Unrelated Machine Scheduling in Different Information Models

vorgelegt von

Alexander Lindermayr

vom Fachbereich 3 – Mathematik und Informatik
der Universität Bremen
zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften
Dr. rer. nat.

genehmigte Dissertation

Gutachter*innen: Prof. Dr. Nicole Megow
Universität Bremen

Prof. Benjamin Moseley, PhD
Carnegie Mellon University

Tag der wissenschaftlichen Aussprache: 22. November 2024

Bremen, November 2024

Acknowledgments

There are so many people who joined and supported me on my journey towards this thesis, and I thank you all for making it possible.

In particular, I would like to thank Nicole for being an incredible advisor: Awakening my interest in research and combinatorial optimization, setting the right accents at the right time, giving me the freedom to explore and initiate different research directions, but always paying attention that I do not drift away, and always being available for questions about research, academia, and life. I would also like to thank Sebastian for giving me an early opportunity to do research during my master studies, and to finally rejoin my journey as a committee member. Further, a big thanks goes to Ben for taking the second assessment of this thesis.

I thank all my colleagues in Bremen for creating such an active, enjoyable, fun, and relaxing environment to which I was always looking forward coming: Bart, Daniel, Elias, Felix, Franzi, Jens, Lukas, Mohit, Nicole, Nicole, Nikolas, Torben, Vijay, Zhenwei, and many more. Special thanks go to Felix and Jens: To Felix for our fast and fun road bike tours, which sometimes also ended up in research sessions on the road, and to Jens for being the best office mate, research collaborator, travel companion (especially during 30 hour flights), and expert on office decorations such as Hawaiian gods on ancient Greek pillars.

Further, I am also very grateful to all my friends both in Bremen and in Bavaria who have always supported me. Finally, I want to thank my parents for their never-ending love, support, engagement, and patience.

Bremen, November 2024 Alexander Lindermayr

Contents

Chapter 1

Introduction

Imagine that you are the manager of a translation company. At the beginning of each month, you receive books that need to be translated. You have a team of translators and each translator has a different expertise for each language and genre. This means that they translate different numbers of pages per hour, and most importantly, these translation speeds can be completely *unrelated* for each translator and each book. Your task is to *schedule* the translation of each book by assigning books to translators over different time periods. Since you want to keep your customers, your goal is to *complete* the translations of the books as quickly as possible. More specifically, your goal is to minimize the *average* time it takes to complete the translation of a book each month.

Now, imagine that you are a physicist. For your latest simulation, you need to perform various calculations. Your university has computers with different hardware accelerators available. Since your calculations benefit very differently from each kind of hardware acceleration, each calculation requires a different duration to complete on every computer, and these durations can be completely *unrelated* for each calculation and each computer. Your task is to find a *schedule* by assigning each calculation to one of the computers. Since you want to examine the results of your simulation as quickly as possible, your goal is to minimize the time required to complete all calculations.

These are just two examples of *scheduling problems* that arise in our everyday lives. In this thesis, we study these and related scheduling problems. Since good *strategies* for the above two examples can be very similar, it would be redundant to study each of these *applications* on their own. Instead, we abstract from all details that are specific to an application and focus on the essential information required to solve the underlying actual scheduling problem. Then, we can apply these abstract results to concrete applications.

We now give a first look at this abstract view. In a scheduling problem, the task is to assign *jobs* (books, calculations) to time windows on shared resources (translators, computers) to *process* them. We call these resources *machines* and such an assignment a *schedule*. A machine can process at most one job at any time, and a job can be processed by at most one machine at any time. Every job has a *processing requirement* (number of pages of a book) and is processed at a *speed* (pages per hour) on each machine. Equivalently, every job requires a different time to finish if only processed by one machine, that is, the *processing time* (duration of the physicist's calculation) on that machine. For example, if a book has 100 pages (the processing requirement of the job) and a translator translates 5 pages per hour (the speed of the job on the machine), it will take them 20 hours to complete the book (the processing time of the job on the machine). The speeds and thus processing times can be entirely unrelated for each machine and each job.

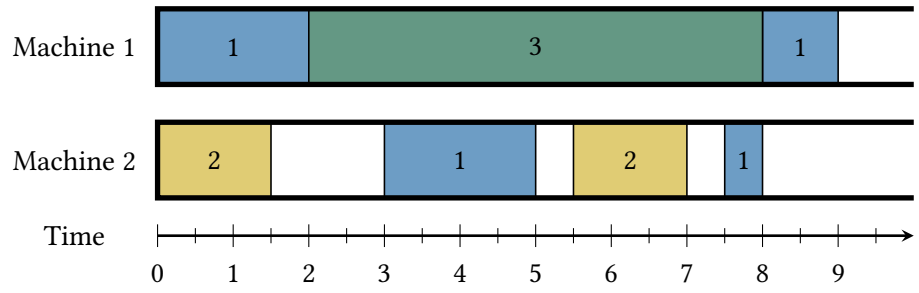


Figure 1.1: A feasible preemptive and migratory schedule with two machines and three jobs. Job 1 (blue) has a processing requirement of 14, a speed of 3 on Machine 1, and a speed of 2 on Machine 2. It migrates from Machine 1 to Machine 2 and later back to Machine 1. Job 1 is completed at time 9 because it runs for $2 + 1$ time units on Machine 1 and $2+0.5$ time units on Machine 2, and thus, it receives a total processing equal to $3 \cdot 3+2 \cdot 2.5 = 14$. Machine 2 preempts Job 2 (yellow) at time 1.5 and continues it at time 5.5. Job 3 (green) has a processing time of 6 on Machine 1. The makespan of the schedule is 9 and the average completion time is $\frac{1}{3}(7 + 8 + 9) = 8$.

We call the collection of these quantities an *instance* of a scheduling problem. A job *is completed* when it has received as much processing as required (all pages of a book are translated), and the point in time when this happens in the schedule is the *completion time* of that job.

Going back to the initial examples, there are additional restrictions on schedules in some applications. For example, each of the physicist's calculations must not be interrupted, and thus, has to be scheduled in a single continuous time interval on one computer. We call such schedules *non-preemptive*. Otherwise, if a machine stops working on an unfinished job, we say that the schedule is *preemptive*. For the translation company, we may want to allow preemptive schedules because every translator needs some breaks, or a translator may work on more than one book at a time, switching between them from time to time. Certain books may need to be entirely translated by one translator to maintain a consistent style. If this is the case for every job, we call the schedule *non-migratory*. Otherwise, if a job can be processed by multiple machines, we say that the schedule is *migratory*. We will give precise definitions of schedules and such properties at the beginning of this thesis. We call a schedule *feasible* if it completes all jobs and satisfies the requirements of the problem, for example, that it is preemptive and non-migratory. Figure 1.1 shows an example of a feasible schedule.

We study scheduling problems with an optimization objective. In an optimization problem, the task is to find a solution for a given instance that optimizes some numerical objective among all feasible solutions for that instance. This means that every solution has an associated numeric objective value, and a feasible solution is *optimal* if there is no other feasible solution with a *better* objective value, that is, a *smaller* or *larger* objective value, depending on whether we want to *minimize* or *maximize* the objective. For an instance I, we usually denote the optimal objective value by $\mathrm{OPT}(I)$. This optimal objective value may be obtained by multiple distinct optimal solutions. In this thesis, we mainly study two *minimization* objectives for scheduling problems: the maximum completion time, also called *makespan* (the time when the physicist's simulation is ready), and the *average of the weighted completion times* (the average

book completion time). In the latter, every job may have an associated numerical *weