4.2.
    *Phase 2 – Merging*
 3: for $\ell = 2, 3, \ldots, k$ do
 4:     while $\hat{t}_{\ell-1} + n_{\ell-1} > \hat{t}_\ell$ do
 5:         Merge blocks $B_\ell$ and $B_{\ell-1}$ into block $B_\ell$.
 6:         Update $\hat{t}_\ell$ according to Equation 4.3.
 7:         Delete block $B_{\ell-1}$.
 8:     end while
 9: end for
10: return Remaining blocks with updated starting times.
```

The merging technique on a single machine leads to the following result with an additional running time due to the sorting of due dates.

Theorem 4.7. *An optimal schedule for minimizing the total unweighted squared deviation can be found in $\mathcal{O}(k \log k)$ time on a single machine and $\mathcal{O}(mk + k \log k)$ time on m identical machines.*

4.3 Single common due date and general weights

In this section, we consider the special case of a single common due date ($k = 1$) and general job weights. Following the literature, we distinguish between the unrestricted ($d \geq (\lceil n/m \rceil + 1)/2$) and the restricted setting ($d < (\lceil n/m \rceil + 1)/2$). We derive structural properties of optimal schedules on identical parallel machines in both settings.

Observe that when considering two jobs we want the job with higher weight to finish closer to the due date even if they are assigned to different machines. To formalize this idea, we introduce the class of *inverse v-shaped schedules*.

Definition 4.8. *Let σ be a feasible schedule. We say that σ is an* inverse v-shaped *schedule if for any pair of jobs j, j' it holds that*

$$(C_j - d)^2 > (C_{j'} - d)^2 \implies w_j \leq w_{j'}.$$

We now show that any optimal schedule must be inverse v-shaped.

Lemma 4.9. *For the problem with a single due date, in the identical parallel machine environment, any optimal schedule σ^* is an inverse v-shaped schedule.*

Proof. Let σ^* be an optimal schedule and suppose for the sake of contradiction it is not inverse v-shaped. Then, there exists a pair of jobs $j, j' \in \mathcal{J}$ such that

$$(C_j - d)^2 > (C_{j'} - d)^2 \text{ and } w_j > w_{j'}.$$

Then, we can construct an alternative schedule σ' by swapping jobs j and j'. The change in the objective function is

$$\mathsf{WSD}(\sigma') - \mathsf{WSD}(\sigma^*) = \left(w_j - w_{j'}\right)\left((C_{j'} - d)^2 - (C_j - d)^2\right) < 0.$$

This contradicts the optimality of σ^*. Therefore, any optimal schedule must be inverse v-shaped. $\square$

Following the idea of the unit weight case considered in Section 4.2, we prove that there exists an optimal schedule with a balanced assignment.

Lemma 4.10. *For the problem with a single due date, in the identical parallel machine environment, any optimal schedule σ^* is balanced.*

Proof. Let σ^* be an optimal schedule and suppose towards contradiction that σ^* is not balanced. This implies that there exists a pair of machines i and i' such that i processes at least two jobs more than machine i'. As jobs are scheduled as a consecutive job per machine without idle time this implies that the length of the block on machine i is at least the length of the block on i' plus 2. Let S_{j_1}, $S_{j_1'}$ be the start time of the first job on machine i and i', respectively and C_{j_2}, $S_{j_{2'}'}$ be the completion time of the last job on machine i and i', respectively. Then, one of the following must hold:

$$S_{j_1} \leq S_{j_1'} - 1,$$
$$C_{j_2} \geq C_{j_2'} + 1.$$

Hence, we can either move the first or last job on machine i to machine i' which decreases the difference in the number of jobs by 1. Furthermore, due to this modification of the schedule, the optimal starting time of the block of jobs on machine i' and i change, which leads to a decrease in the objective value. If the first job on machine i' starts at time 0, then $C_{j_2} > C_{j_2'} + 1$ and moving j_2 to machine i' also leads to a decrease of the objective. This contradicts the optimality of σ^*. $\square$

In the restricted setting, we prove that a more specific balanced schedule ensures optimality. Hereto, let $\mathcal{M}_{early}$ and $\mathcal{M}_{late}$ denote the set of machines where the job that finishes closest to the due date finishes early or late, respectively.

Lemma 4.11. *An optimal schedule σ^* and integers $0 \leq \ell_1, \ell_2 \leq m$ exist, such that the first ℓ_1 machines in $\mathcal{M}_{early}$, and the last ℓ_2 machines in $\mathcal{M}_{late}$ process $\lfloor d \rfloor$ jobs before d, and all other machines process $\lfloor d \rfloor - 1$ job before d.*

Proof. Let σ^* be an optimal schedule. By Lemma 4.10 we know that σ^* is balanced, and hence, the number of jobs processed on each machine differs by at most 1. Furthermore, observe that every machine must finish at least $\lfloor d \rfloor - 1$ many jobs before d as otherwise we may move the last job to the front and decrease the objective value.

Consider $\mathcal{M}_{early}$ and $\mathcal{M}_{late}$ as defined earlier and assume that each set is ordered in non-decreasing order of the absolute difference between the due date and the completion time of the job finishing closest to the due date. Let ℓ_1 be the number of machines finishing $\lfloor d \rfloor$ jobs before the deadline. Suppose towards contradiction that there exists a pair of machines i, i' such that i completes $\lfloor d \rfloor$ jobs before the deadline and i' completes $\lfloor d \rfloor - 1$ jobs before the deadline and $\epsilon_{i'} < \epsilon_i$ where ϵ_i ($\epsilon_{i'}$) is defined as the difference between the due date and the completion time of the job finishing closest to the due date on machine i (i'). Let j and j' be the jobs completing closest to the deadline on machine i and i'. Since machine i schedules $\lfloor d \rfloor$ jobs before d and machine i' schedules $\lfloor d \rfloor - 1$ jobs before d we know that the first job on machine i' starts at time

$$d - \epsilon_{i'} - \lfloor d \rfloor + 1,$$

and the first job on machine i completes at time

$$d - \epsilon_i - \lfloor d \rfloor + 1.$$

Since $\epsilon_{i'} < \epsilon_i$, scheduling the first job on machine i on machine i' instead decreases the squared deviation from the due date for this job while keeping all other jobs the same. This contradicts the assumption that σ^* is optimal and implies that the ℓ_1 machines finishing $\lfloor d \rfloor$ before the deadline must be those with the smallest ϵ_i values.

Using an analogous argument we can show the same for $\mathcal{M}_{late}$. □

Finally, we show that given an assignment of jobs to machines, the optimal schedule on an individual machine can be computed in polynomial time.

Lemma 4.12. *Let σ be a feasible schedule in the identical machine environment. Consider machine i and let $\mathcal{J}(i)$ be the set of jobs assigned to i. Then, the optimal order of jobs and starting time of the first job can be found in polynomial time.*

Proof. To prove this lemma we must distinguish between the unrestricted and restricted setting for this single machine.

First, consider the unrestricted setting where $d \geq \frac{n(i)+1}{2}$ with $n(i) = |\mathcal{J}(i)|$. In an optimal schedule, $\mathcal{J}(i)$ must be scheduled without any idle time and the job completing closest to the due date satisfies $|C_j - d| \leq \frac{1}{2}$. Furthermore, by Lemma 4.9 we know that this job is the highest weight job all jobs assigned to machine i. Assume w.l.o.g.that this job finishes after the due date and has completion time $d + \epsilon$ for some $0 \leq \epsilon \leq \frac{1}{2}$. Then, we can find the optimal sequence of jobs in time $O(n \log n)$ by iteratively assigning jobs to the position closest to the due date. Finally, we need to find the starting time of this consecutive block of jobs. Assume that jobs are now indexed as $1, \ldots, n(i)$ and that the block starts processing at time t, then the sum of weighted squared deviations is

$$\sum_{j=1}^{n(i)} w_j (t + j - d)^2.$$

This is minimal for the starting time

$$\hat{t} = \frac{\sum_{a=1}^{n} w_{j_a}(d-a)}{\sum_{a=1}^{n} w_{j_a}} = d - \frac{\sum_{a=1}^{n} w_{j_a} a}{\sum_{a=1}^{n} w_{j_a}}.$$

For the restricted setting, where $d < \frac{n(i)+1}{2}$, we must distinguish between four cases. The highest weight job may either finish before or after the due date and $\lfloor d \rfloor$ or $\lfloor d \rfloor - 1$ many jobs finish before the due date. For each of these four cases, we can compute an optimal schedule in time $O(n \log n)$ by Lemma 4.9 and finding the optimal starting time. Then, taking the best among these yields an optimal solution. $\square$

We now use these structural properties and the algorithm for a single machine to find optimal schedules in the identical parallel machine environment. Again, we distinguish between the unrestricted and the restricted setting.

For the unrestricted setting, we show that the approach for a single machine given in Lemma 4.12 can be generalised to the identical parallel machine environment by assuming that machines are sorted in non-decreasing order by the absolute difference of the completion time of the job finishing closest to the due date and the due date and applying Lemma 4.9.

Theorem 4.13. *An optimal schedule σ^* for the unconstrained problem, in the identical parallel machine environment, can be found in $\mathcal{O}(n \log n)$ time.*

Proof. In order to prove this result we proceed similar to the proof of Lemma 4.12. First, observe that on each machine i the job closest to the due date completes at time $d + \epsilon_i$. We may assume that none of these jobs complete early such that $0 \leq \epsilon_i \leq \frac{1}{2}$; otherwise, we may again mirror the schedule around the due date. Since the machines are identical, we may assume that $0 \leq \epsilon_1 \leq \epsilon_2 \leq \ldots \leq \epsilon_m$. By Lemma 4.9 this again implies that we only need to sort jobs in non-increasing order of weights and assign jobs greedily to the position that is closest to the due date. This results in an assignment of jobs to machines in $O(n \log n)$ time.

Applying the algorithm underlying Lemma 4.12 we can compute the optimal starting times for the machines in time $O(m)$. $\square$

For the restricted problem, finding an optimal assignment of jobs to machines is a bit more challenging. Hereto, we use Lemma 4.11 to restrict the number of different schedules that need to be considered.

Theorem 4.14. *An optimal schedule σ^* for the constrained problem, in the identical parallel machine environment, can be found in $\mathcal{O}(m^2 \cdot 2^m \cdot n + n \log n)$ time.*

Proof. Let $\mathcal{M}_{early}$ and $\mathcal{M}_{late}$ be the set of machines such that the job finishing closest to the deadline finishes early or late, respectively. For each machine we need to guess whether it is in $\mathcal{M}_{early}$ or $\mathcal{M}_{late}$. This gives a total of $O(2^m)$ guesses. Furthermore, by Lemma 4.11 we know it suffices to guess two values l_1 and l_2 such that the first l_1 machines of $\mathcal{M}_{early}$ and the first l_2 machines of $\mathcal{M}_{late}$ finish $\lfloor d \rfloor$ jobs before the deadline and all other machines finish $\lfloor d \rfloor - 1$ jobs before the deadline. For each guess of the sets $\mathcal{M}_{early}$ and $\mathcal{M}_{late}$ this gives $O(m^2)$ many guesses. Hence, we need to consider $O(m^2 \times 2^m)$ different schedules.

For each such schedule we proceed similar to the unconstrained setting, however, with a small difference. Again, we know that the closest job to the due date on machine i completes at time $d + \epsilon_i$ with $|\epsilon_i| \leq \frac{1}{2}$. Furthermore, $\epsilon_i \geq 0$ for all $i \in \mathcal{M}_{late}$ and $\epsilon_i < 0$ for all $i \in \mathcal{M}_{early}$. Assuming that the ϵ values are sorted in non-decreasing order in each set we may greedily assign jobs to machines following a inverse v-shaped pattern by Lemma 4.9 which can be done in $O(n \log n)$. Finding the optimal starting time on each machine concludes the proof. □

4.4 The single machine problem with a constant number of distinct due dates

In this section, we consider the single machine problem with a constant number of distinct due dates and general weights. The results presented in the previous section for a single due date, depend on the insight that jobs of the same due date form consecutive blocks which is not necessarily the case when multiple due dates are considered as the following example shows.

EXAMPLE 4.1: EXAMPLE WITH NECESSARY IDLE TIME

Consider an instance with seven jobs with

- $d_1 = d_2 = 7$ and $d_3 = d_4 = d_5 = d_6 = d_7 = 5$,
- $w_1 = w_2 = w_3 = 1$ and $w_4 = w_5 = w_6 = w_7 = \epsilon$.

Figure 4.4.1 shows an optimal schedule when ϵ approaches zero.

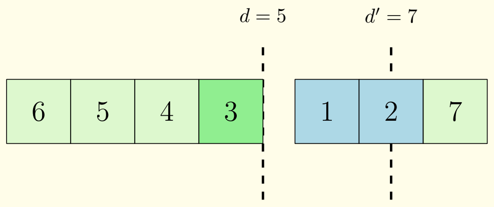


Figure 4.4.1: Optimal schedule for example with idle time

For the setting with a constant number of distinct due dates, we present a polynomial time additive approximation scheme with $\rho(I) = w_{\max} = \max_j w_j$. To this end, we use the fact that once the starting times of a schedule are fixed, we can find the best schedule satisfying these starting times in polynomial time by solving an assignment problem. This has been used in the setting of absolute earliness and tardiness penalties [157]. Based on this, the idea is to limit the number of starting times of the blocks as well as their lengths. In order to do so, we use the following observation on the maximum number of idle periods between consecutive blocks. This follows from the fact we have at most k distinct blocks as otherwise shifting two blocks together would improve the objective.

Observation 4.15. *For any optimal schedule σ^*, there will be at most $k-1$ idle periods.*

In the following, we assume that $\epsilon \leq 1$ and $1/\epsilon$ is integral. We set $\alpha = \frac{\epsilon}{2n^2}$ and we restrict ourselves to starting times that are multiples of α.

Lemma 4.16. *Consider an optimal schedule σ^* with total cost $\mathit{WSD}(\sigma^*)$. Let σ' be obtained from σ^* by rounding up the starting times to the nearest integer multiple of α. Then*

$$\mathit{WSD}(\sigma') \leq \mathit{WSD}(\sigma) + \epsilon w_{\max}.$$

Proof. By our assumption on ϵ, we know that 1 is an integral multiple of α. Therefore, it can be easily verified that by rounding up to the nearest multiple of α, the schedule remains feasible.

The increase of the contribution of a single job j, due to increasing the completion time of this job by a value $0 \leq \alpha' < \alpha$, can be bounded by

$$\begin{aligned}((C_j + \alpha') - d_j)^2 &= (C_j - d_j)^2 + 2(C_j - d_j)\alpha' + (\alpha')^2 \\ &\leq (C_j - d_j)^2 + (n + k - 1)\alpha' + \alpha' \\ &\leq (C_j - d_j)^2 + 2n\alpha' \leq (C_j - d_j)^2 + 2n\alpha,\end{aligned} \tag{4.4}$$

where the first inequality uses Corollary 4.15 to argue that between d_j and the processing of job j there are at most $k - 1$ idle periods. Any idle period between d_j and the processing of job j needs to have length less than 1 as otherwise job j can be scheduled closer to its due date. Hence, the cost of σ' can be bounded by

$$\mathsf{WSD}(\sigma') \leq \mathsf{WSD}(\sigma) + 2n\alpha \sum_j w_j \leq 2n^2\alpha w_{\max} = \mathsf{WSD}(\sigma) + \epsilon w_{\max}. \quad \square$$

We close this chapter by devising our additive polynomial time approximation scheme.

Theorem 4.17. *For any $\epsilon > 0$, a schedule σ_ϵ can be found with cost at most $\epsilon w_{\max}$ higher than the cost of an optimal schedule σ^* in time $O\left(\frac{n^{4k+2}}{\epsilon^k}\right)$.*

Proof. We can guess the starting times of the jobs, by guess the following values.

- s: The overall starting time of the schedule,
- n_ℓ: The number of jobs scheduled in consecutive block ℓ with $1 \leq \ell \leq k$, and
- q_ℓ: The length of idle period between block ℓ and block $\ell - 1$ with $2 \leq \ell \leq k$.

Since, the first job in a schedule must start in the interval $[d_1 - n, d_1]$ the number of different choices of s is bounded by $O(n/\alpha)$. For the number of jobs scheduled in each block, there are $\binom{n+k-1}{k-1} \in \mathcal{O}(n^{k-1})$ possibilities. For the length of each idle period, we need to consider $O(\frac{n}{\alpha})$ as the distance between two due dates is of order $O(n)$; otherwise the problem can be split into two sub-problems such that the optimal schedules on these sub-problems do not overlap. This gives a total number of guesses of $O\left(\frac{n^{2k-1}}{\alpha^k}\right)$. Finally, for each guess we compute the optimal assignment of n jobs to n starting times using an algorithm for the balanced assignment problem running in time $O(n^3)$ [105]. By definition of α this implies a total running time of $O\left(\frac{n^{4k+2}}{\epsilon^k}\right)$. $\square$

Chapter 5

Theoretical and empirical analysis of stochastic online scheduling policies on uniform machines

5.1 Introduction

The previous chapters dealt with different scheduling problems on parallel machines in the absence of any type of uncertainty. In this chapter, we consider the problem of minimizing the total weighted completion time on uniform related machines in the stochastic online setting. In the three-field notation (see Chapter 1), this problem is denoted by $Q||\sum_j w_jC_j$ or $Q|r_j|\sum_j w_jC_j$ depending on whether or not all jobs are available for processing right from the start. In both cases this problem is well known to be $\mathcal{NP}$-hard since the identical parallel machine environment is a special case of the uniform machine environment where all machine speeds are the same [22]. Applications of this problem appear in manufacturing, computing and compiler optimization, see e.g [27, 29].

This chapter is adapted from: M. Buchem and T. Vredeveld. “Performance analysis of fixed assignment policies for stochastic online scheduling on uniform parallel machines”. In: *Computers & Operations Research* 125 (2021)

Problem definition. We are given a set of jobs $\mathcal{J} = \{1, \ldots, n\}$ which we want to schedule non-preemptively on m machines. Each machine i operates at speed s_i and can process at most one job at a time. For each job j we are given an associated processing requirement P_j, which is a random variable. We assume that all processing requirements are stochastically independent. Moreover, we use the convention to denote the random variable for the processing requirement by P_j and a realization of this random variable by the lowercase p_j. The time it takes to process a specific job j on machine i is given by P_j/s_i. In addition to the processing requirement, every job j is given a non-negative weight w_j indicating its importance and a release date r_j before which job j may not be processed. If for every j we have that $r_j = 0$, we say that we have trivial release dates; otherwise, we say that the problem has non-trivial release dates. The information on the processing times of the jobs implies a value Δ which is an upper bound on the squared coefficient of variation:

$$\frac{\text{var}[P_j]}{E[P_j]^2} \leq \Delta \quad \forall j \in \mathcal{J}.$$

Given an instance and a stochastic online scheduling (SOS) policy, let C_j denote the completion time of job j. Note that C_j is a random variable. For an SOS policy Π and an instance $\mathcal{I}$, we denote by $\Pi(\mathcal{I})$ the random variable indicating the total weighted completion time of Π on instance $\mathcal{I}$. The expected value of this is given by $E\left[\Pi(I)\right]$. The aim is to find an SOS policy minimizing $E\left[\Pi(I)\right]$. In this chapter, we are particularly interested in the class of *fixed assignment policies* in which jobs are first assigned to machines and then single machine SOS policies are used for each specific machine.

Related work. The problem of minimizing total weighted completion time on parallel machines has been extensively studied in the scheduling literature in both the stochastic scheduling and stochastic online scheduling model.

Rothkopf [138] shows that the WSEPT-rule, the stochastic variant of Smith's rule [150], is optimal for a single machine. Kämpke [94] extends this result by showing optimality of the WSEPT-rule for identical and uniform parallel machine when job processing times follow exponential distributions and jobs admit agreeable weights, i.e., we can sort jobs such that $w_j/E\left[P_j\right] \geq w_k/E\left[P_k\right]$ also implies that $w_j \geq w_k$. Möhring et al. [122] derive a performance guarantee of $1 + \frac{(m-1)(\Delta+1)}{2m}$ for WSEPT on identical parallel machines with trivial release dates. This result is based on a linear programming relaxation which is also used to other obtain list scheduling policies with performance guarantees of $3 - \frac{1}{m} + \max\{1, \frac{m-1}{m}\Delta\}$ with release dates, and $1 + \frac{(m-1)(\Delta+1)}{2m}$ in the case without release dates [122]. The linear programming relaxation underlying these results is a generalization of a relaxation used to obtain lower bounds and approximation algorithms in the deterministic setting (see e.g. [70, 144, 145, 135]). Later, Jäger and Skutella [88] generalized the Kawaguchi-Kyan lower bound [98] to the stochastic setting proving that the the performance guarantee of WSEPT is $1 + \frac{1}{2}(\sqrt{2} - 1)(\Delta + 1)$, matching the performance guarantee of the *WSPT*-rule in the deterministic setting [98]. Many other policies have been introduced for the identical parallel machine environment. Schulz [143] develops a randomized rounding technique based on the methods developed by Correa and Wagner [37] in the deterministic setting for non-trivial release dates. This technique yields an approximation guarantee of $2 + \Delta$. For a long time the research on stochastic scheduling to minimize total weighted completion time focused on the setting of identical parallel machines. Skutella et al. [149] considered the setting of unrelated machines and presented an adaptation of the time-indexed linear programming relaxation and a randomized rounding technique for the stochastic setting. This approach leads to an approximation guarantee of $\frac{3}{2} + \frac{\Delta}{2} + \epsilon$- and a $2 + \Delta + \epsilon$-approximation in the case without and with release dates, respectively, where ϵ denotes the loss of optimality caused by using a sub-optimal solution of the time-indexed linear program as a basis for the randomized algorithm.

The stochastic online scheduling model as considered in this chapter was introduced by Megow et al. [118] and Chou et al. [34]. Megow et al. [118] introduce fixed assignment policies for identical parallel machines. In the online-list model they yield the same approximation guarantee as the WSEPT-rule. In the online-time model, they make use of the *α-Shift WSEPT*-rule which attains a $(2+\delta)$-competitive ratio for so-called δ-NBUE distributions on a single machine. This rule is used to obtain an approximation guarantee of $1+\max\{1+\frac{\delta}{\alpha}, \alpha+\delta+\frac{(m-1)(\Delta+1)}{2m}\}$ for identical parallel machines. The policies presented in [118] are generalizations of the policies presented in [117] for deterministic online scheduling problems. The asymptotic optimality of the WSEPT-rule has been shown by Chou et al. [34] for a single machine and by Gu and Lu [64] for uniform parallel machines. Furthermore, the techniques used by Schulz [143] in the stochastic offline setting is extended to the online setting by Schulz [146] yielding a competitive ratio of $2.309+1.309\Delta$ in the online-time model. Gupta et al. [67, 65] take a closer look at the power of greedy SOS policies for unrelated parallel machines. In the online-list model, they obtain an approximation guarantee of $4+2\Delta$. For the online-time model, the approximation guarantee is $\rho = (7.216+3.608\Delta)h(\Delta)$, where $h(\Delta) = 1+\sqrt{\Delta}/2 < 3/2$ for $\Delta < 1$ and, when $\Delta \geq 1$, $h(\Delta) = 1+\Delta/(\Delta+1) < 2$ [65]. Recently, Jäger [87] introduced different policies improving upon this approximation guarantee for different cases of Δ. Balseiro et al. [13] consider fixed assignment policies for the unrelated as well as uniform parallel machine environment using convex quadratic relaxations similar to the one used in [148] and linear relaxations as used in [122, 158]. Zhang et al. [173] introduce a random assignment policy in the setting of unrelated machines and show that it is asymptotically optimal for uniform parallel machines.

Next to the analysis of the performance of specific SOS policies an interesting research avenue has been concerned with deriving lower bounds on the performance guarantee of SOS policies. Since deterministic processing times are a special case of stochastic processing times, the lower bound of 1.309 shown by Vestjens [165] also holds for SOS policies. Megow et al. [118] derive a lower bound for fixed assignment policies in the identical parallel machine environment. Skutella et al. [149] show that if the performance guarantee of a fixed assignment policy depends on Δ only, then it must be at least $\frac{\Delta}{2}$ and at most $\frac{3+\Delta}{2}$. Eberle et al. [47] prove that even in the stochastic offline setting the more general class of index policies does not allow for distribution-independent approximation guarantees.

Contributions and Outline. The main contribution of this chapter is the theoretical and computational analysis of the performance of fixed assignment policies for a special case of uniform parallel machines. In particular, we consider the special case in which the machine speeds can be one of two values with $s_i \in \{1, s\}$ $(s > 1)$. In the remainder of this chapter we define this case more specifically as $s_1 = \dots s_k = s > 1$ and $s_{k+1} = \dots = s_m = 1$ for some integer k. This special case has been frequently considered in the literature. In [32, 33, 111, 113] the case is considered with $k = 1$ for makespan minimization. Liu and Yang [114] consider this case with the objective of minimizing total completion time. Dolgui et al. [46] and Leung et al. [110] consider the setting for general k with the aim to minimize makespan and total weighted completion time, respectively. Liu and Liu [113] consider the special case where $k = m - 1$.

Before turning our attention to the SOS policies, we present some lower bounds used in the analysis of the policies in Section 5.2.

Then, in Section 5.3, we consider *fixed assignment policies* for the online-list as well as the online-time model. For the online-list model, we show that the greedy approach by by Megow et al. [118] can be generalized to uniform related machines by taking into account machine speeds in the greedy assignment of jobs to machines. For this adapted policy, we derive a performance guarantee of $\rho = \frac{m}{k}\left[1 + \frac{(m-1)(\Delta+1)}{2m}\right]$ and show that it is asymptotically optimal when when weights and expected processing times are bounded from above and below by some constants. For the online-time model, we again consider a fixed assignment policy similar to the one introduced by Megow et al. [118] and adjust it taking some ingredients introduced by Gupta et al. [67]. We prove that this policy obtains an approximation guarantee of $\rho = 6\frac{m}{k} + \frac{(m-1)(\Delta-1)}{k}$.

Finally, in Section 5.4 we present a computational analysis of the presented policies and lower bounds for different classes of processing time distributions.

5.2 Lower bounds based on linear programming

In order to analyze the performance of SOS policies, lower bounds on the expected value of an optimal stochastic scheduling policy are needed. Many different techniques have been used in the literature to derive lower bounds such as linear programming relaxations or convex quadratic programming relaxation. In this chapter we make particular use of two lower bounds based on linear programming relaxations in which job completion times are used as natural date variables.

Before introducing the considered lower bounds, we define the notion of priority sets. These are based on the optimality of the WSEPT-rule on a single machine with trivial release dates [138]. For a given job j, $H(j)$ contains the jobs of higher priority according to the $WSEPT$-rule for a single machine and $L(j)$ contains the jobs of lower priority. Note that, in case of equal priority, $H(j)$ contains the jobs that arrived before j, i.e., jobs with lower index. Formally, the priority sets are defined as:

$$\begin{aligned} H(j) &= \left\{k \in J \middle| \frac{w_k}{E[P_k]} > \frac{w_j}{E[P_j]}\right\} \cup \left\{k \leq j \middle| \frac{w_k}{E[P_k]} = \frac{w_j}{E[P_j]}\right\}, \\ L(j) &= J \setminus H(j). \end{aligned} \tag{5.1}$$

Both lower bounds are based on linear programming relaxations with job completion times as variables for which the feasible region can be described using supermodular set functions. Relaxations of this form were used in the deterministic setting for different machine environments (see e.g. [70, 144, 145, 135]). An optimal solution to such a relaxation can be found using Edmond's greedy algorithm [48].

Based on this idea, Balseiro et al. [13] derive a lower bound for the uniform parallel machine environment.

Lemma 5.1 (Following from Proposition 5.2 in [13]). *Let $\mathcal{I}$ be an instance of the stochastic online scheduling problem to minimize total weighted completion time on uniform parallel machines with k fast machines. Then, the following lower bound on the expected objective value of an optimal policy OPT holds:*

$$\begin{aligned} E[OPT(I)] \geq \frac{1}{S}\sum_{j=1}^{n} w_j \sum_{k\in H(j)} E[P_k] \\ - \left[\left(\frac{1}{2}+\frac{1}{2S}-\frac{1}{s}\right)+\frac{\Delta}{2}\left(1-\frac{1}{S}\right)\right]\sum_{j=1}^{n} w_j E[P_j], \end{aligned}$$

where $S = ks + m - k$ is the sum of the machine speeds.

An alternative lower bound can be derived from the lower bound for identical parallel machines introduced by Möhring et al. [122]. By setting all machine speeds equal to the maximum machine speed, the bound in [122] leads to the following lower bound for the uniform parallel machine environment.

Lemma 5.2 (Lower bound based on fast identical machines). *Let $\mathcal{I}$ be an instance of the stochastic online scheduling problem to minimize total weighted completion time on uniform parallel machines with machine speeds $s_i \in \{1, s\}$. Then, the following lower bound on the expected objective value of an optimal policy OPT holds:*

$$E[OPT(I)] \geq \frac{1}{ms}\sum_{j=1}^{n} w_j \sum_{k\in H(j)} E[P_k] - \frac{(\Delta-1)(m-1)}{2ms}\sum_{j} w_j E[P_j].$$

5.3 SOS polices via speed-scaling

In this section, we present SOS policies for the online-list and online-time model. The policies are based on the policies introduced by Megow et al. [118] and generalized for the unrelated machine setting by Gupta et al. [68]. In the remainder of this section we let $j \to i$ denote that job j is assigned to machine i. Furthermore, let i_j denote the machine that job j is assigned to. Finally, we assume that jobs are presented to the scheduler in order of increasing index.

5.3.1 The online-list model

In the online-list model, Megow et al. [118] introduced the *MinIncrease*-policy for identical parallel machines. Here, we generalize this policy to the setting of uniform parallel machines by taking into account machine speeds when assigning jobs to machines.

The key idea is to assign jobs to machines greedily and apply the WSEPT-rule to each individual machine. Algorithm 5.1 gives a detailed description of the SpeedScaled-MinIncrease (sMI) policy.

Algorithm 5.1 SpeedScaled-MinIncrease (sMI)

Each time a job j is presented to the scheduler, it is assigned to the machine $i_j = \arg\min\{z(j,i) \,:\, 1 \leq i \leq m\}$, where $z(j,i)$ is defined as

$$z(j,i) = \frac{1}{s_i}\left(w_j \sum_{\substack{k \in H(j) \\ k<j, k \to i}} E[P_k] + E[P_j] \sum_{\substack{k \in L(j) \\ k<j, k \to i}} w_k + w_j E[P_j] \right) \tag{5.2}$$

Once all jobs are assigned to machines, the jobs on each machine are sequenced according to the *WSEPT*-rule.

In order to derive the performance guarantee of the *sMI*-policy, we first show that the expected objective value is well defined in terms of the increase functions applied in the assignment of jobs.

Lemma 5.3. *Let $E[sMI(I)]$ be the expected objective value of the sMI-policy. Then, $E[sMI(I)] = \sum_j z(j, i_j)$.*

Proof. Since the *WSEPT*-rule is applied to to each machine and jobs are assigned independent of the realization of processing times, we have:

$$\begin{aligned}
E[sMI(I)] &= \sum_j w_j E[C_j] \\
&= \sum_j w_j \sum_{k \in H(j), k \to i_j} \frac{E[P_k]}{s_{i_j}} \\
&= \sum_j w_j \left(\sum_{\substack{k \in H(j) \\ k<j, k \to i_j}} \frac{E[P_k]}{s_{i_j}} + \sum_{\substack{k \in H(j) \\ k>j, k \to i_j}} \frac{E[P_k]}{s_{i_j}} + \frac{E[P_j]}{s_{i_j}} \right).
\end{aligned} \tag{5.3}$$

Applying a simple index rearrangement yields:

$$\sum_j w_j \sum_{\substack{k \in H(j) \\ k \to k<j, i_j}} \frac{E[P_k]}{s_{i_j}} = \sum_j \frac{E[P_j]}{s_{i_j}} \sum_{\substack{k \in L(j) \\ k \to k<j, i_j}} w_k. \tag{5.4}$$

Based on this index rearrangement (5.3) gives:

$$\begin{aligned}
E[sMI(I)] &= \sum_j w_j \left(\sum_{\substack{k \in H(j) \\ k<j, k \to i_j}} \frac{E[P_k]}{s_{i_j}} + \sum_{\substack{k \in H(j) \\ k>j, k \to i_j}} \frac{E[P_k]}{s_{i_j}} + \frac{E[P_j]}{s_{i_j}} \right) \\
&= \sum_j \frac{1}{s_{i_j}} \left(w_j \sum_{\substack{k \in H(j) \\ k<j, k \to i_j}} E[P_k] + E[P_j] \sum_{\substack{k \in L(j) \\ k<j, k \to i_j}} w_k + w_j E[P_j] \right) \\
&= \sum_j z(j, i_j).
\end{aligned}$$

□

Using Lemma 5.3, we can prove the performance guarantee of the *sMI*-policy in the Online-List model.

Theorem 5.4. *Consider the stochastic online scheduling problem on uniform parallel machines without release dates and machine speeds $s_1 = ... = s_k > 1$ and $s_{k+1} = \ldots = s_m = 1$. Then, the sMI-policy yields a performance guarantee ρ, where*

$$\rho = \frac{m}{k}\left[1 + \frac{(m-1)(\Delta+1)}{2m}\right].$$

Proof. Consider some job j and let i_j be the machine to which the *sMI*-policy assigns j to, then

$$z(j, i_j) \leq z(j, i) \quad \forall i \in \{1, ..., m\}.$$

Multiplying each of these inequalities by s_i and summing over all i yields

$$S \cdot z(j, i_j) = \sum_i s_i z(j, i_j) \leq \sum_i s_i z(j, i).$$

By the definition of $z(j, i)$ we have

$$\begin{aligned} S \cdot z(j, i_j) \leq & \sum_i s_i z(j, i) \\ \leq & w_j \sum_{k \in H(j), k<j} E[P_k] + E[P_j] \sum_{k \in L(j), k<j} w_k + m\, w_j E[P_j] \\ = & w_j \sum_{k \in H(j), k \leq j} E[P_k] + E[P_j] \sum_{k \in L(j), k<j} w_k + (m-1) w_j E[P_j]. \end{aligned}$$

Next, by dividing by S and summing over all jobs together with the index rearrangement of (5.4), we obtain

$$E[sMI(I)] = \sum_j z(j, i_j) \leq \frac{1}{S} \sum_j w_j \sum_{k \in H(j)} E[P_k] + \frac{(m-1)}{S} \sum_j w_j E[P_j]. \tag{5.5}$$

Finally, we use the simple lower bound $E[OPT(I)] \geq \sum_j w_j \frac{E[P_j]}{s}$ and the lower bound of Lemma 5.2 to obtain:

$$\begin{aligned}
E[sMI(I)] \leq & \frac{1}{S}\sum_j w_j \sum_{k\in H(j)} E[P_k] + \frac{m-1}{S}\sum_j w_j E[P_j] \\
\leq & \frac{ms}{S}\left[\frac{1}{ms}\sum_j w_j \sum_{k\in H(j)} E[P_k] - \frac{(m-1)(\Delta-1)}{2ms}\sum_j w_j E[P_j]\right. \\
& \left. + \frac{(m-1)(\Delta-1)}{2ms}\sum_j w_j E[P_j]\right] + \frac{(m-1)s}{S}E[OPT(I)] \\
\leq & \frac{ms}{S}\left[E[OPT(I) + \frac{(m-1)(\Delta-1)}{2m}E[OPT(I)]\right] \\
& + \frac{(m-1)s}{S}E[OPT(I)] \\
\leq & \frac{ms}{S}E[OPT(I)] + \frac{(\Delta+1)(m-1)s}{2S}E[OPT(I)].
\end{aligned}$$

As the sum of speeds is defined as $S = ks + m - k$, we have:

$$\begin{aligned}
E[sMI(I)] \leq & \frac{ms}{S}E[OPT(I)] + \frac{(\Delta+1)(m-1)s}{2S}E[OPT(I)] \\
= & \frac{ms}{ks+m-k}E[OPT(I)] + \frac{(\Delta+1)(m-1)s}{2(ks+m-k)}E[OPT(I)] \\
< & \frac{m}{k}E[OPT(I)] + \frac{(\Delta+1)(m-1)}{2k}E[OPT(I)]. \qquad \square
\end{aligned}$$

Corollary 5.5. *Consider the stochastic online scheduling problem on uniform parallel machines without release dates with $k = m-1$ fast machines. Then, the sMI-policy yields a performance guarantee ρ, where*

$$\rho = \frac{3}{2} + \frac{1}{m-1} + \frac{\Delta}{2}.$$

Next, we show that the *sMI*-policy is asymptotically optimal. This result holds even for general machine speeds.

Theorem 5.6. *Suppose $\underline{w} \leq w_j \leq \overline{w}$, $\underline{p} \leq E[P_j] \leq \overline{p}$ holds for all jobs j for some constants $\underline{w}$, $\overline{w}$, $\underline{p}$ and $\overline{p}$. Then, the sMI-policy is asymptotically optimal, i.e.,*

$$\frac{E[sMI(I)] - E[OPT(I)]}{E[OPT(I)]} \xrightarrow{n\to\infty} 0.$$

Proof. Let

$$f(s,k) = \left(\frac{1}{2} + \frac{1}{2S} - \frac{1}{s}\right) + \frac{\Delta}{2}\left(1 - \frac{1}{S}\right)$$

with $S = ks + m - k$. Applying the lower bound of Balseiro et al. [13] (see Lemma 5.1) and using (5.5) we can bound the expected value of the *sMI*-policy by

$$\begin{aligned} E[sMI(I)] &\leq \frac{1}{S}\sum_j w_j \sum_{k\in H(j)} E[P_k] + \frac{m-1}{S}\sum_j w_j E[P_j] \\ &\leq E[OPT(I)] + f(s,k)\sum_j w_j E[P_j] + \frac{m-1}{S}\sum_j w_j E[P_j]. \end{aligned}$$

From this it follows that the relative error compared to the expected objective value of an optimal policy is bounded from above as follows:

$$\frac{E[sMI(I)] - E[OPT(I)]}{E[OPT(I)]} \leq f(s,k) \cdot \frac{\sum_j w_j E[P_j]}{E[OPT(I)]}.$$

To obtain the asymptotic result, we need to bound the ratio between $\sum_j w_j E[P_j]$ and $E[OPT(I)]$. The assumptions that $w_j \leq \overline{w}$ and $E[P_j] \leq \overline{p}$ can be used to obtain an upper bound on the sum of weighted expected processing times

$$\sum_j w_j E[P_j] \leq \sum_j \overline{wp} \leq n\overline{wp}.$$

On the other hand, we use the assumptions that $w_j \geq \underline{w}$ and $E[P_j] \geq \underline{p}$ to obtain a lower bound for $E[OPT(I)]$

$$\sum_{j=1}^{n} w_j \sum_{k \in H(j)} E[P_k] \geq \sum_{j=1}^{n} \underline{w} \sum_{k \in H(j)} \underline{p} \geq \frac{n(n+1)}{2} \underline{wp}.$$

These bounds in combination with the lower bound on $E[OPT(I)]$ in Lemma 5.1 yield the following bound:

$$\begin{aligned}
\frac{\sum_j w_j E[P_j]}{E[OPT(I)]} &\leq \frac{\sum_j w_j E[P_j]}{\frac{1}{S}\sum_{j=1}^{n} w_j \sum_{k \in H(j)} E[P_k] - f(s,k)\sum_j w_j E[P_j]} \\
&\leq \frac{1}{\frac{1}{S}\frac{\sum_{j=1}^{n} w_j \sum_{k \in H(j)} E[P_k]}{\sum_j w_j E[P_j]} - f(s,k)} \\
&\leq \frac{1}{\frac{1}{S}\frac{\frac{n(n+1)}{2}\underline{wp}}{n\overline{wp}} - f(s,k)} \\
&= \frac{1}{\frac{1}{S}\frac{(n+1)\underline{wp}}{2\overline{wp}} - f(s,k)} \xrightarrow{n \to \infty} 0.
\end{aligned}$$

Here, the limit is due to the fact that $\frac{\underline{wp}}{\overline{wp}}$ is constant and S, s and Δ are independent of n. Hence, we have that

$$f(s,k) \cdot \frac{\sum_j w_j E[P_j]}{E[OPT(I)]} \xrightarrow{n \to \infty} 0. \qquad \square$$

5.3.2 The online-time model

In this section, we consider the online-time model where jobs with non-trivial release dates are presented to the scheduler. For this model, we again develop a fixed assignment policy based on a random assignment of jobs to machines similar to the policy studied in [118] for the identical parallel machine environment. Hereto, we make use of a speed-weighted probability of assigning a job to a machine instead of a uniform probability. For each individual machine, we use a shifted variant of the WSEPT-rule following ideas of an earlier version of the policy of Gupta et al. [68]. The idea behind this shifted rule is to introduce unforced idle time, i.e., intervals in which machines do not process jobs even though some jobs are available for processing. While this additional idle time seems counterintuitive, it is necessary to obtain an upper bound on the expected completion time of a job under the considered policy as can be seen in the example in Remark 2 of Section 6 in [66]. Recently, Gupta et al. [67] presented a different way of introducing unforced idleness. However, this requires keeping track of a solution to a deterministic scheduling problem and complicates the description of the SOS policy, whereas our analysis does not benefit from the more recent description of the policy. Our policy for uniform parallel machines is described in Algorithm 5.2. We refer to this policy as the *QRandom* policy.

Algorithm 5.2 *QRandom*-policy

```
Every time a job $j$ is presented to the scheduler at its release date $r_j$,
it is assigned to machine $i$ with probability $\sigma_i = \frac{s_i}{S}$.

Once a job is assigned to a machine, a modified release date for this
job is computed as $\hat{r}_j = max\{r_j, \frac{E[P_j]}{s_i}\}$.

Whenever a machine $i$ falls idle at time $t$, we consider the set of jobs
assigned to $i$ such that $\hat{r}_j \leq t$. Among these jobs, let job $k$ be the job
of highest priority, i.e., $\frac{w_k}{E[P_k]} > \frac{w_j}{E[P_j]}$ or $\frac{w_k}{E[P_k]} = \frac{w_j}{E[P_j]}$ and $k < j$ for all
$j \neq k$ also assigned to $i$. The policy then inserts an additional forced
idleness of $\frac{E[P_k]}{s_i}$ before starting job $k$.
```

In order to prove the performance guarantee of the *QRandom*-policy, we make use of the following bound on the expected completion time of a job given that it is assigned to a specific machine. This bound is based on Lemma 5 in [68] which gives a similar bound for unrelated machines.

Lemma 5.7. *Under the QRandom policy, if a job j is assigned to machine i, then the expected completion time of j given $j \to i$ is bounded as follows:*

$$\begin{aligned} E[C_j^Q | j \to i] \leq & 4 \max\left\{ r_j, \frac{E[P_j]}{s_i} \right\} + 2 \sum_{k \to i, k \in H(j)} \frac{E[P_k]}{s_i} \\ \leq & 4 r_j + 4 \frac{E[P_j]}{s_i} + 2 \sum_{k \to i, k \in H(j)} \frac{E[P_k]}{s_i}. \end{aligned}$$

Based on this lemma, the following performance guarantee can be derived for the *QRandom* policy.

Theorem 5.8. *Consider the stochastic online scheduling problem on uniform parallel machines with release dates with k fast machines with speed s and $m - k$ slow machines with speed 1. Then, the QRandom policy yields a performance guarantee ρ, where*

$$\rho = 6 \frac{m}{k} + \frac{(m-1)(\Delta - 1)}{k}.$$

Proof. First of all, based on Lemma 5.7, we know that the conditional expected completion time of job j is bounded from above. Taking into consideration the probabilities with which a job is assigned to a machine, the unconditional expected completion time under the *QRandom* policy yields:

$$\begin{aligned} E[C_j^Q] = & \sum_i \sigma_i E[C_j | j \to i] \\ \leq & 4 \sum_i \frac{s_i}{S} r_j + 4 \sum_i \frac{s_i}{S} \frac{E[P_j]}{s_i} + 2 \sum_i \frac{s_i}{S} \left(\sum_{k \to i, k \in H(j)} \frac{E[P_k]}{s_i} \right) \\ \leq & 4 r_j + 4 \frac{ms}{S} \frac{E[P_j]}{s} + \frac{2}{S} \sum_{k \in H(j)} E[P_k]. \end{aligned}$$

Here, the first inequality follows from the bound in Lemma 5.7 and the second inequality follows from the fact that $\sum_i \frac{1}{S} = m/S$ that in the last part of the upper bound the machine individual speed is cancelled out. Further simplifying the upper bound yields:

$$\begin{aligned}
E[C_j^Q] \leq & 4\left(r_j + \frac{ms}{ks+m-k}\frac{E[P_j]}{s}\right) + \frac{2}{ks+m-k}\sum_{k\in H(j)} E[P_k] \\
\leq & 4\left[r_j + \frac{m}{k}\frac{E[P_j]}{s}\right] + \frac{2m}{k}\frac{1}{ms}\sum_{k\in H(j)} E[P_k] \\
\leq & 4\frac{m}{k}\left(r_j + \frac{E[P_j]}{s}\right) + \frac{2m}{k}\frac{1}{ms}\sum_{k\in H(j)} E[P_k].
\end{aligned}$$

Let $E[Q(I)] = \sum_j w_j E[C_j^Q]$ be the expected objective of the *QRandom*-policy. Then, we can use the bound on the expected completion times above and the lower bound on the optimal solution in Lemma 5.2 to derive the performance guarantee of the *QRandom*-policy:

$$\begin{aligned}
E[Q(I)] \leq & 4\frac{m}{k}\sum_j w_j\left(r_j + \frac{E[P_j]}{s}\right) + 2\frac{m}{k}\frac{1}{ms}\sum_j w_j \sum_{k\in H(j)} E[P_k] \\
\leq & 4\frac{m}{k}E[OPT(I)] + 2\frac{m}{k}\left[\frac{1}{ms}\sum_j w_j \sum_{k\in H(j)} E[P_k]\right. \\
& \left. - \frac{(m-1)(\Delta-1)}{2ms}\sum_j w_j E[P_j] + \frac{(m-1)(\Delta-1)}{2ms}\sum_j w_j E[P_j]\right] \\
\leq & 4\frac{m}{k}E[OPT(I)] + \frac{2m}{k}\left[E[OPT(I)] + \frac{(m-1)(\Delta-1)}{2m}E[OPT(I)]\right] \\
\leq & 6\frac{m}{k}E[OPT(I)] + \frac{(m-1)(\Delta-1)}{k}E[OPT(I)].
\end{aligned}$$

From the first to the second inequality we use the fact that $E[OPT(I)] \geq \sum_j w_j \left(r_j + \frac{E[P_j]}{s}\right)$ and from the second to third we use the lower bound based on fast identical machines and $E[OPT(I)] \leq \sum_j w_j \frac{E[P_j]}{s}$. □

Corollary 5.9. *Consider the stochastic online scheduling problem on uniform parallel machines with release dates with $m-1$ fast machines with speed s and 1 slow machine with speed 1. Then, the QRandom policy yields a performance guarantee ρ, where*

$$\rho = 5 + \Delta + \frac{6}{m-1}.$$

5.4 Computational study

In this section, we present the results of a computational study of both the policies presented in the previous section as well as the two lower bounds used in the theoretical analysis. In order to obtain valuable insights a large set of scenarios is considered for the special case with $k = m - 1$.

For the lower bounds, the main goal is to analyse how they compare to each other in different scenarios and to be able to choose the most adequate lower bound when calculating the experimental performance of the policies. For the policies, the goal is to analyse the realized performance in various settings and see how this compares to the theoretically derived performance guarantees as well as verifying how the asymptotic behaviour of the *sMI*-policy is visible in practice.

5.4.1 Lower bound analysis

In practice, we want to choose the most accurate lower bound of the expected objective value of an optimal stochastic scheduling policy to accurately estimate the realized performance guarantees of considered policies. Therefore, we need to choose the largest available bower bound. In the following, we assess the two lower bounds used in the derivation of the theoretical performance guarantees in practice. In order to compare the two lower bounds, we analyze their behavior with respect to three parameters: number of machines, number of jobs and the speed of the fast machines. In particular, we are interested in answering the following questions:

L.1 What is the magnitude of the effect the number of machines, the number of jobs and the speed of the fast machines have on the quality of the lower bounds?

L.2 How do the two theoretical lower bounds compare to each other in practice?

We consider multiple scenarios based on the above mentioned parameters. To this end, we make use of randomly generated instances using the values shown in Table 5.4.1. For each scenario, given by a combination of parameters, we calculate the lower bounds for 250 randomly generated instances and compute the ratio of the lower bounds over the simple benchmark bound. Finally, we calculate the average ratio for every scenario and compare the ratios for the two lower bounds.

Parameter	**Values**
m	2,5,10
n	10,20,50,100,200,500,1000
s	2,3,4,5,6,7,8,9

Table 5.4.1: Parameter choice for lower bound analysis.

Although, the bound on the squared coefficient of variation, Δ, has an impact on the lower bounds as well, we only focus on instances exponential distributions with $\Delta = 1$. Under this assumption, the lower bounds presented in Lemmas 5.1 and 5.2 give the following bounds, respectively:

$$\frac{1}{ks+m-k}\sum_{j=1}^{n} w_j \sum_{k\in H(j)} E[P_k] - (1-\frac{1}{s})\sum_{j=1}^{n} w_j E[P_j], \tag{5.6}$$

$$\frac{1}{ms}\sum_{j=1}^{n} w_j \sum_{k\in H(j)} E[P_k]. \tag{5.7}$$

Hereafter, we refer to the lower bounds in Lemmas 5.1 and 5.2 as *Uniform-Bound* and *Fast-Bound*, respectively. Note that for exponentially distributed processing times, the Fast-bound is solely given by the expected objective value of the *WSEPT*-rule applied to a single machine running at speed ms.

In order to evaluate the lower bounds, we use a simple lower bound as a benchmark which is given by

$$\frac{1}{s}\sum_{j} w_j E[P_j].$$

Based on this benchmark bound, we obtain the following ratios for the Uniform-bound and Fast-bound, respectively:

$$\frac{s}{ks+m-k}\frac{\sum_{j=1}^{n} w_j \sum_{k\in H(j)} E[P_k]}{\sum_{j=1}^{n} w_j E[P_j]} - (s-1), \tag{5.8}$$

$$\frac{1}{m}\frac{\sum_{j=1}^{n} w_j \sum_{k\in H(j)} E[P_k]}{\sum_{j=1}^{n} w_j E[P_j]}. \tag{5.9}$$

The results of the lower bound study can be found in Tables 5.4.2 - 5.4.4. Towards answering Question **L.1**, we first take a look at the two lower bounds individually. In Tables 5.4.2a - 5.4.4a, the computational results for the Uniform-bound are presented. First of all, one can observe that the quality of the lower bound highly depends on each of the three parameters. The Uniform-bound shows to be more accurate for larger instances, i.e., the ratio compared to the simple bound grows as n grows larger. With respect to m and s, one can observe that with a growing number of machines or a larger maximum speed, the Uniform-bound becomes weaker when being compared to the simple bound. Most striking is the observation that for large values of m and s and small instances the lower bound becomes negative. This is the case since the right part of (5.8) becomes large for larger values of s whereas the left part is small for instances with few jobs. Therefore, the Uniform-bound (and its ratio compared to the benchmark bound) become negative for small instances with large values of s. This effect becomes larger for scenarios with many machines since the factor of the first part of (5.8) becomes smaller.

For the Fast-bound (see Tables 5.4.2b - 5.4.4b) we can also observe that its quality compared to the simple bound increases when the number of jobs increase. Again, this is to be expected since the ratio in (5.8) increases as the instance size increases. Similarly, to the Uniform-bound, we can observe that the number of machines has a negative impact on the quality of the Fast-bound. This is due to the factor of $1/m$ in (5.8). For the machine speeds, however, the Fast-bound remains relatively stable for any number of jobs and machines. This is due to the ratio of the Fast-bound and the benchmark bound being independent of s.

n \ s	2	3	4	5	6	7	8	9
10	1.235	0.446	−0.331	−1.228	−2.203	−3.091	−4.067	−5.043
20	2.964	2.512	1.728	0.978	0.156	−0.790	−1.691	−2.658
50	8.263	8.418	8.167	7.587	6.854	6.118	5.384	4.503
100	17.061	18.299	18.780	18.675	18.149	17.720	17.022	16.412
200	34.735	38.242	39.877	40.534	40.933	40.972	40.616	40.219
500	87.549	97.553	103.111	106.847	108.888	110.153	111.251	111.652
1000	175.897	196.729	209.065	216.585	222.248	226.321	228.655	230.527

(a) Uniform-bound.

n \ s	2	3	4	5	6	7	8	9
10	1.676	1.631	1.668	1.663	1.632	1.662	1.650	1.643
20	2.973	3.008	2.955	2.987	3.008	2.977	2.986	2.968
50	6.947	6.946	6.979	6.952	6.915	6.925	6.966	6.946
100	13.546	13.533	13.613	13.605	13.503	13.554	13.512	13.562
200	26.802	26.828	26.798	26.721	26.794	26.841	26.784	26.788
500	66.412	66.369	66.319	66.508	66.434	66.373	66.516	66.473
1000	132.673	132.486	132.540	132.351	132.561	132.755	132.556	132.515

(b) Fast-bound.

Table 5.4.2: Lower bound ratios compared to simple bound for $m = 2$.

n \ s	2	3	4	5	6	7	8	9
10	−0.255	−1.228	−2.220	−3.211	−4.209	−5.198	−6.211	−7.201
20	0.327	−0.630	−1.601	−2.583	−3.572	−4.553	−5.549	−6.562
50	2.087	1.197	0.250	−0.693	−1.639	−2.640	−3.637	−4.650
100	5.030	4.258	3.387	2.471	1.487	0.530	−0.403	−1.407
200	10.908	10.378	9.574	8.738	7.824	6.921	5.963	5.002
500	28.550	28.674	28.218	27.652	26.894	26.040	25.213	24.348
1000	58.051	59.199	59.357	59.103	58.583	57.919	57.200	56.506

(a) Uniform-bound.

n \ s	2	3	4	5	6	7	8	9
10	0.671	0.669	0.663	0.663	0.659	0.664	0.651	0.657
20	1.195	1.187	1.189	1.190	1.190	1.199	1.197	1.182
50	2.779	2.771	2.763	2.778	2.801	2.784	2.774	2.755
100	5.427	5.424	5.429	5.436	5.406	5.410	5.442	5.421
200	10.717	10.728	10.688	10.700	10.686	10.706	10.694	10.690
500	26.595	26.584	26.535	26.588	26.579	26.548	26.576	26.597
1000	53.146	53.039	53.003	53.007	52.985	52.961	52.965	53.038

(b) Fast-bound.

Table 5.4.3: Lower bound ratios compared to simple bound for $m = 5$.

n \ s	2	3	4	5	6	7	8	9
10	−0.654	−1.645	−2.647	−3.637	−4.640	−5.636	−6.638	−7.638
20	−0.369	−1.362	−2.361	−3.358	−4.347	−5.344	−6.343	−7.349
50	0.463	−0.513	−1.493	−2.499	−3.488	−4.480	−5.479	−6.476
100	1.855	0.904	−0.074	−1.042	−2.047	−3.041	−4.023	−5.016
200	4.653	3.743	2.776	1.841	0.845	−0.151	−1.125	−2.125
500	12.999	12.247	11.378	10.438	9.501	8.515	7.559	6.609
1000	26.872	26.391	25.658	24.844	23.889	22.991	22.032	21.083

(a) Uniform-bound.

n \ s	2	3	4	5	6	7	8	9
10	0.329	0.331	0.327	0.334	0.330	0.333	0.331	0.330
20	0.600	0.595	0.591	0.591	0.599	0.600	0.599	0.593
50	1.390	1.388	1.394	1.381	1.386	1.390	1.388	1.388
100	2.712	2.711	2.707	2.721	2.707	2.705	2.717	2.719
200	5.370	5.360	5.343	5.374	5.358	5.348	5.361	5.353
500	13.299	13.297	13.299	13.283	13.293	13.271	13.285	13.311
1000	26.478	26.498	26.509	26.536	26.481	26.506	26.492	26.498

(b) Fast-bound.

Table 5.4.4: Lower bound ratios compared to simple bound for $m = 10$.

To finalize the analysis of the computational results for the lower bounds we consider Question **L.2**. We can observe that the Uniform-bound is less accurate for smaller instances due to the observations made above. For larger instances, the Uniform-bound is preferable compared to the Fast-bound. The difference between the two becomes smaller for large values of m since in these scenarios we have to speed up relatively fewer machines for the Fast-Bound. Overall, none of the two bounds clearly dominates the other for the considered special case. As Δ, keeping all other parameters the same, has a negative effect on both lower bounds in theory, we expect it to have a similar effect on the Uniform-bound as s and on the Fast-bound as m.

5.4.2 Policy analysis

To evaluate the policies introduced in Section 5.3 from an empirical point-of-view, we construct a simulation study to compare the realized performance guarantees to the theoretical performance guarantees for different scenarios. The computational study aims to answer the following questions:

P.1 How does the realized performance of the policies compare to the theoretical performance guarantee?

P.2 What is the magnitude of the effect the number of machines, the number of jobs, the speed of the fast machines and the degree of variation of the processing times have on the performance guarantee of the policies in practice?

In order to obtain a sufficient collection of scenarios to answer these questions, different parameters are considered. These parameters include: the number of jobs, the number of machines, the speed of the fast machines and the upper bound on the squared coefficient of variation, Δ. For n, m and s we select the same values as for the analysis of the lower bounds (see Table 5.4.1). Again, we restrict ourselves to the special case with $m-1$ fast machines. To analyse the effect of the degree of variation, we consider different classes processing time distributions with different values of Δ. First, we consider exponential distributions with $\Delta = 1$ and for each job j a random value $\mu_j \in \{1, 2, ..., 50\}$ is generated such that $E[P_j] = \mu_j$. Secondly, we consider discrete uniform random variables. For each job j we generate a value μ_j such that $P_j \sim Uniform(0, 2\mu_j)$, $E[P_j] = \mu_j$ and $\Delta = \frac{1}{3}$. In order to analyse the effect of Δ on the performance of the policies we choose log-normal distributions as another family of distributions. Trietsch et al. [156] showed that both in theory and practice the log-normal distribution has many features that appropriately model processing times in both machine and activity scheduling. Among these features are non-negativity and the possibility to easily test distributions with high coefficients of variation. We choose log-normal distributions are with $P_j \sim Lognormal(\mu_j, \sigma^2)$ where $\sigma^2 = \ln 11$ for every job j. This gives a bound on the squared coefficient of variation of $\Delta = e^{\sigma^2} - 1 = 10$. For each scenario, we implement the policies for 50 different instances and compute the expected objective value. Based on this, the realized performance guarantee is calculated by using the maximum of the Uniform-bound, the Fast-bound and the aforementioned simple bound. Furthermore, for the *QRandom*-policy, we use an adapted version of the algorithm in [122, 158] to compute a lower bound taking into account non-trivial release dates (see Appendix 5.A).

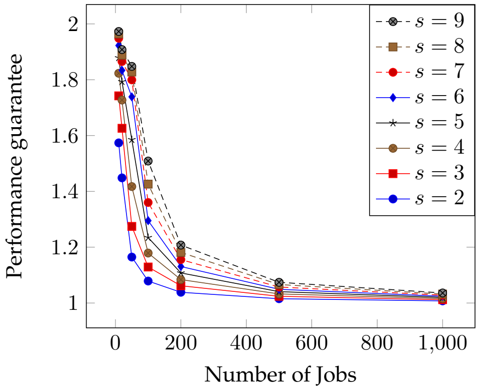
In the following, we present some of the simulation results for both policies. For each class of processing time distributions, we show the results for $m = 2$ and varying machine speeds and number of jobs as well as the results for $s = 5$ and varying number of machines and jobs. The complete results can be found in the Appendix (see Tables 5.B.1 - 5.B.18).

Simulation results for *sMI* policy. The results of the selected scenarios for the *sMI*-policy can be seen in Figures 5.4.1a - 5.4.3b. For the special case with $m - 1$ fast machines the performance guarantee is

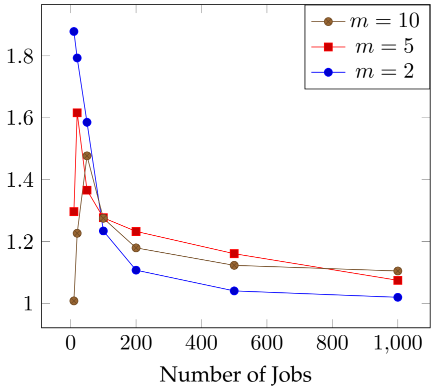
$$\frac{3}{2} + \frac{1}{m-1} + \frac{\Delta}{2}$$

First of all, regarding Question **P.1** we can observe that for each family of processing time distributions the realized performance guarantee is smaller than the theoretical performance guarantee. Furthermore, all figures show the asymptotic optimality of the *sMI*-policy as the realized performance guarantee tends to 1 as the number of jobs increases. With respect to Question **P.2**, we can see that the machine speeds seem to have a small impact on the performance of the *sMI*-policy. Finally, the performance guarantee is larger for larger values of Δ which is in line with the expected impact of Δ based on the theoretical performance guarantee. This impact diminishes for larger instances.

The most striking observation is that for large values of m, s or Δ the expected behavior of the performance guarantee is observed only for larger instances. Specifically, the performance guarantee first increases with the number of jobs. An example of this can be seen in Figure 5.4.1b, where the performance guarantee for $m = 10$ increases up to $n = 50$. This coincides with the behavior of the lower bounds observed earlier. As Table 5.4.4b shows, the simple bound is chosen for instances with less than 50 jobs, possibly causing the observed behavior as the simple bound can be accurate for smaller instances.

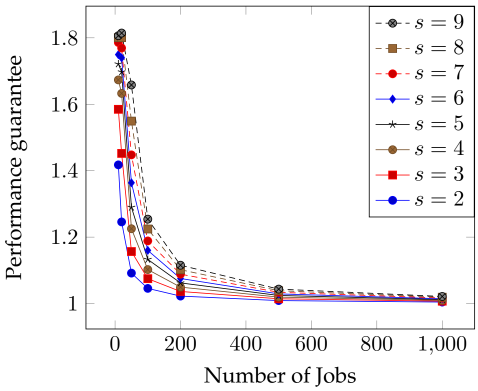


(a) $m = 2$ and varying s.

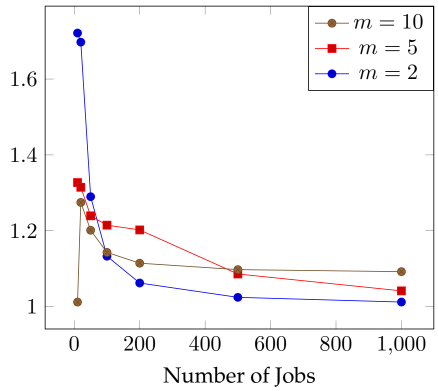


(b) $s = 5$ and varying m.

Figure 5.4.1: Performance of *sMI*-policy with exponential distributions.

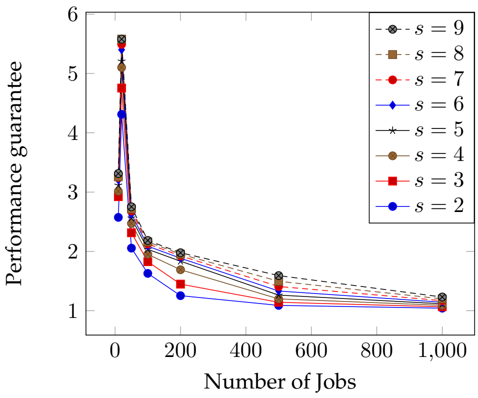


(a) $m = 2$ and varying s.

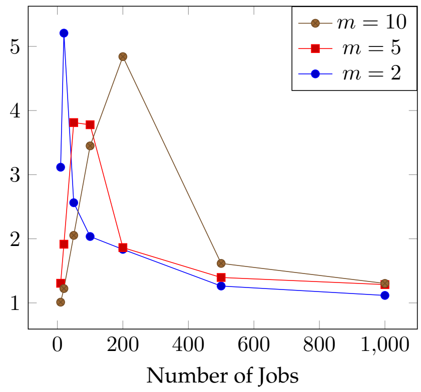


(b) $s = 5$ and varying m.

Figure 5.4.2: Performance of *sMI*-policy for uniformly distributed processing times.



(a) $m = 2$ and varying s.



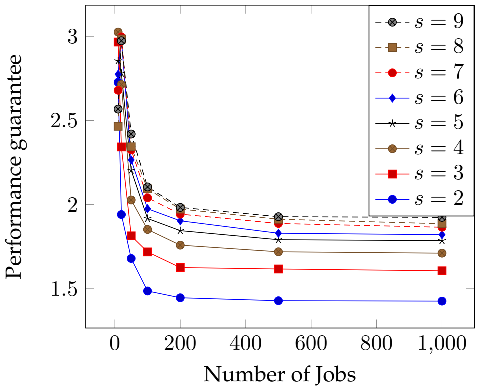
(b) $s = 5$ and varying m.

Figure 5.4.3: Performance of *sMI*-policy for log-normally distributed processing times.

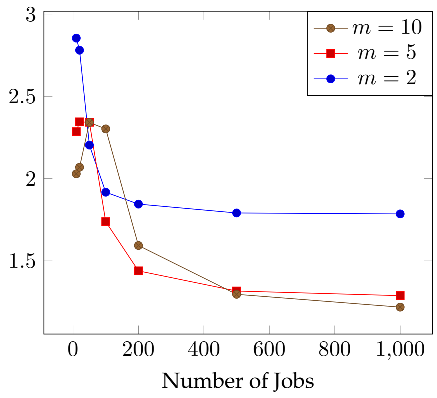
Simulation results for *QRandom*-policy. In Figures 5.4.4 - 5.4.6, the selected results of the computational study of the *QRandom*-policy are shown. Recall, that the theoretical performance guarantee for the special case with $m-1$ fast machines considered is

$$\rho = 5 + \Delta + \frac{6}{m-1}.$$

With respect to Question **P.1**, we can observe that the realized performance guarantee is significantly smaller for each group of scenarios. In fact, the realized performance guarantee is not more than 5 for each of the selected scenarios while the theoretical performance guarantee above is strictly larger than 5. Furthermore, we can observe an asymptotic behavior of the realized performance guarantee since the performance guarantee decreases as the instance size increases and remains relatively stable for very large instances. Regarding Question **P.2** we can see that Δ has an impact on the realized performance. However, this impact is not as large as expected. When comparing the results in Figures 5.4.4a and 5.4.6a the difference between the realized performance guarantees is much smaller than 9 which is the theoretical difference between settings with exponentially distributed processing times with $\Delta = 1$ and log-normally distributed processing times with $\Delta = 10$. Regarding the machine speeds, we can again observe that the performance is slightly worse for larger values of s than for smaller values. Similarly, the realized performance guarantee increases with the number of machines which is not in line with the theoretical performance guarantee. The most striking observation is again the policy's performance for smaller instances with many machines. Again this can be explained with the observations made in the computational study of the lower bounds.

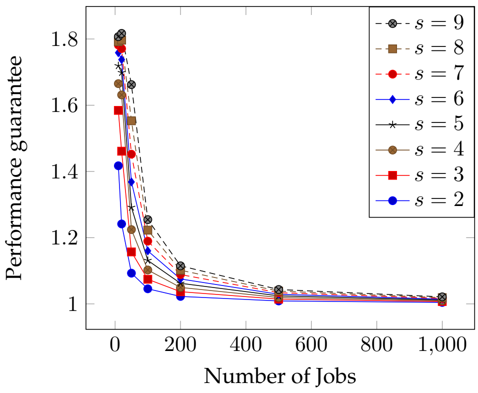


(a) $m = 2$ and varying s.

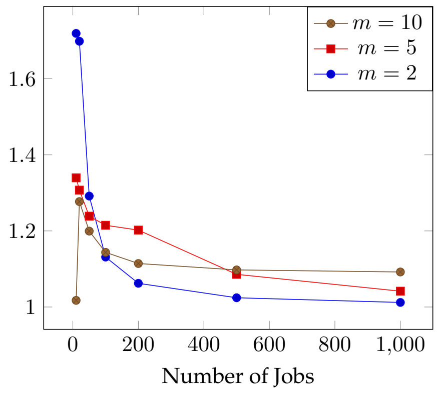


(b) $s = 5$ and varying m.

Figure 5.4.4: Performance of *QRandom*-policy for exponentially distributed processing times.

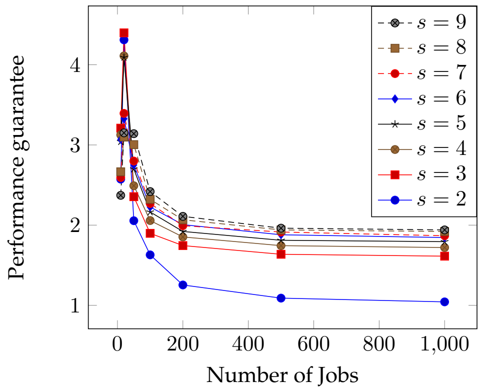


(a) $m = 2$ and varying s.

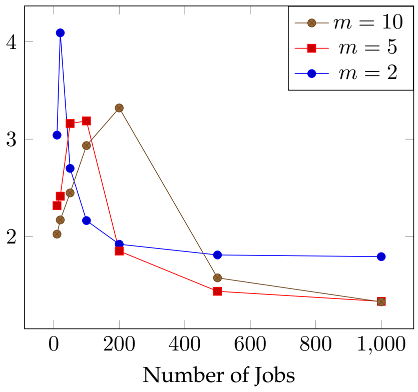


(b) $s = 5$ and varying m.

Figure 5.4.5: Performance of *QRandom*-policy for uniformly distributed processing times.



(a) $m = 2$ and varying s.



(b) $s = 5$ and varying m.

Figure 5.4.6: Performance of *QRandom*-policy for log-normally distributed processing times.

Appendix

5.A Improved lower bound for online-time model

In order to strengthen the lower bound given in Lemma 5.2, Möhring et al. [122] (see also Uetz [158]) introduce the following linear programming relaxation:

$$\begin{aligned}
\text{minimize} \quad & \sum_{j=1}^{n} w_j \quad E[C_j] \\
\text{subject to} \quad & \sum_{j \in A} E[p_j]E[C_j] \geq f(A), \quad \forall A \subseteq J \\
& \qquad\quad E[C_j] \geq \ell_j \qquad\quad \forall j \in J
\end{aligned} \tag{5.10}$$

Here, ℓ_j can be any natural lower bound on the expected processing time. In our case, $\ell_j = r_j + \frac{E[P_j]}{s}$. Furthermore, the set function $f(A)$ is defined as

$$f(A) = \frac{1}{2ms}\left(\left(\sum_{j\in A} E[p_j]\right)^2 + \sum_{j\in A} E[p_j]^2\right) - \frac{(m-1)(\Delta+1)}{2ms}\sum_{j\in A} E[p_j]^2.$$

Following along the same lines as Möhring et al. [122] (see also Uetz [158]), this linear programming relaxation is optimized for

$$E[C_j] = \frac{f^{\ell}(\{1,...,j\}) - f^{\ell}(\{1,...,j-1\})}{\frac{E[p_j]}{s}} \quad \text{for} \quad j = 1,...,n,$$

where $f^{\ell}(A)$ are auxilliary set functions defined as follows:

$$f^{\ell}(A) = \sum_{j\in A} \frac{E[p_j]}{s}\ell_j + \max_{B\subseteq A}\left\{ f(B) - \sum_{j\in B}\frac{E[p_j]}{s}\ell_j \right\} \quad A \subseteq J.$$

The second part of $f^{\ell}(A)$ is denoted as $g^{\ell}(B)$. To compute $f^{\ell}(A)$ it is necessary to solve the maximization problem of g^{ℓ} over any ground set A. To this end, Möhring et al. [122] show that the maximization problem of g_ℓ over a ground set A can be solved by first ordering the jobs in non-decreasing order of $\frac{(\Delta-1)(m-1)}{2m}E[p_j]/s + \ell_j$. If this order is assumed to given by $1, ..., |A|$, then the set maximizing $g^\ell(B)$ must be one of the nested sets, i.e., $A^* \in \{\emptyset, \{1\}, ..., \{1, ..., |A|\}\}$. Hence, this can be done in time $O(n \log n)$. Based on this result, they present a polynomial-time algorithm to find $f^\ell(1, \ldots, j)$ for every $j = 1, \ldots, n$.

Algorithm 5.3 Efficiently computing $f^\ell(\{1, ..., j\})$ for all $j = 1, ..., n$

```
**Input:**$\Delta$,$m$, $s$ and $E[P_j], \ell_j$ for all $j = 1, ..., n$
**Output:** $f^\ell(\{1, ..., j\})$ for all $j = 1, ..., n$
**Initialization:** List $L \leftarrow \emptyset$, $\sum_1 \leftarrow 0$
**for** $j = 1, ..., n$ **do**
    $\sum_1 \leftarrow \sum_1 + {}^{E[P_j]}/_{s}\ell_j$
    Insert $j$ into $L$ according to non-decreasing values $\frac{(m-1)(\Delta-1)}{2m} {}^{E[P_j]}/_{s} + \ell_j$
    Initialize $\sum_2 \leftarrow 0, g \leftarrow 0, g_{max} \leftarrow 0$
    **for** $k = 1, ..., j$ in the order given by list $L$ **do**
        $g \leftarrow g + {}^{E[P_k]}/_{s} \left( \frac{1}{m} \sum_2 + \frac{1}{m} {}^{E[P_k]}/_{s} - \ell_k - \frac{(m-1)(\Delta-1)}{2m} {}^{E[P_k]}/_{s} \right)$
        **if** $g > g_{max}$ **then**
            $g_{max} \leftarrow g$
        **end if**
        $\sum_2 \leftarrow \sum_2 + {}^{E[P_k]}/_{s}$
    **end for**
**end for**
```

5.B Complete results of computational study

n \ s	2	3	4	5	6	7	8	9
10	1.574	1.742	1.823	1.879	1.923	1.948	1.964	1.973
20	1.448	1.626	1.727	1.793	1.834	1.865	1.888	1.908
50	1.165	1.274	1.417	1.585	1.739	1.800	1.828	1.848
100	1.079	1.129	1.179	1.234	1.295	1.360	1.426	1.509
200	1.038	1.062	1.084	1.108	1.131	1.156	1.181	1.208
500	1.015	1.024	1.033	1.041	1.049	1.057	1.066	1.074
1000	1.008	1.012	1.016	1.020	1.024	1.028	1.032	1.036

Table 5.B.1: Performance of *sMI-policy* with exponentially distributed processing times and $m = 2$.

n \ s	2	3	4	5	6	7	8	9
10	1.277	1.290	1.308	1.296	1.309	1.301	1.292	1.317
20	1.550	1.579	1.590	1.616	1.622	1.626	1.614	1.624
50	1.287	1.328	1.350	1.366	1.373	1.378	1.381	1.385
100	1.199	1.242	1.263	1.277	1.286	1.292	1.296	1.301
200	1.134	1.197	1.219	1.233	1.242	1.249	1.254	1.258
500	1.051	1.085	1.122	1.161	1.202	1.224	1.229	1.233
1000	1.025	1.041	1.058	1.075	1.093	1.111	1.130	1.149

Table 5.B.2: Performance of *sMI-policy* with exponentially distributed processing times and $m = 5$.

n \ s	2	3	4	5	6	7	8	9
10	1.009	1.008	1.008	1.009	1.008	1.009	1.011	1.008
20	1.216	1.228	1.217	1.227	1.236	1.226	1.234	1.229
50	1.436	1.462	1.468	1.477	1.477	1.480	1.480	1.486
100	1.242	1.260	1.272	1.275	1.279	1.282	1.286	1.286
200	1.146	1.165	1.175	1.180	1.184	1.186	1.188	1.189
500	1.089	1.108	1.117	1.123	1.127	1.130	1.132	1.134
1000	1.055	1.089	1.099	1.105	1.109	1.112	1.114	1.116

Table 5.B.3: Performance of *sMI-policy* with exponentially distributed processing times and $m = 10$.

s n	2	3	4	5	6	7	8	9
10	1.418	1.585	1.673	1.721	1.749	1.786	1.801	1.806
20	1.246	1.452	1.633	1.697	1.740	1.769	1.801	1.814
50	1.092	1.157	1.225	1.290	1.364	1.448	1.549	1.658
100	1.046	1.075	1.102	1.133	1.160	1.189	1.224	1.254
200	1.022	1.036	1.049	1.062	1.075	1.088	1.102	1.115
500	1.009	1.014	1.019	1.024	1.029	1.034	1.039	1.043
1000	1.004	1.007	1.010	1.012	1.014	1.017	1.019	1.021

Table 5.B.4: Performance of *sMI-policy* with uniformly distributed processing times and $m = 2$.

s n	2	3	4	5	6	7	8	9
10	1.310	1.348	1.345	1.327	1.345	1.337	1.335	1.338
20	1.250	1.284	1.303	1.315	1.317	1.322	1.318	1.319
50	1.167	1.207	1.228	1.239	1.249	1.254	1.258	1.261
100	1.137	1.179	1.202	1.215	1.224	1.230	1.235	1.239
200	1.067	1.120	1.177	1.202	1.211	1.218	1.223	1.227
500	1.026	1.046	1.066	1.086	1.107	1.130	1.152	1.176
1000	1.013	1.022	1.032	1.041	1.051	1.061	1.071	1.082

Table 5.B.5: Performance of *sMI-policy* with uniformly distributed processing times and $m = 5$.

s n	2	3	4	5	6	7	8	9
10	1.019	1.015	1.016	1.012	1.014	1.015	1.016	1.018
20	1.268	1.273	1.269	1.275	1.276	1.284	1.268	1.274
50	1.168	1.186	1.194	1.201	1.201	1.202	1.203	1.204
100	1.111	1.129	1.138	1.143	1.147	1.149	1.150	1.152
200	1.081	1.100	1.109	1.114	1.118	1.121	1.123	1.124
500	1.056	1.082	1.092	1.097	1.101	1.104	1.106	1.108
1000	1.027	1.049	1.071	1.092	1.096	1.099	1.101	1.103

Table 5.B.6: Performance of *sMI-policy* with uniformly distributed processing times and $m = 10$.

s / n	2	3	4	5	6	7	8	9
10	2.574	2.923	3.014	3.116	3.204	3.245	3.257	3.313
20	4.310	4.754	5.101	5.207	5.400	5.491	5.578	5.578
50	2.055	2.314	2.472	2.562	2.630	2.683	2.716	2.752
100	1.629	1.827	1.951	2.036	2.088	2.127	2.161	2.185
200	1.254	1.449	1.691	1.835	1.885	1.925	1.954	1.978
500	1.090	1.142	1.200	1.263	1.332	1.409	1.497	1.592
1000	1.043	1.067	1.092	1.117	1.143	1.170	1.200	1.231

Table 5.B.7: Performance of *sMI-policy* with log-normally distributed processing times and $m = 2$.

s / n	2	3	4	5	6	7	8	9
10	1.300	1.297	1.321	1.306	1.309	1.326	1.301	1.287
20	1.837	1.874	1.921	1.915	1.915	1.909	1.921	1.950
50	3.525	3.687	3.818	3.812	3.764	3.840	3.810	3.891
100	3.574	3.795	3.821	3.779	3.885	3.986	3.866	3.960
200	1.738	1.807	1.843	1.863	1.873	1.887	1.891	1.892
500	1.305	1.353	1.380	1.397	1.408	1.416	1.421	1.426
1000	1.190	1.247	1.271	1.286	1.296	1.304	1.309	1.314

Table 5.B.8: Performance of *sMI-policy* with log-normally distributed processing times and $m = 5$.

s / n	2	3	4	5	6	7	8	9
10	1.009	1.010	1.009	1.012	1.009	1.010	1.009	1.010
20	1.231	1.231	1.232	1.224	1.226	1.226	1.239	1.234
50	2.003	2.020	2.023	2.053	2.058	2.040	2.034	2.066
100	3.322	3.443	3.439	3.449	3.475	3.475	3.495	3.497
200	4.779	4.829	4.919	4.840	4.881	4.853	4.926	4.832
500	1.565	1.590	1.607	1.616	1.622	1.628	1.627	1.632
1000	1.263	1.286	1.297	1.304	1.308	1.313	1.314	1.317

Table 5.B.9: Performance of *sMI-policy* with log-normally distributed processing times and $m = 10$.

n \ s	2	3	4	5	6	7	8	9
10	2.727	2.966	3.025	2.854	2.775	2.679	2.465	2.568
20	1.941	2.343	2.714	2.780	2.981	2.997	2.984	2.973
50	1.680	1.814	2.027	2.204	2.265	2.325	2.343	2.419
100	1.487	1.719	1.852	1.917	1.975	2.040	2.093	2.105
200	1.446	1.626	1.759	1.845	1.904	1.943	1.973	1.983
500	1.429	1.617	1.720	1.791	1.830	1.888	1.913	1.928
1000	1.426	1.606	1.711	1.786	1.821	1.865	1.886	1.924

Table 5.B.10: Performance of *QRandom*-policy with exponentially distributed processing times and $m = 2$.

n \ s	2	3	4	5	6	7	8	9
10	2.382	2.365	2.222	2.285	2.222	2.122	2.174	2.118
20	2.781	2.678	2.375	2.345	2.371	2.247	2.146	2.213
50	1.870	1.993	2.253	2.342	2.546	2.645	2.553	2.546
100	1.482	1.556	1.673	1.738	1.816	1.875	1.928	2.030
200	1.303	1.390	1.418	1.439	1.488	1.523	1.570	1.564
500	1.214	1.269	1.305	1.317	1.347	1.351	1.360	1.371
1000	1.193	1.241	1.268	1.289	1.299	1.316	1.323	1.317

Table 5.B.11: Performance of *QRandom*-policy with exponentially distributed processing times and $m = 5$.

n \ s	2	3	4	5	6	7	8	9
10	2.277	2.132	2.201	2.029	2.017	2.164	1.965	2.059
20	2.397	2.232	2.155	2.069	2.091	2.059	2.102	2.151
50	2.646	2.715	2.467	2.341	2.332	2.179	2.173	2.073
100	1.757	1.945	2.117	2.302	2.464	2.553	2.446	2.360
200	1.410	1.479	1.561	1.594	1.681	1.763	1.836	1.891
500	1.218	1.253	1.275	1.297	1.328	1.340	1.364	1.378
1000	1.168	1.190	1.201	1.219	1.233	1.233	1.240	1.259

Table 5.B.12: Performance of *QRandom*-policy with exponentially distributed processing times and $m = 10$.

n \ s	2	3	4	5	6	7	8	9
10	1.417	1.584	1.665	1.719	1.759	1.782	1.792	1.807
20	1.241	1.461	1.631	1.699	1.738	1.771	1.796	1.817
50	1.093	1.157	1.224	1.292	1.368	1.452	1.553	1.662
100	1.046	1.074	1.102	1.131	1.160	1.189	1.223	1.254
200	1.023	1.037	1.050	1.062	1.075	1.089	1.102	1.115
500	1.009	1.014	1.019	1.024	1.029	1.034	1.039	1.043
1000	1.004	1.007	1.010	1.012	1.014	1.017	1.019	1.021

Table 5.B.13: Performance of *QRandom*-policy with uniformly distributed processing times and $m = 2$.

n \ s	2	3	4	5	6	7	8	9
10	1.324	1.334	1.320	1.340	1.348	1.354	1.353	1.334
20	1.251	1.287	1.302	1.307	1.314	1.316	1.318	1.319
50	1.166	1.207	1.228	1.239	1.249	1.254	1.259	1.262
100	1.136	1.180	1.201	1.215	1.223	1.230	1.235	1.239
200	1.067	1.119	1.177	1.202	1.211	1.218	1.223	1.227
500	1.026	1.046	1.066	1.086	1.107	1.130	1.152	1.176
1000	1.013	1.022	1.032	1.042	1.051	1.061	1.071	1.082

Table 5.B.14: Performance of *QRandom*-policy with uniformly distributed processing times and $m = 5$.

n \ s	2	3	4	5	6	7	8	9
10	1.018	1.015	1.016	1.017	1.014	1.015	1.016	1.015
20	1.264	1.275	1.277	1.277	1.277	1.273	1.283	1.274
50	1.169	1.186	1.194	1.200	1.202	1.202	1.205	1.205
100	1.111	1.129	1.138	1.144	1.146	1.148	1.151	1.152
200	1.082	1.099	1.109	1.114	1.118	1.120	1.123	1.124
500	1.055	1.082	1.092	1.098	1.101	1.104	1.106	1.108
1000	1.027	1.049	1.071	1.092	1.096	1.099	1.101	1.103

Table 5.B.15: Performance of *QRandom*-policy with uniformly distributed processing times and $m = 10$.

n \ s	2	3	4	5	6	7	8	9
10	3.287	3.209	3.126	3.041	3.069	2.594	2.665	2.372
20	4.450	4.398	4.110	4.092	3.326	3.394	3.098	3.151
50	1.983	2.355	2.491	2.700	2.750	2.799	3.004	3.141
100	1.652	1.898	2.057	2.164	2.226	2.270	2.322	2.419
200	1.509	1.745	1.853	1.921	2.006	1.993	2.067	2.109
500	1.462	1.637	1.744	1.811	1.880	1.914	1.940	1.962
1000	1.432	1.613	1.721	1.794	1.845	1.868	1.921	1.941

Table 5.B.16: Performance of *QRandom*-policy with log-normally distributed processing times and $m = 2$.

n \ s	2	3	4	5	6	7	8	9
10	2.470	2.376	2.269	2.318	2.208	2.309	2.221	2.072
20	2.870	2.723	2.552	2.414	2.298	2.238	2.199	2.218
50	4.278	3.664	3.413	3.161	2.928	2.730	2.617	2.468
100	2.712	2.853	3.080	3.187	3.267	3.303	3.398	3.197
200	1.634	1.744	1.766	1.851	1.868	1.923	1.939	2.025
500	1.327	1.378	1.409	1.438	1.459	1.465	1.476	1.494
1000	1.251	1.301	1.323	1.335	1.364	1.352	1.363	1.372

Table 5.B.17: Performance of *QRandom*-policy with log-normally distributed processing times and $m = 5$.

n \ s	2	3	4	5	6	7	8	9
10	2.266	2.196	2.059	2.025	2.003	2.014	2.047	2.052
20	2.388	2.272	2.137	2.171	2.101	2.037	1.978	2.068
50	2.980	2.705	2.505	2.448	2.288	2.193	2.211	2.154
100	4.178	3.528	3.249	2.934	2.731	2.623	2.464	2.405
200	2.894	3.159	3.220	3.320	3.535	3.394	3.337	3.130
500	1.488	1.533	1.563	1.576	1.626	1.636	1.659	1.720
1000	1.268	1.293	1.321	1.330	1.341	1.349	1.369	1.376

Table 5.B.18: Performance of *QRandom*-policy with log-normally distributed processing times and $m = 10$.

Chapter 6

Optimizing fuel consumption under uncertainty for a single vessel in inland waterways

6.1 Introduction

The transportation and logistics industry is a key driver in reaching sustainability accounting for nearly a quarter of the EU's greenhouse gas emissions [51]. Towards the goal of zero-emission transportation and a reduction of at least 90% by 2050 [52], the European Commission has identified various areas for action and change [51, 52]. Among these is the increase of the efficiency of transportation systems as well as the move towards low-emission transportation modes. Inland waterways have been shown to be one of the most sustainable and CO_2-efficient transport modes per tonne of goods carried [53] and are a natural tool towards achieving the aforementioned goals.

This chapter has been adapted from: M. Buchem, J. A. P. Golak, and A. Grigoriev. "Vessel velocity decisions in inland waterway transportation under uncertainty". In: *European Journal of Operational Research* 296 (2022), pp. 669–678

Besides its advantages, inland waterway transportation faces many challenges such as the presence of locks forming a crucial bottleneck for the efficiency and management. Locks are necessary for inland waterways to maintain sufficient water levels for navigation and moving vessels between river segments of different water levels. In recent years, the possible role of inland waterway transportation and the challenges faced have sparked a large interest in the theoretical optimization of lock operations from a scheduling point-of-view.

Problem Description and Preliminaries. In this chapter, we take the perspective of a single vessel facing uncertainty at a lock with the goal of optimizing the vessel's fuel consumption by deciding on the vessel's velocity and its arrival time at the lock. We consider a waterway split in two river segments, the upstream and downstream segment. The waterway contains a single lock with a single chamber of unit capacity, i.e., the lock can serve one vessel at a time. The vessel we consider, vessel a, is approaching the lock from the upstream segment while a second vessel, vessel b, is approaching the lock from the downstream segment. The vessel for which we can control the velocity is vessel a, whereas vessel b is outside of our control. A visualization of the setting is given in Figure 6.1.1.

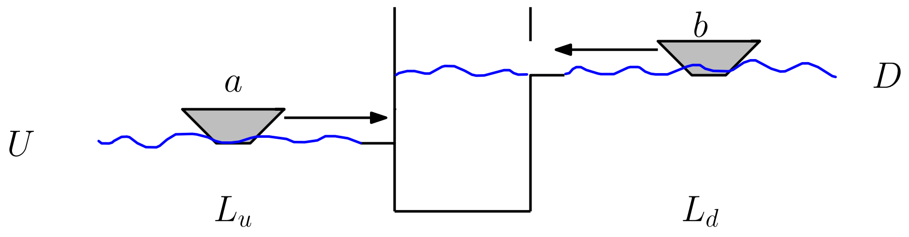


Figure 6.1.1: Visualization of problem setting.

An instance of the problem is defined as follows. The length of the waterway is defined by the lengths of the two segments. Let L_u denote the length of the upstream segment and, similarly, L_d denote the length of the downstream segment. Furthermore, the lock is characterized by a deterministic lockage time. This is the time it takes the lock to move a vessel from one of the river segments excluding the vessel-dependent docking and undocking time. We denote the lockage time by P. Moreover, we assume that the lock operates based on a first-come-first-serve basis with ties being broken in favor of vessel a. Vessel a is characterized by its deadline D at which it must have reached the end of the downstream segment and its total docking and undocking time η_a. For vessel a, the docking and undocking time is deterministic and, thus, the total time it takes vessel a to move from the upstream to the downstream segment is given by $\tau_a = P + \eta_a$. Vessel b is characterized by its fixed and known arrival time at the lock, denoted by r, and its total docking and undocking time η_b. In contrast to the η_a, η_b is not deterministic and subject to uncertainty as the skipper of vessel a is not aware of the exact nature of vessel b. Therefore, η_b is given as a random variable and the total processing time of vessel b in the lock is defined as the random variable $\tau_b = P + \eta_b$ and $\tau_b = c_i$ with probability p_i for $i = 1, \dots, n$. Finally, for each possible realization of τ_b, vessel a is informed at time $r + c_i$ if vessel b has exited the lock or not. The instance is fully defined by the tuple $\mathcal{I} = (\eta_a, \eta_b, P, r, D, L_d, L_u)$. In this chapter, we focus on the setting where the probability distribution underlying η_b is discrete.

The goal is to minimize the total expected fuel consumption needed by vessel a to cross the complete waterway. The real fuel consumption per time unit can be approximated as a cubic function of the vessel's velocity given vessel specifications ρ [55, 85, 124, 126]. In accordance with the literature, we define the fuel consumption per time unit as $F^t(v) = \rho \cdot v^3$. In order to travel a distance d at constant velocity v takes time $t = d/v$. Therefore, the total fuel consumption required for travelling distance d at velocity v is $F(v, d) = F^t(v)t = d \cdot \rho \cdot v^2$. Since $v \geq 0$ the fuel consumption is convex. Furthermore, we make the following assumptions:

- The velocity of vessel a can be fully controlled at any point in time and can be changed instantly.

- Vessel a's velocity is subject to minimum and maximum velocities.
- The lock is positioned to the vessel that arrives first, i.e., arriving first at the lock implies the possibility of immediate docking into the lock.
- The ship specifications ρ are equal to 1 such that the fuel consumption functions are $F^t(v) = v^3$ and $F(v, d) = dv^2$ expressed in unit of time and distance, respectively.
- Vessels a and b are said to be in a conflict situation meaning that at the optimal arrival time of vessel a when ignoring the presence of vessel b there is a positive probability that the lock is still processing vessel b.

Remark 6.1. Since the total fuel consumption is convex for non-negative velocities, once vessel a has left the lock at time t the fuel consumption on the downstream segment is optimized by a constant velocity

$$\frac{L_d}{D - t}$$

yielding a total fuel consumption on the downstream segment of

$$\frac{L_d^3}{(D - t)^2}.$$

A solution to the problem is defined by a *velocity policy* for vessel a. For each point in time t, a velocity policy must choose a velocity for vessel a. Given the previous remark, we restrict ourselves to velocity policies choosing a velocity for any moment in time on the upstream segment. These velocity choices then imply a specific constant velocity on the downstream segment. We require a velocity policy to be non-anticipatory such that it must make its decisions only based on the knowledge available up to time t and all the a-priori distributional information.

This information includes the availability of the lock, denoted by $\omega(t)$:

$$\omega(t) = \begin{cases} -1, & \text{if vessel } b \text{ has not left the lock at time } t; \\ t - t_b, & \text{if vessel } b \text{ has exited the lock at time } t_b \leq t. \end{cases}$$

Recall that we only consider cases in which vessels a and b are in a conflict with each other. Therefore, $\omega(t) = -1$ for $t < r$.

Formally, a velocity policy is a decision rule which for any decision moment t and any realization of $\omega(t)$ defines a velocity $v(t, \omega(t))$ for vessel a. As mentioned before, a velocity policy may only take into account the full knowledge accumulated up to point t. Furthermore, a velocity policy must satisfy two conditions to be feasible. The first condition is that for all t and $\omega(t)$, $v(t, \omega(t))$, must satisfy possible imposed minimum and maximum velocities denoted by $v_{\min} \geq 0$ and $v_{\max} \geq v_{\min}$, respectively. The second condition is that the velocity policy needs to ensure that vessel a traverses the full upstream segment and that for any choice of velocities and realizations of τ_b, enough time remains for vessel a to cross the downstream segment. To define this condition, we need to introduce additional notions.

Let $T(v)$ denote the random arrival time of vessel a at the lock under velocity policy $v(\cdot)$. The waiting time at the lock for vessel a, denoted by $q(T(v))$, is defined as

$$q(T(v)) = \begin{cases} \max\left\{r + \tau_b - T(v),\ 0\right\}, & \text{if } r < T(v) < r + c_n; \\ 0, & \text{otherwise.} \end{cases}$$

The waiting time is a random variable as well. For $T(v) = r_a > r + c_i$ for some $i = 1, \ldots, n$ we have that $q(r_a) = r + c_j - r_a$ with probability p_j for $j > i$ and $q(r_a) = 0$ with probability $\sum_{k=1}^{i} p_k$.

Based on these random variables, we can formulate the second condition as follows:

$$\int_0^{T(v)} v(t, \omega(t))\mathrm{d}t = L_u,$$

$$T(v) + q(T(v)) + \tau_a \leq D - \frac{L_d}{v_{\max}}.$$

The first constraint ensures that vessel a reaches the lock at time $T(v)$ and the second constraint ensures that enough time remains to cross the downward segment while satisfying the maximum velocity.

Finally, we define the total expected fuel consumption of a velocity policy v. Since the fuel consumption inside the lock is 0, the total fuel consumption is split in two parts, the consumption on the upstream and downstream segments as follows:

$$E\left[\int_0^{T(v)} v(t,\omega(t))^3 \mathrm{d}t + \Big(\frac{L_d}{D - T(v) - q(T(v)) - \tau_a}\Big)^2 L_d\right].$$

Thus, finding an optimal velocity policy can be formulated as:

$$\min_{v(.)} E\left[\int_0^{T(v)} v(t,\omega(t))^3 \mathrm{d}t + \Big(\frac{L_d}{D - T(v) - q(T(v)) - \tau_a}\Big)^2 L_d\right] \tag{6.1}$$

$$s.t.$$

$$\int_0^{T(v)} v(t,\omega(t))\mathrm{d}t = L_u \tag{6.2}$$

$$T(v) + q(T(v)) + \tau_a \leq D - \frac{L_d}{v_{\max}} \tag{6.3}$$

$$v_{min} \leq v(t,\omega(t)) \leq v_{max} \quad \forall t \geq 0 \text{ and } \omega(t) \tag{6.4}$$

An illustration of the problem is given in the following example.

EXAMPLE 6.1: ILLUSTRATIVE EXAMPLE

Consider an instance defined by the following parameters:

- *Waterway:* $L_u = L_d = 1$,
- *Vessel a:* $D = 6$ and $\tau_a = 1$ and
- *Vessel b:* $r = 0$ and $\tau_b \sim unif\{1, 3\}$.

The presence of vessel b leads to a conflict situation since the optimal arrival time at the lock of vessel a ignoring the presence of vessel b is $t^* = 2.5$.
A natural velocity policy is to approach the lock with velocity $\frac{1}{2.5}$ until time 2 and then update the velocity depending on the lock's availability: if the lock is available at time 2 the velocity remains the same, if the lock is not available the velocity is changed such that the vessel arrives at the lock at time 3. This velocity policy yields a total expected fuel consumption of approximately 0.342. This policy is visualized in Figure 6.1.2, where each box shows the position at the time of this node and the velocity chosen at the previous time point. In contrast to this dynamic approach, not taking into account the uncertainty and choosing a constant velocity with arrival time 2.5 leads to a total expected fuel consumption of 0.35. Hence, even a simple dynamic velocity policy leads to possible fuel savings of approximately 2.3%.

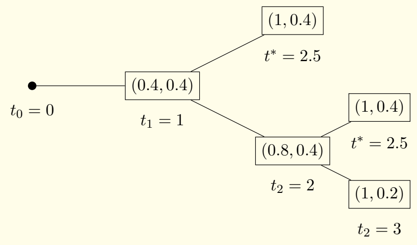


Figure 6.1.2: Visualization of simple dynamic solution to Example 6.1.

Related work. The optimization of lock operations in inland waterways entails three interrelated subproblems: the assignments of vessels to lock chambers, the placement of vessels within a single chamber and the scheduling of lockage operations [164]. An extensive overview of the results for these subproblems can be found in [125, 162]. We turn our attention towards the third subproblem.

Hermans [77] considers a lock scheduling problem for a single lock and a single chamber with bidirectional traffic on a waterway and presents a polynomial time dynamic program to find a feasible schedule subject to vessel deadlines to cross the waterway. Passchyn et al. [129] present a polynomial time algorithm to minimize total waiting time of vessels with known arrival times at the lock for a lock with a single chamber. In [128], this problem is extended to a single lock with multiple chambers and schedules with zero waiting time at the locks are desired. A linear time algorithm for a lock with two chambers as well as a dynamic program for an arbitrary number of chambers are presented. Pinson and Spieksma [132] consider the lock scheduling problem in the online setting and allow for possible lookahead as to find a feasible schedule with zero total waiting time. Verstichel et al. [163] present a mathematical programming formulation for the lock scheduling problem considering multiple chambers and introduce heuristic methods. The above results focus on scheduling a single lock (with multiple chambers). In many inland waterways, however, a sequence of lock is given. Passchyn et al. [127] introduce a mathematical programming formulation for scheduling multiple locks in a sequence. Passchyn and Spieksma [130] introduce a batch scheduling problem on identical parallel machines in a sequence with the aim to minimize total completion time. They show that the problem is strongly $\mathcal{NP}$-hard even for two identical parallel machines when all vessels travel in the same direction. Prandstetter et al. [133] present a heuristic and search algorithm to minimize total travel times and consider the real world example of the Danube river. The specific objectives of minimizing fuel consumption or total emissions have become more present in the area of lock scheduling in the recent years. Passchyn et al. [126] introduce a mathematical programming formulation of the lock scheduling problem with the goal to minimize total emissions and investigate the trade-off between emissions and travel times. Defryn et al. [43] view the problem from a game-theoretic point-of-view where each vessel chooses its individual velocity and arrival time at the lock and the goal is to minimize total fuel consumption.

While uncertainty has not yet been considered in inland waterway management, multiple models with uncertainty are known for maritime shipping. In maritime shipping, container vessels must visit multiple ports along a route for loading and unloading cargo. These loading and unloading operations can be subject to random fluctuations. Brouer et al. [20] introduce the vessel schedule recovery problem in which random service times at ports lead to disruptions. These disruptions lead to possible changes in the remaining schedule to satisfy a certain service level. Wang et al. [167] present a non-linear stochastic programming formulation to find a robust schedule for maritime shipping routes when late arrivals are not allowed. Aydin et al. [7] consider a dynamic approach to optimize the velocity of a vessel in liner shipping to minimize total fuel consumption. Andersen et al. [4] present a model for optimizing traffic in the Kiel Canal subject to uncertain arrival times of vessels at the entrance of the canal. This generalizes earlier work by Lübbecke et al. [115] in the deterministic setting.

Outline. The remainder of this chapter is structured as follows. In Section 6.2, we first derive two structural properties of optimal velocity policies. These properties allow us to derive a recursive formulation leading to a dynamic program for finding a close-to-optimal velocity policy. In Section 6.3, we introduce a simple heuristic velocity policy based on the idea of a fixed arrival time at the lock for vessel a. In Section 6.4, we conduct a computational study to analyse and compare the quality of the introduced solution methods based on the efficiency as well as simplicity of implementation.

6.2 Optimal and near-optimal dynamic velocity policies

In this section, we first derive a recursive formulation to find an optimal velocity policy based on two observations. Then, we use this recursive formulation to develop a dynamic program for computing a near-optimal velocity policy.

First, observe that as soon as the lock becomes available for vessel a, it is optimal to travel the remaining distance of the upstream segment with a constant velocity. This is due to the convexity of fuel consumption for non-negative velocities. Additionally, no waiting time is necessary once the lock is available.

Observation 6.2. *Let v^* be an optimal velocity policy. Then, for any time t at which vessel b has already left the lock, the following holds:*

$$v^*(t, \omega(t)) = \begin{cases} v^*(t - \omega(t), 0), & \textit{if vessel } a \textit{ has not reached the lock at time } t; \\ 0, & \textit{if vessel } a \textit{ has reached the lock.} \end{cases} \tag{6.5}$$

This property allows us to reformulate the total expected fuel consumption. Consider a point in time t and let $X(t)$ define the position of vessel a at time t. Then, for the remaining distance of the upstream segment the optimal final fuel consumption is defined by

$$F(X(t)) = \min_{v_{min} \le v \le v_{max}} v^2(L_u - X) + \frac{L_d^3}{(D - (t + (L_u - X)/v + \tau_a))^2}. \tag{6.6}$$

Note that the optimal constant velocity for a given $X(t)$ can be found using first order conditions. Let $\tilde{p}_t = 1 - \sum_{i:r+c_i \le t} p_i$ be the probability that the lock is not available at time t and p_t be the probability that the lock is available at time t. Then, due to Observation 6.2, the total expected fuel consumption can be reformulated as

$$\begin{aligned}
&E\left[\int_0^{T(v)} v(t,\omega(t))^3 \mathrm{d}t + \Big(\frac{L_d}{D - T(v) - q(T(v)) - \tau}\Big)^2 L_d\right] \\
&= \int_0^{T(v)} \tilde{p}_t v(t,-1)^3 \mathrm{d}t + \int_0^{T(v)} p_t F(X(t)) \mathrm{d}t.
\end{aligned} \tag{6.7}$$

Here, the first part corresponds to the expected fuel consumption before the lock finishes processing vessel b and the second part represents the fuel consumption after the lock finishes processing vessel b. This in combination with the convexity of fuel consumption implies that velocity changes are not optimal unless new information about the status of the lock is revealed.

Next, we use the fact that the processing time distribution for vessel b is discrete to restrict the moments in time at which velocity changes occur. As the fuel consumption is convex, constant velocity is desired whenever possible [85]. In the time intervals between realizations of τ_b, no new information about the lock is revealed and, therefore, any velocity changes in such an interval can be compensated by using a constant (average) velocity.

Observation 6.3. *Let v^* be an optimal velocity policy. Then, velocity changes only occur at time $t \in \{0, r + c_1, \dots, r + c_n\}$.*

This property of an optimal velocity policy allows us to further reformulate the total expected fuel consumption. To clarify notation, we reinterpret the decisions of a velocity policy. At decision moment i (corresponding to time $r + c_i$), the velocity policy chooses a location $X(r + c_{i+1})$ at which the vessel will be located at decision moment $i+1$ if the lock is not available at time $r + c_i$. If, however, the lock is available a final velocity $v(i)$ is chosen according to (6.6). Since this final cost can be considered given for any decision moment and possible location, a velocity policy is uniquely defined by $X(r + c_i)$ for $i = 1, \dots, n$. This implies that the vessel travels with velocity $X(r+c_1)/(r+c_1)$ from time

0 to time $r + c_1$ and with velocity $\frac{X(r+c_{i+1})-X(r+c_i)}{(c_{i+1}-c_i)}$ from time $r + c_i$ to time $r + c_{i+1}$. Thus, Equation (6.7) can be reformulated as follows

$$\begin{aligned}
&\int_0^{T(v)} \tilde{p}_t v(t,-1)^3 \mathrm{d}t + \int_0^{T(v)} p_t F(X(t))\mathrm{d}t \\
=&\frac{X(r+c_1)^3}{(r+c_1)^2} \\
&+\sum_{i=1}^{n-1}\Big(\tilde{p}_{r+c_i}\cdot\frac{(X(r+c_{i+1})-X(r+c_i))^3}{(c_{i+1}-c_i)^2}+p_i\cdot F(X(r+c_i))\Big) \\
&+p_n\cdot F(X(r+c_n))
\end{aligned} \tag{6.8}$$

Based on this reformulation, we show that an optimal velocity policy can be found via a recursive method of minimizing the expected total fuel consumption after time $r + c_i$ for all $i = n, \dots, 1$ for all possible locations of the vessel at any decision moment and conditionally on the fact that we must start at position 0 of the upstream segment. To this end, we split the expected fuel consumption after time $r + c_i$ into two components. First, the expected fuel consumption if the lock is still processing at time $r+c_i$. Secondly, the expected total fuel consumption if the lock becomes available at time $r + c_i$. Let $g_i(X(r + c_i))$ be the minimum expected total fuel consumption after time $r + c_i$ given that at that time the vessel is located in position $X(r + c_i)$ if the lock has not been available up to this point in time. Furthermore, define by q_i the probability that the lock finishes moving vessel b after c_i time steps given that it has not finished before. Formally, $q_i = P(\tau_b = r + c_i|\tau_b \geq r+c_i)$. Note that for time $r+c_n$ the probability that the lock is still in the process of moving vessel b is zero and, therefore, the fuel consumption only consists of the second component. Consequently, we obtain the following formulation for the total expected fuel consumption:

$$g_n(X(r+c_n)) = p_n \cdot F(X(r+c_n)). \tag{6.9}$$

For $1 \leq i \leq n-1$ we have:

$$g_i(X(r+c_i)) = (1-q_i) \cdot \min_{X \in A_i(X(r+c_i))} \left(\frac{(X - X(r+c_i))^3}{(c_{i+1}-c_i)^2} + g_{i+1}(X) \right) + q_i \cdot F(X(r+c_i)). \tag{6.10}$$

Here, $A_i(\cdot)$ denotes the feasible set of choices for the velocity policy. More precisely, $A_i(X(r+c_i)) = \{X | X(r+c_i) \leq X \leq L_u$ and $v_{min} \leq \frac{X - X(r+c_i)}{c_{i+1}-c_i} \leq v_{\max}\}$ contains all possible positions at time $r + c_{i+1}$ given that at time $r+c_i$ the vessel is located at position $X(r+c_i)$. As vessel a starts in position 0, we obtain

$$g_0 = \min_{X \in A_0} \left\{ \frac{X^3}{(r+c_1)^2} + g_1(X) \right\}, \tag{6.11}$$

with $A(0) = \{X | 0 \leq X \leq L_u$ and $v_{min} \leq \frac{X}{r+c_1} \leq v_{\max}\}$.

This recursive expression allows us to find a near-optimal velocity policy via dynamic programming based on discretization of $A_i(\cdot)$.

Algorithm 6.1 Dynamic program to find near-optimal velocity policy.

```
 1: Input: $\mathcal{I}$, $\varepsilon$
 2: Output: Minimal fuel consumption of traversing the waterway.
    Phase 1 – Preliminaries
 3: Let $K := \lceil L_u/\varepsilon \rceil$
 4: $X_j = j\frac{L_u}{K}$ for all $j = 0\dots, K$
 5: $V$ is $n \times K$ matrix
 6: Compute $F(X_j(r+c_i))$ for all $j = 0, \dots, K$ and all $i = 1, \dots, n$
 7: $q_i = P(\tau_b = r + c_i | \tau_b \geq r + c_i)$
    Phase 2 – Initialization
 8: $V(n, X_j) = p_n \cdot F(X_j(r + c_n))$ for all $j = 0, \dots, K$
    Phase 3 – Recursion
 9: for $i = n-1, \dots, 2$ do
10:     for $j = 1, \dots, K$ do
11:         $V(i, X_j) = (1 - q_i) \min_{X_{j'} \in A} \left\{ \frac{(X_{j'} - X_j)^3}{(c_{i+1} - c_i)^3} + V(i+1, X_{j'}) \right\} + q_i \cdot F(X_j(r + c_i))$
12:         $A_i(X(r+c_i)) = \left\{ X | X(r+c_i) \leq X \leq L_u, v_{min} \leq \frac{X - X(r+c_i)}{c_{i+1} - c_i} \leq v_{\max} \right\}$
13:     end for
14: end for
15: Return: $\min_{X_j \in A(0)} \left\{ \frac{X_j^3}{(r+c_1)^2} + V(2, X_j) \right\}$
```

Theorem 6.4. *Consider an instance $\mathcal{I}$ and let $\mathrm{OPT}(\mathcal{I})$ be an optimal velocity policy. Then, for any $\epsilon > 0$, Algorithm 6.1 finds a velocity policy v with total expected fuel consumption $f^v(\mathcal{I})$ such that*

$$f^v(\mathcal{I}) \leq (1 + \varepsilon) f(\mathrm{OPT}(\mathcal{I})),$$

in time $O(nL_u^2/\epsilon^2)$.

Proof. The running time of the algorithm depends on the running time to compute the fixed final cost and the initialization and recursion. Let C denote the time it takes to compute the fixed final cost. Then, the initialization takes $O(CL_u/\varepsilon)$ time as $K := \lceil L_u/\varepsilon \rceil$. Furthermore, the recursion computes the minimum over all X'_j, where $j' = 1, \dots, K$ for all $i = 1, \dots, n$ and X_j, where $j = 1, \dots, K$. This amounts to a running time of $O(Cn(L_u/\varepsilon)^2$. Hence, the total running time of the dynamic program is in $O(CL_u/\varepsilon + Cn(L_u/\varepsilon)^2) = O(n(L_u/\varepsilon)^2)$.

The approximation guarantee of the dynamic program follows from convexity of the fuel consumption. Based on discretizing the possible positions of vessel a, we have that for any $i \in \{0, \dots, n-1\}$ the position chosen by the dynamic program X deviates from the optimal position X^* by at most ε. Hence, $|V(X,i) - V(X^*,i)| \leq \varepsilon V(X^*,i)$. From this it follows that the total expected fuel consumption of the velocity policy found by the dynamic program is at most $1 + \varepsilon$ as large as the total expected fuel consumption of an optimal velocity policy. $\square$

The following illustrates a near-optimal velocity policy for Example 6.1.

Example 6.2: Near-Optimal Velocity Policy

Recall the instance described in Example 6.1 given by the following parameters:

- *Waterway:* $L_u = L_d = 1$,
- *Vessel a:* $D = 6$ and $\tau_a = 1$ and
- *Vessel b:* $r = 0$ and $\tau_b \sim unif\{1, 3\}$.

Previously, we considered a natural dynamic velocity policy. Using Algorithm 6.1 we can find a near-optimal velocity policy. This is depicted in Figure 6.2.1. This optimal dynamic policy yields a total fuel consumption of approximately 0.3377 implying fuel savings of 3.5% compared to the fuel consumption when the presence of vessel b is ignored and 1.24% compared to the simple policy.

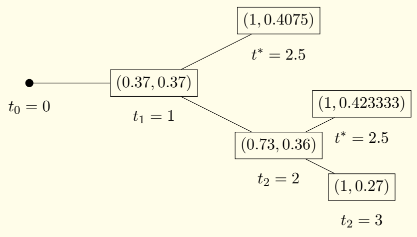


Figure 6.2.1: Visualization of optimal dynamic solution to Example 6.1.

6.3 Fixed arrival velocity policies

In this section, we introduce a class of simple policies called *fixed arrival* policies which choose a constant velocity on both river segments.

Definition 6.5. *A velocity policy v is called a fixed arrival policy if for some $v_{\min} \leq \hat{v} \leq v_{\max}$ we have*

$$v(t, \omega(t)) = \hat{v} \text{ for all } t \text{ and } \omega(t). \tag{6.12}$$

The deterministic arrival time at the lock is

$$T(v) = \frac{L_u}{v}.$$

Note that for a fixed arrival policy to be feasible there must still be time to cross the downstream segment, i.e., $T(v) + q(T(v)) + \tau_a \leq D - L_d/v_{\max}$, where $q(T(v))$ is again the waiting time at the lock for vessel a defined in the previous section.

The total expected fuel consumption of a fixed arrival policy is

$$\frac{L_u^3}{T(\hat{v})^2} + \sum_{i=0}^{n} p_i \left[\left(\frac{L_d}{D - (\max\{r_a, r + c_i\} + \tau_a)} \right)^2 L_d \right]. \tag{6.13}$$

The first part is the deterministic fuel consumption on the upstream segment and the second part is the expected fuel consumption on the downstream segment.

An *optimal fixed arrival* policy can be obtained by solving the following mathematical program.

$$\min_{\hat{v}} \frac{L_u^3}{T(\hat{v})^2} + \sum_{i=0}^{n} p_i \left[\left(\frac{L_d}{D - (\max\{T(\hat{v}), r + c_i\} + \tau_a)} \right)^2 L_d \right] \tag{6.14}$$

$$s.t.$$

$$L_u/v_{max} \leq T(\hat{v}) < \min\left\{D - \tau_a, L_u/v_{min}\right\}. \tag{6.15}$$

While finding a closed-form solution is desirable, this again involves finding roots of a polynomial of degree at least 6 which is, generally, analytically intractable.

In the following, we illustrate an optimal fixed arrival policy for Example 6.1 and compare the obtained expected fuel consumption to the other considered policies for this example.

Example 6.3: Optimal Fixed Arrival Policy

Recall that the instance is characterized by the following values:

- *Waterway:* $L_u = L_d = 1$,
- *Vessel a:* $D = 6$ and $\tau_a = 1$ and
- *Vessel b:* $r = 0$ and $\tau_b \sim unif\{1, 3\}$.

A detailed formulation of the total expected fuel consumption in the fixed arrival setting is (see Figure 6.3.1 for a plot):

$$f(t) = \begin{cases} \frac{1}{t^2} + \frac{1}{3}\frac{1}{(6-(1+1))^2} + \frac{1}{3}\frac{1}{(6-(2+1))^2} + \frac{1}{3}\frac{1}{(6-(3+1))^2} & 0 < t < 1; \\ \frac{1}{t^2} + \frac{1}{3}\frac{1}{(6-(t+1))^2} + \frac{1}{3}\frac{1}{(6-(2+1))^2} + \frac{1}{3}\frac{1}{(6-(3+1))^2} & 1 \leq t < 2; \\ \frac{1}{t^2} + \frac{2}{3}\frac{1}{(6-(t+1))^2} + \frac{1}{3}\frac{1}{(6-(3+1))^2} & 2 \leq t < 3; \\ \frac{1}{t^2} + \frac{1}{(6-(t+1))^2} & 3 \leq t < 5. \end{cases}$$

The optimal fixed arrival policy chooses a deterministic arrival at the lock at time 2.668 yielding a total expected fuel consumption of

$$f(2.668) = 0.34.$$

This implies fuel savings of 2.86% compared to the oblivious approach in which vessel b is ignored. Compared to the near-optimal velocity policy the fuel consumption is only approximately 0.68% higher. Hence, for this example even an optimal fixed arrival policy leads to a significant share of possible fuel savings.

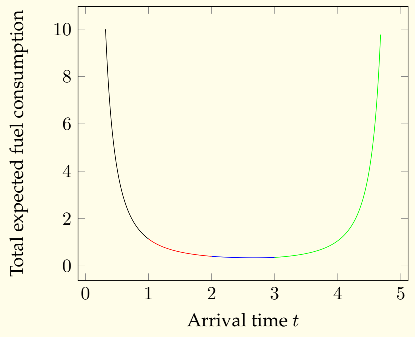


Figure 6.3.1: Objective function for the illustrative example.

6.4 Computational Study

We investigate the near-optimal velocity policy as well as the fixed arrival policy from different point-of-views based on a computational study. In particular, we consider two aspects of the policies: efficiency and simplicity. The efficiency of a policy is determined by its potential fuel savings compared to a benchmark policy. The simplicity of a policy is defined as its possibility of implementation from a practical point-of-view.

In order to assess the efficiency of the policies at hand, we compare them to the benchmark policy mentioned before which does not take into account the presence of vessel b and the inherent uncertainty. We refer to this benchmark policy as a-priori fixed arrival policy. This policy chooses a fixed arrival time

$$t^* = \frac{L_u + L_d}{D - \tau_a} L_u.$$

In order to appropriately compare the total expected fuel consumptions, we compute the total expected fuel consumption of t^* under uncertainty and use this consumption as a benchmark consumption. Let v^*_{dyn} and v^*_{fixed} denote the near-optimal velocity policy and optimal fixed arrival policy, respectively. Then, the efficiency of the policies is measured by two ratios: (1) the ratio of the total expected fuel consumption of v^*_{dyn} and t^* and (2) the ratio of the fuel consumption of v^*_{dyn} and v^*_{fixed}. The first ratio shows the near-optimal possible fuel savings and the second ratio indicates the share of these achievable with the optimal fixed arrival time policy. Based on these ratios, we aim to gain valuable insights answering the following questions regarding the efficiency of velocity policies:

E.1 What is the magnitude of fuel savings of the optimal dynamic velocity policy? How do the characteristics of vessel b impact the possible fuel savings?

E.2 What share of these savings can be achieved by the optimal fixed arrival policy? How do the characteristics of vessel b impact the possible fuel savings?

Additionally, we analyse the policies from a more practical point of view, i.e., from the eye of a skipper operating a vessel. While the fixed arrival policy is simple to implement for the skipper, the near-optimal velocity policy may imply frequent velocity changes and is, therefore, less attractive for skippers. To investigate the trade-off between simplicity and efficiency we consider the number of necessary velocity changes under the near-optimal policy and put this in relation to the added value in terms of fuel savings by comparing the fuel consumption of v^*_{dyn} to that of v^*_{fixed}. We aim to answer the following questions:

S.1 To what extent is the optimal dynamic velocity policy implementable by skippers? What is the trade-off between fuel savings and simplicity of the policy for skippers?

S.2 How do the characteristics of vessel b impact the simplicity?

6.4.1 Instance and Scenario parameters

The base setting of our computational study is given by a fixed waterway with a centrally located lock and a fixed deadline of vessel a:

- $L_u = L_d = 20\text{km}$
- $D = 150\text{min}$

We create different scenarios using the following parameters:

- τ_a: Lock processing time of vessel a
- r: Arrival time of vessel b at the lock
- τ_b: Lock processing time of vessel b

For the last parameter, we choose two families of scenarios based on the type of probability distribution.

First, we consider uniformly distributed τ_b. Uniform distributions are both relevant from a practical point of view as well as a theoretical point of view. From a practical point of view, uniform distributions can be used offer insights into the effect that the size of the time windows between realizations, i.e., the number of decision points may have. From a theoretical perspective uniform distributions function as a benchmark distribution with a small degree of variation. Throughout the study, we assume that follows a discrete uniform distribution within the interval $[20, 40]$. Furthermore, we parameterize the distribution by the number of realizations n. In this setting, we let $\tau_a = 30$.

The second class of scenarios is based on a truncated geometric distribution underlying τ_b. This captures real-life lock processing times as well as theoretical properties such as being memoryless. In this setting, we consider τ_b to be in the interval $[20, 39]$ based on a truncated geometric distribution with parameter $p \in \{0.1, 0.2, ..., 0.9\}$. The processing time of vessel a is constant with $\tau_a = 29.5$. In both classes of scenarios we choose the arrival time of vessel b at the lock to be $r \in [22, 59]$. An overview of the scenario parameters is given in Table 6.4.1.

τ_b	***Parameters***	τ_a	r
Uniform	$n \in \{2, 3, 5, 11, 21\}$	30	$[22, 59]$
Geometric	$p \in \{0.1, 0.2, ..., 0.9\}$	29.5	$[22, 59]$

Table 6.4.1: Scenario parameters.

The optimal arrival times for the benchmark policy in the two scenario families are

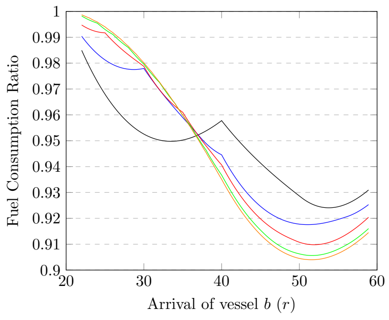
$$t^*_{uni} = \frac{L_u + L_d}{D - \tau_a} L_u = 60$$

and

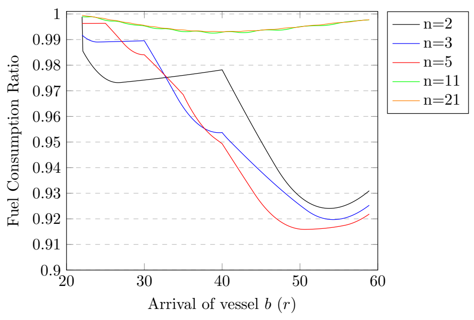
$$t^*_{geo} = \frac{L_u + L_d}{D - \tau_a} L_u = 60.25.$$

6.4.2 Efficiency

In Figure 6.4.1 the fuel consumption ratios of the near-optimal as well as the fixed arrival policy are plotted for uniformly distributed processing times as functions of the arrival time of vessel b at the lock for different value of n. Here, $n = 3$ implies that we use realizations $20 + k \cdot 20/(n-1)$ for $k = 0, \ldots, n-1$.



(a) Fuel consumption ratio of v^*_{dyn} compared to t^*_{uni}.



(b) Fuel consumption ratio of v^*_{dyn} compared to v^*_{fixed}.

Figure 6.4.1: Fuel consumption ratios for uniformly distributed processing times for τ_b.

To answer Question **E.1**, we turn our attention towards Figure 6.4.1a. First of all, we can observe that significant fuel changes of more than 9% can be achieved. Secondly, we can observe that for any n, the ratio follows a piece-wise convex function with respect to the arrival of vessel b. Taking a closer look at this relation, one can conclude that at any of the points between two convex parts one more realization of $r + \tau_b$ moves past t^*_{uni}. Consider Figure 6.4.2 for a more detailed example. Here, we can observe that at each point in time where one more realization of the lock finishing vessel b moves past the optimal arrival time ignoring vessel b, t^*_{uni}, the fuel consumption ratio transitions into a new convex piece. Intuitively, this can be explained as follows. Consider, for example the setting with $n = 2$. In this case, the lock becomes available at time $r + 20$ or at time $r + 40$ with probability $1/2$ each. This means, that when $r \leq 40$ the probability of the lock being available before time t^*_{uni} is 0.5 and for $r > 40$ this probability is 0. As t^*_{uni} is the optimal fixed arrival time without any uncertainty, this is the best fuel consumption we can reach and arriving at the lock close to this time is desired if its possible that the lock is available close to this time. Therefore, in both intervals below and above 40, the fuel consumption ratio follows a parabolic function.

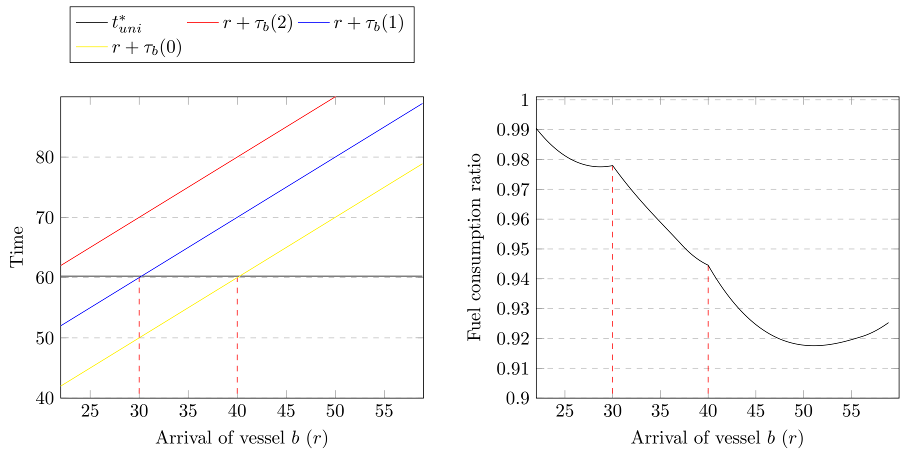
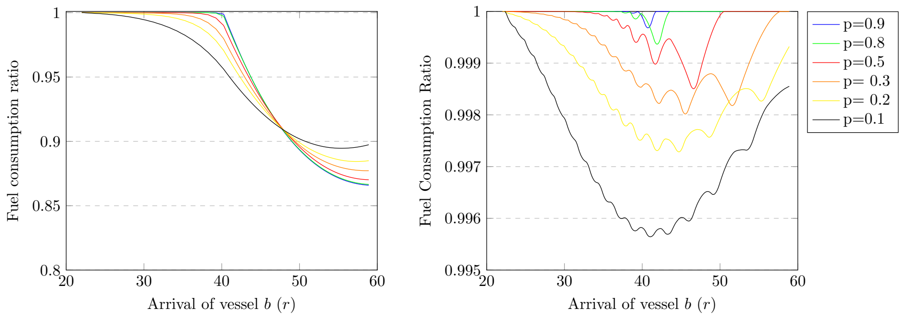


Figure 6.4.2: Detailed look at relation between fuel consumption of v^*_{dyn} and r with uniformly distributed processing times and $n = 3$.

With respect to Question **E.2**, Figure 6.4.1b shows the fuel consumption ratio of the near-optimal velocity policy and the fixed arrival policy. We can see that for small values of n the dynamic velocity policy leads to significantly higher fuel savings than the fixed arrival policy. For larger values of n, however, the fixed arrival policy already covers a large share of the possible fuel savings. In fact, for $n = 11$ or $n = 21$ the fuel consumption of v^*_{dyn} differs from the fuel consumption of v^*_{fixed} by less than 1%. Note that as the number of realizations increases the size of the time intervals between realizations decreases. This implies that choosing a fixed arrival time is subject to smaller expected waiting times at the lock and, therefore, the fixed arrival time may account for uncertainty more accurately in these settings. The relation between r and the fuel consumption ratio is piece-wise convex as well. This follows from the same explanation as the relation before.

Next, we take a closer look at the efficiency of the two policies in the setting where τ_b following geometric distributions (Figure 6.4.3).



(a) Fuel consumption ratio of v^*_{dyn} compared to t^*_{geo}.

(b) Fuel consumption ratio of v^*_{dyn} compared to v^*_{fixed}.

Figure 6.4.3: Fuel consumption ratios for geometrically distributed processing times for τ_b.

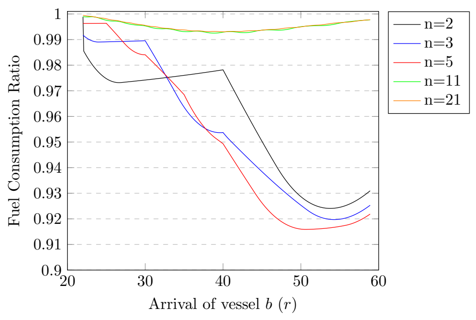
With respect to Question **E.1**, we can observe that the fuel consumption ratio of v^*_{dyn} compared to the benchmark solution follows a smooth relation with respect to r and savings of up to approximately 14% are possible. Similarly to the fuel savings for scenarios with uniformly distributed τ_b, the fuel savings decrease up to some value of r and increase beyond this value. This is again related to the t^*_{geo}, the optimal arrival time of vessel a at the lock when vessel b is not present. A peculiar observation is that the role of p switches at some value r'. Before r' the fuel savings are higher for smaller values of p which imply a higher degree of uncertainty at the lock. After this point the fuel savings are higher for larger values of p. This phenomenon can also be explained with the role of t^*_{geo}. For smaller values of r there is a positive probability that the lock can be entered by vessel a before t^*_{geo} and this probability is higher for higher values of p. This implies that a dynamic velocity policy will be less likely to significantly deviate from this arrival time. For high values of r the probability of entering the lock is higher when p is lower and, therefore, again the added value of changing velocities decreases.

Similar to the results of Figure 6.4.1b for a large number of realizations of τ_b, Figure 6.4.3b gives a clear answer to Question **E.2**. The fuel savings that are possible by taking into account are nearly fully covered by the optimal fixed arrival policy v^*_{fixed}. This highlights the power of such a simple velocity policy in settings where velocity changes may occur frequently. However, small additional savings can still be achieved with v^*_{dyn}.

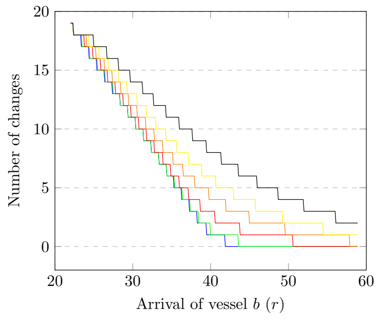
6.4.3 Simplicity

The trade-off between simplicity of the near-optimal velocity policy and additional fuel savings compared to the optimal fixed arrival policy is shown in Figures 6.4.4 and 6.4.5 for the scenarios with uniform and geometric distributions, respectively. Both classes of scenarios show similar results. First of all, the number of velocity changes decreases as the arrival of vessel b at the lock increases. An early arrival of vessel b at the lock implies that vessel a has more opportunities to enter the lock around its optimal benchmark arrival. Furthermore, for large values of r the number of possible realizations of τ_b such that vessel a can enter the lock close to time t^*_{uni} or t^*_{geo} decreases and, therefore, it is desired to reach the lock as soon as possible to avoid larger fuel consumption on the downstream segment. For the uniform distributions the number of velocity changes depends on the number of realizations which is also implied by Lemma 6.3. Finally, for the truncated geometric distribution we can observe that the number of changes decreases faster for larger values of p. Large values of p indicate a high probability of the lock finishing early. Therefore, an early arrival at the lock implies a low negative impact of waiting time on fuel consumption as high realizations of τ_b are less likely.

With respect to Question **S.2**, two observations can be made. For uniform distributions with low values of n the additional fuel savings through v^*_{dyn} can be achieved with few velocity changes which makes it attractive in practice. For uniform distributions with large n and geometric distributions, however, velocity changes are very frequent and only lead to small fuel savings over the optimal fixed arrival policy.

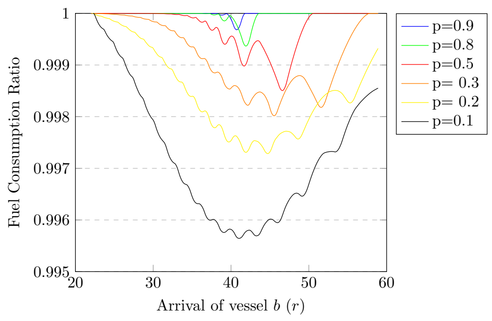


(a) Fuel consumption: v^*_{dyn}/v^*_{fixed}.

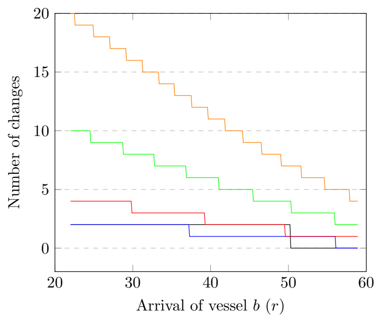


(b) Number of velocity changes for v^*_{dyn}.

Figure 6.4.4: Simplicity versus Efficiency with uniformly distributed τ_b.



(a) Fuel consumption: v^*_{dyn}/v^*_{fixed}.



(b) Number of velocity changes for v^*_{dyn}.

Figure 6.4.5: Simplicity versus Efficiency with geometrically distributed τ_b.

To summarize, in settings with more frequent realizations of τ_b the optimal fixed arrival policy covers a significant share of the possible fuel savings when taking into account uncertainty at the lock. When the number of realizations of τ_b is small and the time intervals between these realizations is large a dynamic velocity policy which can change velocity is preferable since a fixed arrival policy may lead to large waiting times at the lock. These waiting times lead to an increase in fuel consumption on the downstream segment of the waterway.

Bibliography

[1] A. H. Abdekhodaee and A. Wirth. "Scheduling parallel machines with a single server: some solvable cases and heuristics". In: *Computers & Operations Research* 29 (2002), pp. 295–315.

[2] B. Alidaee. "Minimizing absolute and squared deviation of completion times from due dates". In: *Production and Operations Management* 3 (1994), pp. 133–147.

[3] N. Alon, Y. Azar, G. J. Woeginger, and T. Yadid. "Approximation schemes for scheduling on parallel machines". In: *Journal of Scheduling* 1 (1998), pp. 55–66.

[4] T. Andersen, J. H. Hove, K. Fagerholt, and F. Meisel. "Scheduling ships with uncertain arrival times through the Kiel Canal". In: *Maritime Transport Research* 2 (2021).

[5] C. Annamalai, C. Kalaitzis, and O. Svensson. "Combinatorial Algorithm for Restricted Max-Min Fair Allocation". In: *ACM Transactions on Algorithms* 13 (2017), 37:1–37:28.

[6] A. Asadpour, U. Feige, and A. Saberi. "Santa claus meets hypergraph matchings". In: *ACM Transactions on Algorithms* 8 (2012), 24:1–24:9.

[7] H. Aydin N.and Lee and S. A. Mansouri. "Speed optimization and bunkering in liner shipping in the presence of uncertain service times and time windows at ports". In: *European Journal of Operational Research* 259 (2017), pp. 143–154.

[8] U. Bagchi, Y. Chang, and R. Sullivan. "Minimizing absolute and squared deviations of completion times with different earliness and tardiness penalties and a common due date". In: *Naval Research Logistics* 34 (1987), pp. 739–751.

[9] U. Bagchi, R. Sullivan, and Y. Chang. "Minimizing Mean Squared Deviation of Completion Times About a Common Due Date". In: *Management Science* 33 (1987), pp. 894–906.

[10] U. Bagchi, R. S. Sullivan, and Y. Chang. "Minimizing mean squared deviation of completion times about a common due date". In: *Management Science* 33 (1987), pp. 894–906.

[11] B. S. Baker and E. G. Coffman Jr. “Mutual exclusion scheduling”. In: *Theoretical Computer Science* 162 (1996), pp. 225–243.

[12] K. Baker and G. Scudder. “Sequencing with Earliness and Tardiness Penalties: A Review”. In: *Operations Research* 38 (1990), pp. 22–36.

[13] S. R. Balseiro, D. B. Brown, and C. Chen. “Static routing in stochastic scheduling: Performance guarantees and asymptotic optimality”. In: *Operations Research* 66 (2018), pp. 1641–1660.

[14] N. Bansal and M. Sviridenko. “The Santa Claus Problem”. In: *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, STOC*. 2006, pp. 31–40.

[15] M. Bendraouche and M. Boudhar. “Scheduling jobs on identical machines with agreement graph”. In: *Computers & Operations Research* 39 (2012), pp. 382–390.

[16] H. L. Bodlaender and F. V. Fomin. “Equitable colorings of bounded treewidth graphs”. In: *Theoretical Computer Science* 349 (2005), pp. 22–30.

[17] H. L. Bodlaender and K. Jansen. “On the complexity of scheduling incompatible jobs with unit-times”. In: *Proceedings of the International Symposium on Mathematical Foundations of Computer Science*. Springer. 1993, pp. 291–300.

[18] H. L. Bodlaender and K. Jansen. “Restrictions of graph partition problems. Part I”. In: *Theoretical Computer Science* 148 (1995), pp. 93–109.

[19] H. L. Bodlaender, K. Jansen, and G. J. Woeginger. “Scheduling with incompatible jobs”. In: *Discrete Applied Mathematics* 55 (1994), pp. 219–232.

[20] B. D. Brouer, J. Dirksen, D. Pisinger, C. E. M. Plum, and B. Vaaben. “The Vessel Schedule Recovery Problem (VSRP)–A MIP model for handling disruptions in liner shipping”. In: *European Journal of Operational Research* 224 (2013), pp. 362–374.

[21] P. Brucker, C. Dhaenens-Flipo, S. Knust, S. A. Kravchenko, and F. Werner. “Complexity results for parallel machine problems with a single server”. In: *Journal of Scheduling* 5 (2002), pp. 429–457.

[22] J. Bruno, E. G. Coffman Jr., and R. Sethi. “Scheduling independent tasks to reduce mean finishing time”. In: *Communications of the ACM* 17 (1974), pp. 382–387.

[23] M. Buchem, J. A. P. Golak, and A. Grigoriev. "Vessel velocity decisions in inland waterway transportation under uncertainty". In: *European Journal of Operational Research* 296 (2022), pp. 669–678.

[24] M. Buchem, L. Kleist, and D. Schmidt genannt Waldschmidt. "Scheduling with Machine Conflicts". In: *CoRR* abs/2102.08231 (2021).

[25] M. Buchem, L. Rohwedder, T. Vredeveld, and A. Wiese. "Additive Approximation Schemes for Load Balancing Problems". In: *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2021.

[26] M. Buchem and T. Vredeveld. "Performance analysis of fixed assignment policies for stochastic online scheduling on uniform parallel machines". In: *Computers & Operations Research* 125 (2021).

[27] S. Chakrabarti and S. Muthukrishnan. "Resource scheduling for parallel database and scientific applications". In: *Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*. 1996, pp. 329–335.

[28] D. Chakrabarty. "Max-Min Allocation". In: *Encyclopedia of Algorithms*. Ed. by M.-Y. Kao. Boston, MA: Springer US, 2016, pp. 1244–1247.

[29] C. Chekuri, R. Johnson, R. Motwani, B. Natarajan, B. Rau, and M. Schlansker. "Profile-driven instruction level parallel scheduling with application to super blocks". In: *Proceedings of the 29th IEEE/ACM International Symposium on Microarchitecture*. 1996, pp. 58–69.

[30] L. Chen, K. Jansen, and G. Zhang. "On the optimality of exact and approximation algorithms for scheduling problems". In: *Journal of Computer and System Sciences* 96 (2018), pp. 1–32.

[31] S.-W. Cheng and Y. Mao. "Restricted Max-Min Allocation: Approximation and Integrality Gap". In: *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming, ICALP*. 2019, 38:1–38:13.

[32] T. Cheng, C. Ng, and V. Kotov. "A new algorithm for online uniform machine scheduling to minimize the makespan". In: *Information Processing Letters* 99 (2006), pp. 102–105.

[33] Y. Cho and S. Sahni. "Bounds for List Schedules on Uniform Processors". In: *SIAM Journal on Computing* 9 (1980), pp. 91–103.

[34] M. C. Chou, H. Liu, M. Queyranne, and D. Simchi-Levi. “On the asymptotic optimality of a simple on-line algorithm for the stochastic single-machine weighted completion time problem and its extensions”. In: *Operations Research* 54 (2006), pp. 464–474.

[35] M. Chrobak, J. Csirik, C. Imreh, J. Noga, J. Sgall, and G. J. Woeginger. “The buffer minimization problem for multiprocessor scheduling with conflicts”. In: *International Colloquium on Automata, Languages, and Programming*. Springer. 2001, pp. 862–874.

[36] S. A. Cook. “The Complexity of Theorem-Proving Procedures”. In: *Proceedings of the Third Annual ACM Symposium on Theory of Computing*. STOC ’71. 1971, pp. 151–158.

[37] J. R. Correa and M. R. Wagner. “LP-based online scheduling: from single to parallel machines”. In: *Mathematical Programming* 119 (2009), pp. 109–136.

[38] J. Csirik, H. Kellerer, and G. Woeginger. “The exact LPT-bound for maximizing the minimum completion time”. In: *Operations Research Letters* 11 (1992), pp. 281–287.

[39] S. Das and A. Wiese. “On Minimizing the Makespan When Some Jobs Cannot Be Assigned on the Same Machine”. In: *Proceedings of the 25th Annual European Symposium on Algorithms*. 2017.

[40] S. Davies, T. Rothvoss, and Y. Zhang. “A Tale of Santa Claus, Hypergraphs and Matroids”. In: *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA*. 2020, pp. 2748–2757.

[41] P. De, J. Ghosh, and C. Wells. “A Note on the Minimization of Mean Squared Deviation of Completion Times About a Common Due Date”. In: *Management Science* 35 (1989), pp. 1143–1147.

[42] P. De, J. Ghosh, and C. Wells. “Scheduling about a common due date with earliness and tardiness penalties”. In: *Computers & Operations Research* 17 (1990), pp. 231–241.

[43] C. Defryn, J. A. P. Golak, A. Grigoriev, and V. Timmermans. “Inland waterway efficiency through skipper collaboration and joint speed optimization”. In: *European Journal of Operational Research* 292 (2021), pp. 276–285.

[44] M. A. Dempster, J. K. Lenstra, and A. H. G. Rinnooy Kan. *Deterministic and Stochastic Scheduling: proceedings of the NATO Advanced Study and Research Institute on Theoretical Approaches to Schelduling Problems held in Durham, England, July 6-17, 1981*. NATO Advanced Study Institutes Series: Ser. C. Springer Netherlands, 1982.

[45] B. L. Deuermeyer, D. K. Friesen, and M. A. Langston. "Scheduling to maximize the minimum processor finish time in a multiprocessor system". In: *SIAM Journal on Algebraic Discrete Methods* 3 (1982), pp. 190–196.

[46] A. Dolgui, V. Kotov, A. Nekrashevich, and A. Quilliot. "General parametric scheme for the online uniform machine scheduling problem with two different speeds". In: *Information Processing Letters* 134 (2018), pp. 18–23.

[47] F. Eberle, F. Fischer, J. Matuschke, and N. Megow. "On index policies for stochastic minsum scheduling". In: *Operations Research Letters* 47 (2019), pp. 213–218.

[48] J. Edmonds. "Matroids and the greedy algorithm". In: *Mathematical Programming* 1 (1971), pp. 127–136.

[49] S. Eilon and I.G. Chowdhury. "Minimising Waiting Time Variance in the Single Machine Problem". In: *Management Science* 23 (1977), pp. 567–575.

[50] F. Eisenbrand and G. Shmonin. "Carathéodory bounds for integer cones". In: *Operations Research Letters* 34 (2006), pp. 564–568.

[51] European Commission. *A European Strategy for Low-Emission Mobility*. https://eur-lex.europa.eu/resource.html?uri=cellar:e44d3c21-531e-11e6-89bd-01aa75ed71a1.0002.02/DOC_1&format=PDF. Accessed 05 May 2022. 2016.

[52] European Commission. *The European Green Deal*. https://eur-lex.europa.eu/resource.html?uri=cellar:b828d165-1c22-11ea-8c1f-01aa75ed71a1.0002.02/DOC_1&format=PDF. Accessed 05 May 2022. 2019.

[53] European Parliament. *Inland waterway transport in the EU*. https://www.europarl.europa.eu/RegData/etudes/BRIE/2022/698918/EPRS_BRI(2022)698918_EN.pdf. Accessed 05 May 2022. 2022.

[54] G. Even, M. M. Halldórsson, L. Kaplan, and D. Ron. "Scheduling with conflicts: online and offline algorithms". In: *Journal of Scheduling* 12 (2009), pp. 199–224.

[55] K. Fagerholt, G. Laporte, and I. Norstad. "Reducing fuel emissions by optimizing speed on shipping routes". In: *Journal of the Operational Research Society* 61 (2010), pp. 523–529.

[56] U. Feige. "On allocations that maximize fairness". In: *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*. SIAM, 2008, pp. 287–293.

[57] F. Gardi. "Mutual exclusion scheduling with interval graphs or related classes, Part I". In: *Discrete Applied Mathematics* 157 (2009), pp. 19–35.

[58] M. R. Garey and D. S. Johnson. ""Strong" NP-completeness results: motivation, examples, and implications". In: *Journal of the ACM* 25 (1978), pp. 499–508.

[59] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1979.

[60] P. C. Gilmore and R. E. Gomory. "A linear programming approach to the cutting-stock problem". In: *Operations Research* 9 (1961), pp. 849–859.

[61] R. L. Graham. "Bounds for certain multiprocessing anomalies". In: *Bell System Technical Journal* 45 (1966), pp. 1563–1581.

[62] R. L. Graham. "Bounds on multiprocessing timing anomalies". In: *SIAM Journal on Applied Mathematics* 17 (1969), pp. 416–429.

[63] 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". In: *Annals of Discrete Mathematics* 5 (1979), pp. 287–326.

[64] M. Gu and X. Lu. "Asymptotical optimality of WSEPT for stochastic online scheduling on uniform machines". In: *Annals of Operations Research* 191 (2011), pp. 97–113.

[65] V. Gupta, B. Moseley, M. Uetz, and Q. Xie. "Corrigendum: Greed works—online algorithms for unrelated machine stochastic scheduling". In: *Mathematics of Operations Research* 46 (2021), pp. 1230–1234.

[66] V. Gupta, B. Moseley, M. Uetz, and Q. Xie. "Greed Works–Online Algorithms For Unrelated Machine Stochastic Scheduling". In: *Mathematics of Operations Research* 45 (2019), pp. 497–516.

[67] V. Gupta, B. Moseley, M. Uetz, and Q. Xie. "Greed works—online algorithms for unrelated machine stochastic scheduling". In: *Mathematics of Operations Research* 45 (2020), pp. 497–516.

[68] V. Gupta, B. Moseley, M. Uetz, and Q. Xie. "Stochastic Online Scheduling on Unrelated Machines". In: *Integer Programming and Combinatorial Optimization (IPCO 2017)*. Ed. by Friedrich Eisenbrand and Jochen Koenemann. Vol. 10328. Lecture Notes in Computer Science. Waterloo, ON, Canada: Springer, 2017, pp. 228–240.

[69] B. Haeupler, B. Saha, and A. Srinivasan. "New Constructive Aspects of the Lovász Local Lemma". In: *Journal of the ACM* 58 (2011).

[70] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. "Scheduling to minimize average completion time: Off-line and on-line approximation algorithms". In: *Mathematics of Operations Research* 22 (1997), pp. 513–544.

[71] N. G. Hall, W. Kubiak, and S. P. Sethi. "Earliness–tardiness scheduling problems, II: Deviation of completion times about a restrictive common due date". In: *Operations Research* 39 (1991), pp. 847–856.

[72] N. G. Hall, C. N. Potts, and C. Sriskandarajah. "Parallel machine scheduling with a common server". In: *Discrete Applied Mathematics* 102 (2000), pp. 223–243.

[73] N.G. Hall and M.E. Posner. "Earliness-Tardiness Scheduling Problems, I: Weighted Deviation of Completion Times about a Common Due Date". In: *Operations Research* 39 (1991), pp. 836–846.

[74] P. Hansen, A. Hertz, and J. Kuplinsky. "Bounded vertex colorings of graphs". In: *Discrete Mathematics* 111 (1993), pp. 305–312.

[75] J. Håstad. "Clique is hard to approximate within 1- ε". In: *Acta Mathematica* 182 (1999), pp. 105–142.

[76] P. Haxell and T. Szabó. "Improved Integrality Gap in Max-Min Allocation: or Topology at the North Pole". In: *arXiv preprint arXiv:2202.01143* (2022).

[77] J. Hermans. "Optimization of inland shipping". In: *Journal of Scheduling* 17 (2014), pp. 305–319.

[78] R. Hoberg and T. Rothvoss. “A Logarithmic Additive Integrality Gap for Bin Packing”. In: *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*. SIAM, 2017, pp. 2616–2625.

[79] D. S. Hochbaum. “Various notions of approximations: Good, better, best and more”. In: *Approximation algorithms for NP-hard problems* (1997), pp. 346–398.

[80] D. S. Hochbaum and D. B. Shmoys. “Using dual approximation algorithms for scheduling problems: Theoretical and practical results”. In: *Journal of the ACM* 34 (1987), pp. 144–162.

[81] F. Höhne and R. van Stee. “Buffer minimization with conflicts on a line”. In: *Theoretical Computer Science* 876 (2021), pp. 25–33.

[82] J. A. Hoogeveen, H. Oosterhout, and S. L. van de Velde. “New lower and upper bounds for scheduling around a small common due date”. In: *Operations Research* 42 (1994), pp. 102–110.

[83] J. A. Hoogeveen and S. L. van de Velde. “Earliness-tardiness scheduling around almost equal due dates”. In: *INFORMS Journal on Computing* 9 (1997), pp. 92–99.

[84] J.A. Hoogeveen and S.L. Van de Velde. “Scheduling around a small common due date”. In: *European Journal of Operational Research* 55 (1991), pp. 237–242.

[85] L. M. Hvattum, I. Norstad, K. Fagerholt, and G. Laporte. “Analysis of an exact algorithm for the vessel speed optimization problem”. In: *Networks* 62 (2013), pp. 132–135.

[86] A. Inoue and Y. Kobayashi. “An Additive Approximation Scheme for the Nash Social Welfare Maximization with Identical Additive Valuations”. In: *arXiv preprint arXiv:2201.01419* (2022).

[87] S. J. Jäger. “Approximation in deterministic and stochastic machine scheduling”. PhD thesis. TU Berlin, 2021.

[88] S. J. Jäger and M. Skutella. “Generalizing the Kawaguchi-Kyan Bound to Stochastic Parallel Machine Scheduling”. In: *35th Symposium on Theoretical Aspects of Computer Science*. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 43:1–43:14.

[89] K. Jansen. “An EPTAS for scheduling jobs on uniform processors: Using an MILP relaxation with a constant number of integral variables”. In: *SIAM Journal on Discrete Mathematics* 24 (2010), pp. 457–485.

[90] K. Jansen, K.-M. Klein, and J. Verschae. “Closing the Gap for Makespan Scheduling via Sparsification Techniques”. In: *Mathematics of Operations Research* 45 (2020), pp. 1371–1392.

[91] K. Jansen and L. Rohwedder. “A Quasi-Polynomial Approximation for the Restricted Assignment Problem”. In: *Proceedings of the 19th International Conference on Integer Programming and Combinatorial Optimization, IPCO*. 2017, pp. 305–316.

[92] K. Jansen and L. Rohwedder. “On the Configuration-LP of the Restricted Assignment Problem”. In: *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*. 2017, pp. 2670–2678.

[93] Y. Jiang, Q. Zhang, J. Hu, J. Dong, and M. Ji. “Single-server parallel-machine scheduling with loading and unloading times”. In: *Journal of Combinatorial Optimization* 30 (2015), pp. 201–213.

[94] T. Kämpke. “On the optimality of static priority policies in stochastic scheduling on parallel machines”. In: *Journal of Applied Probability* 24 (1987), pp. 430–448.

[95] J.J. Kanet. “Minimizing the average deviation of job completion times about a common due date”. In: *Naval Research Logistics Quarterly* 28 (1981), pp. 643–651.

[96] N. Karmarkar and R. M. Karp. “An Efficient Approximation Scheme for the One-Dimensional Bin-Packing Problem”. In: *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science, FOCS*. IEEE Computer Society, 1982, pp. 312–320.

[97] R. M. Karp. “Reducibility among combinatorial problems”. In: *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[98] T. Kawaguchi and S. Kyan. “Worst case bound of an LRF schedule for the mean weighted flow-time problem”. In: *SIAM Journal on Computing* 15 (1986), pp. 1119–1129.

[99] W. Kern and W. N. Nawijn. “Scheduling multi-operation jobs with time lags on a single machine”. In: *Proceedings of the 2nd Twente Workshop on Graphs and Combinatorial Optimization*. 1991.

[100] D. Kőnig. “Gráfok és Mátrixok”. In: *Matematikai és Fizikai Lapok* 38 (1931), pp. 116–119.

[101] B. H. Korte and J. Vygen. *Combinatorial Optimization*. Vol. 6. Springer, 2018.

[102] M. Y. Kovalyov and W. Kubiak. "A Fully Polynomial Approximation Scheme for the Weighted Earliness–Tardiness Problem". In: *Operations Research* 47 (1999), pp. 757–761.

[103] S. A. Kravchenko and F. Werner. "Parallel machine scheduling problems with a single server". In: *Mathematical and Computer Modelling* 26 (1997), pp. 1–11.

[104] W. Kubiak. "Completion time variance minimization on a single machine is difficult". In: *Operations Research Letters* 14 (1993), pp. 49–59.

[105] H. W. Kuhn. "The Hungarian method for the assignment problem". In: *Naval Research Logistics Quarterly* 2 (1955), pp. 83–97.

[106] A. Kurpisz, M. Mastrolilli, C. Mathieu, T. Mömke, V. Verdugo, and A. Wiese. "Semidefinite and linear programming integrality gaps for scheduling identical machines". In: *Mathematical Programming* 172 (2018), pp. 231–248.

[107] J. K. Lenstra, D. B. Shmoys, and E. Tardos. "Approximation algorithms for scheduling unrelated parallel machines". In: *Mathematical Programming* 46 (1990), pp. 259–271.

[108] H. W. Lenstra Jr. "Integer programming with a fixed number of variables". In: *Mathematics of operations research* 8 (1983), pp. 538–548.

[109] J. Y. T. Leung. "Bin packing with restricted piece sizes". In: *Information Processing Letters* 31 (1989), pp. 145–149.

[110] J. Y. T. Leung, H. Li, M. Pinedo, and J. Zhang. "Minimizing total weighted completion time when scheduling orders in a flexible environment with uniform machines". In: *Information Processing Letters* 103 (2007), pp. 119–129.

[111] R. Li and L. Shi. "An on-line algorithm for some uniform processor scheduling". In: *SIAM Journal on Computing* 27 (1998), pp. 414–422.

[112] R. J. Lipton, E. Markakis, E. Mossel, and A. Saberi. "On Approximately Fair Allocations of Indivisible Goods". In: *Proceedings of the 5th ACM Conference on Electronic Commerce, EC*. 2004, pp. 125–131.

[113] J. W. S. Liu and C. L. Liu. "Bounds on scheduling algorithms for heterogenous computing systems". In: *Information Processing: Proceedings of the IFIP congress*. 1974.

[114] J. W. S. Liu and A. Yang. "Optimal Scheduling of Independent Tasks on Heterogeneous Computing Systems". In: *Proceedings of the 1974 Annual Conference - Volume 1*. 1974, pp. 38–45.

[115] E. Lübbecke, M. E. Lübbecke, and R. H. Möhring. “Ship traffic optimization for the Kiel Canal”. In: *Operations Research* 67 (2019), pp. 791–812.

[116] C. Lund and M. Yannakakis. “The approximation of maximum subgraph problems”. In: *Proceedings of the 20th International Colloquium on Automata, Languages, and Programming*. Springer. 1993, pp. 40–51.

[117] N. Megow and A. S. Schulz. “On-line scheduling to minimize average completion time revisited”. In: *Operations Research Letters* 32 (2004), pp. 485–490.

[118] N. Megow, M. Uetz, and T. Vredeveld. “Models and algorithms for stochastic online scheduling”. In: *Mathematics of Operations Research* 31 (2006), pp. 513–525.

[119] Merriam-Webster. *Schedule*. In: *Merriam-Webster's Collegiate Dictionary*. 10th. Springfield, MA, USA: Merriam- Webster Inc., 2001.

[120] A.G. Merten and M. E. Muller. “Variance minimization in single machine sequencing problems”. In: *Management Science* 18 (1972), pp. 518–528.

[121] A. Mohabeddine and M. Boudhar. “New results in two identical machines scheduling with agreement graphs”. In: *Theoretical Computer Science* 779 (2019), pp. 37–46.

[122] R. H. Möhring, A. S. Schulz, and M. Uetz. “Approximation in stochastic scheduling: the power of LP-based priority policies”. In: *Journal of the ACM* 46 (1999), pp. 924–942.

[123] G. Mosheiov and U. Yovel. “Minimizing weighted earliness–tardiness and due-date cost with unit processing-time jobs”. In: *European Journal of Operational Research* 172 (2006), pp. 528–544. DOI: `https://doi.org/10.1016/j.ejor.2004.10.021`.

[124] I. Norstad, K. Fagerholt, and G. Laporte. “Tramp ship routing and scheduling with speed optimization”. In: *Transportation Research Part C: Emerging Technologies* 19 (2011), pp. 853–865.

[125] W. Passchyn. “Scheduling locks on inland waterways.” PhD thesis. KU Leuven, 2016.

[126] W. Passchyn, D. Briskorn, and F. C. R. Spieksma. “Mathematical programming models for lock scheduling with an emission objective”. In: *European Journal of Operational Research* 248 (2016), pp. 802–814.

[127] W. Passchyn, D. Briskorn, and F. C. R. Spieksma. "Mathematical programming models for lock scheduling with an emission objective". In: *European Journal of Operational Research* 248 (2016), pp. 802–814.

[128] W. Passchyn, D. Briskorn, and F. C. R. Spieksma. "No-wait scheduling for locks". In: *INFORMS Journal on Computing* 31 (2019), pp. 413–428.

[129] W. Passchyn, S. Coene, D. Briskorn, J. L. Hurink, F. C. R. Spieksma, and G. vanden Berghe. "The lockmaster's problem". In: *European Journal of Operational Research* 251 (2016), pp. 432–441.

[130] W. Passchyn and F. C. R. Spieksma. "Scheduling parallel batching machines in a sequence". In: *Journal of Scheduling* 22 (2019), pp. 335–357.

[131] M. L. Pinedo. *Scheduling*. Heidelberg, Germany: Springer, 2012.

[132] N. Pinson and F. C. R. Spieksma. "Online interval scheduling on two related machines: the power of lookahead". In: *Journal of Combinatorial Optimization* 38 (2019), pp. 224–253.

[133] M. Prandtstetter, U. Ritzinger, P. Schmidt, and M. Ruthmair. "A variable neighborhood search approach for the interdependent lock scheduling problem". In: *European Conference on Evolutionary Computation in Combinatorial Optimization*. Springer. 2015, pp. 36–47.

[134] K. Pruhs, J. Sgall, and E. Torng. "Online Scheduling". In: *Handbook of scheduling: Algorithms, models, and performance analysis*. Ed. by J.Y.T. Leung. Chapman and Hall/CRC, Boca Raton, FL,USA, 2004.

[135] M. Queyranne. "Structure of a simple scheduling polyhedron". In: *Mathematical Programming* 58 (1993), pp. 263–285.

[136] R. Rajkumar, L. Sha, and J. P. Lehoczky. "Real-Time Synchronization Protocols for Multiprocessors." In: *Proceedings of the 9th IEEE Real-Time Systems Symposium*. Vol. 88. 1988, pp. 259–269.

[137] G. A. Rolim and Marcelo S. Nagano. "Structural properties and algorithms for earliness and tardiness scheduling against common due dates and windows: A review". In: *Computers & Industrial Engineering* 149 (2020). Article no. 106803.

[138] M. H. Rothkopf. "Scheduling with random service times". In: *Management Science* 12 (1966), pp. 707–713.

[139] T. Rothvoss. "Better Bin Packing Approximations via Discrepancy Theory". In: *SIAM Journal on Computing* 45 (2016), pp. 930–946.

[140] S. K. Sahni. “Algorithms for scheduling independent tasks”. In: *Journal of the ACM* 23 (1976), pp. 116–127.

[141] S. K. Sahni. “Scheduling master-slave multiprocessor systems”. In: *IEEE Transactions on Computers* 45 (1996), pp. 1195–1199.

[142] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Algorithms and Combinatorics. Heidelberg, Germany: Springer, 2003.

[143] A. S. Schulz. “New Old Algorithms for Stochastic Scheduling”. In: *Algorithms for Optimization with Incomplete Information*. Ed. by Susanne Albers, Rolf H. Möhring, Georg Ch. Pflug, and Rüdiger Schultz. Dagstuhl, Germany: Internationales Begegnungs- und Forschungszentrum für Informatik, Schloss Dagstuhl, Germany, 2005.

[144] A. S. Schulz. “Polytopes and scheduling”. PhD thesis. Technische Universitaet Berlin, 1996.

[145] A. S. Schulz. “Scheduling to minimize total weighted completion time: Performance guarantees of LP-based heuristics and lower bounds”. In: *International Conference on Integer Programming and Combinatorial Optimization*. Springer. 1996, pp. 301–315.

[146] A. S. Schulz. “Stochastic online scheduling revisited”. In: *International Conference on Combinatorial Optimization and Applications*. Springer. 2008, pp. 448–457.

[147] J.B. Sidney. “Optimal Single-Machine Scheduling with Earliness and Tardiness Penalties”. In: *Operations Research* 25 (1977), pp. 62–69.

[148] M. Skutella. “Convex quadratic and semidefinite programming relaxations in scheduling”. In: *Journal of the ACM* 48 (2001), pp. 206–242.

[149] M. Skutella, M. Sviridenko, and M. Uetz. “Unrelated machine scheduling with stochastic processing times”. In: *Mathematics of Operations Research* 41 (2016), pp. 851–864.

[150] W. E. Smith. “Various optimizers for single-stage production”. In: *Naval Research Logistics* 3 (1956), pp. 59–66.

[151] B. Srirangacharyulu and G. Srinivasan. “An exact algorithm to minimize mean squared deviation of job completion times about a common due date”. In: *European Journal on Operational Research* 231 (2013), pp. 547–556.

[152] B. Srirangacharyulu and G. Srinivasan. “Minimising mean squared deviation of job completion times about a common due date in multimachine systems”. In: *European Journal of Industrial Engineering* 5 (2011), pp. 424–447.

[153] Y. Sugimori, K. Kusunoki, F. Cho, and S. UCHIKAWA. “Toyota production system and kanban system materialization of just-in-time and respect-for-human system”. In: *The International Journal of Production Research* 15 (1977), pp. 553–564.

[154] P. S. Sundararaghavan and M. U. Ahmed. “Minimizing the sum of absolute lateness in single-machine and multimachine scheduling”. In: *Naval Research Logistics Quarterly* 31 (1984), pp. 325–333.

[155] O. Svensson. “Santa Claus Schedules Jobs on Unrelated Machines”. In: *SIAM Journal on Computing* 41 (2012), pp. 1318–1341.

[156] D. Trietsch, L. Mazmanyan, L. Gevorgyan, and K. R. Baker. “Modeling activity times by the Parkinson distribution with a lognormal core: Theory and validation”. In: *European Journal of Operational Research* 216 (2012), pp. 386–396.

[157] N. H. Tuong and A. Soukhal. “Due dates assignment and JIT scheduling with equal-size jobs”. In: *European Journal of Operational Research* 205 (2010), pp. 280–289.

[158] M. Uetz. “Algorithms for deterministic and stochastic scheduling”. PhD thesis. TU Berlin, 2001.

[159] P. M. Vaidya. “An algorithm for linear programming which requires O (((m+ n) n 2+(m+ n) 1.5 n) L) arithmetic operations”. In: *Mathematical Programming* 47 (1990), pp. 175–201.

[160] P. M. Vaidya. “Speeding-up linear programming using fast matrix multiplication”. In: *30th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society. 1989, pp. 332–337.

[161] V.V. Vazirani. *Approximation Algorithms*. Heidelberg, Germany: Springer, 2001.

[162] J. Verstichel. “The lock scheduling problem”. PhD thesis. KU Leuven, 2013.

[163] J. Verstichel, P. De Causmaecker, and G. Vanden Berghe. “Scheduling algorithms for the lock scheduling problem”. In: *Procedia-Social and Behavioral Sciences* 20 (2011), pp. 806–815.

[164] J. Verstichel, P. De Causmaecker, F. C. R. Spieksma, and G. van den Berghe. "The generalized lock scheduling problem: An exact approach". In: *Transportation Research Part E: Logistics and Transportation Review* 65 (2014), pp. 16–34.

[165] A. P. A. Vestjens. "On-line machine scheduling". PhD thesis. Technische Universiteit Eindhoven, 1997.

[166] V. G. Vizing. "On an estimate of the chromatic class of a p-graph (in Russian)". In: *Diskret. Analiz* 3 (1964), pp. 25–30.

[167] S. Wang and Q. Meng. "Liner ship route schedule design with sea contingency time and port time uncertainty". In: *Transportation Research Part B: Methodological* 46 (2012), pp. 615–633.

[168] D. de Werra. "Restricted coloring models for timetabling". In: *Discrete Mathematics* 165 (1997), pp. 161–170.

[169] D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms*. Cambridge, United Kingdom: Cambridge University Press, 2011.

[170] G. J. Woeginger. "A polynomial-time approximation scheme for maximizing the minimum machine completion time". In: *Operations Research Letters* 20 (1997), pp. 149–154.

[171] G. J. Woeginger. "When does a dynamic programming formulation guarantee the existence of a fully polynomial time approximation scheme (FPTAS)?" In: *INFORMS Journal on Computing* 12 (2000), pp. 57–74.

[172] X. Xie, Y. Li, H. Zhou, and Y. Zheng. "Scheduling parallel machines with a single server". In: *Proceedings of 2012 International Conference on Measurement, Information and Control*. Vol. 1. IEEE. 2012, pp. 453–456.

[173] X. Zhang, R. Ma, J. Sun, and Z. B. Zhang. "Randomized selection algorithm for online stochastic unrelated machines scheduling". In: *Journal of Combinatorial Optimization* (2020), forthcoming.

[174] D. Zuckerman. "Linear degree extractors and the inapproximability of max clique and chromatic number". In: *Proceedings of the 38th annual ACM Symposium on Theory of Computing*. 2006, pp. 681–690.

Summary and General Discussion

This dissertation entails the theoretical and empirical study of various scheduling problems from theory and practice. In Chapters 2-4, the focus lies on *deterministic scheduling problems* for which the input is known and available without any uncertainty. Chapters 5 and 6 study *scheduling problems with uncertainty*. In the following we summarize the findings of each individual chapter and outline the main conclusions.

In *Chapter 2*, we investigate a family of load balancing problems on identical parallel machines. As it is unlikely that efficient algorithms exist which solve these problems to optimality, we aim at polynomial time algorithms which find solutions of provably good quality. More precisely, we formalize the concept of additive approximation schemes. These approximation schemes offer a solution technique with a clear trade-off between the quality of the solution and the running time needed to find the solution. The challenge when developing additive approximation schemes is that standard techniques for multiplicative approximation schemes are not applicable. To overcome this challenge, we introduce a new relaxation of the considered load balancing problems. Our additive polynomial time approximation schemes are based on structural properties which allow us to solve this relaxation complement with a local search technique which finds a final schedule whose objective value is provably close to that of the solution to the relaxation.

The new techniques as well as the concept of additive approximation schemes offer promising directions for future research such as extending the techniques to other scheduling problems. Especially considering problems for which it is known that no multiplicative approximation scheme exists, the notion of additive approximation schemes may offer a pathway to non-trivial approximation algorithms.

Chapter 3 investigates a natural extension of the identical parallel machine environment by combining this problem with the notion of machine conflicts. These conflicts are relevant when machines are cooperating with a single server responsible for pre- and post-processing of jobs and a pair of conflicting machines may not access the server at the same time. We establish a close connection between this scheduling problem and the graph theoretical problem of finding a maximum induced bipartite subgraph when all jobs are identical and pre-processing, processing and post-processing of any job takes one time unit. This connection leads to a strong inapproximability result implying that for general conflict graphs we cannot achieve any non-trivial approximation guarantees, unless $\mathcal{P} = \mathcal{NP}$. Therefore, we restrict our attention to special graph classes which are relevant from an applied or theoretical point of view.

The results in this chapter open up various interesting directions for future research. An open question which remains is whether, in general, the special case with unit jobs can be solved in polynomial time whenever a specially structured maximum induced bipartite subgraph can be computed in polynomial time or whether there exist graph classes for which this is not possible. If the former is the case, the presented algorithm for bipartite graphs may be a first step towards a polynomial time algorithm for other graph classes. Furthermore, considering more general job parameters with simple conflict graph classes poses an interesting research direction to investigate the quality of classic scheduling algorithms in this context of machine conflicts. Moreover, investigating graph classes which capture machine conflicts caused by spatial proximity such as geometric intersection graphs poses an interesting avenue for future research.

In *Chapter 4*, we turn our attention towards the problem of minimizing the total (weighted) squared deviation from job specific due dates. We focus on the case where all jobs have the same processing time. In the unweighted setting we devised a polynomial time algorithm based on the insight that in an optimal schedule jobs are evenly distributed among machines. In the weighted setting, we developed a polynomial time algorithm for a single common due date if the due date is sufficiently large. If the due date is small, we devised an algorithm which runs in time exponential in the number of machines. Furthermore, when jobs have distinct due dates we present an additive fully polynomial time approximation scheme if the number of distinct due dates is constant.

Multiple interesting questions for future research remain open in the weighted setting. First of all, settling the complexity for a single common restrictive due date is of interest. Furthermore, the setting of k distinct due dates in the weighted version remains interesting since for absolute deviation penalties this can be solved in polynomial time via an assignment problem. However, this uses the fact that optimal schedules use integral starting times when considering absolute deviation penalties, which is not true for squared deviation penalties. Hence, different techniques and insights are needed for this next step.

Chapter 5 entails theoretical and empirical analyses of stochastic online scheduling policies for the problem of minimizing total weighted expected completion time. In particular, we consider the case where machines are either fast or slow. We adapt stochastic online scheduling policies which were previously known for the identical and unrelated machine environment by taking into account the machine speeds. From a theoretical perspective, we derive a performance guarantee for each of the policies in the online-list as well as the online-time model. In the online-list model, we further prove that the policy is asymptotically optimal. From a practical point of view, we analyse the realized performance of the policies compared to the theoretical performance. This analysis shows that the true performance is better than expected based on the theoretical results. The most striking result of the computational study is that the two lower bounds used in the theoretical performance analysis of the policies do not yield better lower bounds than the trivial lower bound of the weighted sum of expected processing times divided by the speed of the fast machine for small instances.

For future research, considering more general speed models is of interest. Furthermore, it remains an open question whether one can show that special policies for the uniform parallel machine environment can yield performance guarantees strictly better than those implied by the more general setting of unrelated parallel machines even for the special case considered in this chapter. This, however, would require new theoretical lower bounds or a different analysis of the policies.

In *Chapter 6* we study a scheduling problem from the perspective of a single vessel occurring in inland waterway management. We define the problem of finding an optimal velocity policy for an individual vessel which faces uncertainty at the lock with the goal of minimizing the expected fuel consumption for traversing an inland waterway split into two segments by a single lock between them. To solve this problem, we first develop a dynamic program to find a near-optimal velocity policy. In addition, we introduce a simple class of policies based on the idea of a fixed arrival time at the lock and describe how to find the best policy among this class. We analyse these two techniques in a computational study by comparing the fuel consumption of the two considered policy to a naive policy ignoring the uncertainty at the lock. The computational study shows, that taking into account uncertainty is vital and significant fuel savings can be achieved by both policies. In fact, we show that the simple policy choosing a fixed arrival time already leads to a significant share of the possible fuel savings due to a (near-)optimal velocity policy.

Many interesting avenues for future research exist based on the problem and results in this chapter. First of all, analysing the theoretical performance of simple velocity policies such as the fixed arrival policy may be of interest to establish theoretical performance guarantees. Furthermore, extending the model to more general settings such as a sequence of multiple locks or multiple vessels approaching the lock is a next step towards a better understanding and implementation of inland waterway transportation.

Impact Paragraph

This thesis investigates different optimization problems in the field of scheduling. Scheduling problems model situations in which limited resources have to be assigned to tasks over time as to minimize costs, maximize profits, balance workloads among resources or improve the efficiency of the usage of resources. The theory of scheduling is concerned with modelling simplified problems to gain a better understanding of the intrinsic structures and challenges faced when solving these problems. The goal is to develop algorithms which solve problems to optimality in an efficient amount of time. However, many scheduling problems turn out to be computationally hard such that we cannot hope to accomplish this goal. To overcome this, we consider approximation algorithms which offer a trade-off by computing solutions provably close to an optimal solution in an efficient amount of time. Approximation algorithms for simplified theoretical problems offer building blocks towards heuristics for applied problems with additional practical constraints. Furthermore, approximation algorithms guarantee that the possible loss compared to an optimal solution is guaranteed to be bounded from above. This is highly relevant as non-optimal solutions negatively affect costs, profits, fairness, customer satisfaction and other standard goals.

In *Chapter 2* we investigate load balancing problems on identical parallel machines. These problems model situations in which resources are used most efficiently whenever the workload is distributed fairly. While most of the approximation algorithms known for these problems follow the multiplicative approximation paradigm, we formalize the concept of additive approximation schemes. This concept may help future research to overcome challenges faced by multiplicative approximation algorithms. From a theoretical perspective, additive approximation schemes even lead to tighter guarantees when the chosen approximation parameter is much smaller than the value of the optimal solution. For practitioners, additive approximation guarantees allow for a quantification of the possible loss compared to an optimal solution and an economic or social cost analysis of the trade-off between closeness to the optimal solution and running time.

Chapter 3 extends the concept of machine conflict graphs to model situations in which machines require access to a server before and after processing jobs. This server, however, can only be accessed simultaneously by machines which are not in conflict with each other. The results presented in this chapter provide first building blocks towards a better understanding of such machine environments and may spark interdisciplinary research between the fields of scheduling and graph theory. The algorithms can be applied by any decision maker facing a machine environment with conflicts. The algorithm for bipartite conflict graphs and unit jobs can be used as a foundation for heuristics on other conflict graphs.

In *Chapter 4*, a just-in-time scheduling problem is investigated. Just-in-time scheduling is highly relevant in many applications where early as well as late completion of jobs is undesirable due to high storage costs, perishable products or customer satisfaction. Quadratic penalties model situations in which small deviations from due dates may not be desirable but should not be penalized as much as large deviations, e.g., when handling perishable products or aiming for customer satisfaction. The methods devised in this chapter can be applied in situations when the duration of all tasks are the same, e.g., in service industries when all jobs are essentially the same but the importance and due date is customer specific.

Chapter 5 investigates techniques to cope with uncertainty in scheduling problems. Uncertainty plays a significant role in practice as decision makers in manufacturing or service industries often face uncertainty or randomness such as the availability or duration of jobs. Stochastic online scheduling addresses these sources of uncertainty in order to provide solution techniques to practitioners which accurately cope with the uncertainty underlying the problem. In this chapter, we contribute to this line of research by devising and analysing stochastic online scheduling policies for a problem in the uniform parallel machine environment which is relevant in applications such as manufacturing, computing and compiler optimization. From a practical perspective, these policies allow decision makers to accurately address and solve problems with underlying uncertainties. Furthermore, the policies can be used as essential building blocks for heuristic techniques taking into account additional practical restrictions as well. As our computational study shows, the policies can be expected to perform better in practice than we proved theoretically. Moreover, from a theoretical perspective the analysis of the algorithms improves previous known performance guarantees for some special cases.

In *Chapter 6*, we introduce a model and solution techniques to address one of the key aspects and drivers towards green and sustainable transportation in Europe. Inland waterways offer a great opportunity as one of the most sustainable and CO_2-efficient means of transportation of goods. To utilize the full potential of inland waterways, mathematical optimization models and techniques such as the one introduced in our work are necessary to reach the goal of sustainable logistics. An important factor that we consider is uncertainty due to different types of vessels, weather and other factors. We introduce multiple techniques for skippers to reduce fuel consumption, emissions and cost in an uncertain environment leading to a sustainable and efficient transportation system in an uncertain environment. Most significantly, we show that even simple techniques which take into account uncertainty strongly outperform techniques which do not take into account this uncertainty at all. This is relevant for both practitioners and theoreticians. For skippers, this provides attractive and applicable methods to reach significant fuel savings and also underlines the necessity of such methods in order to reach a sustainable transportation system. Moreover, from a theoretical perspective this opens up the opportunity towards further considering these simple types of velocity policies to investigate the theoretical performance and use them as a foundation for solution methods to extensions of the considered model.

About the author

Moritz Yannik Buchem was born on June 13, 1994 in Düren-Birkesdorf. In 2013, he received his Abitur from the Gymnasium am Wirteltor in Düren. Afterwards, he obtained his BSc degree in Econometrics and Operations Research in 2016, and his MSc degree in Business Research specializing in Operations Research in 2018, both at Maastricht University. After his studies, he joined the department of Quantitative Economics at Maastricht University as a PhD student under the supervision of Prof. dr. Tjark Vredeveld and Dr. Tim Oosterwijk. The findings of the research conducted during his time as a PhD candidate are partly published and have been presented at various international conferences and workshops. In September 2022, Moritz joined the Combinatorial Optimization group of the Technische Universität München as a postdoctoral researcher with Prof. dr. Andreas Wiese.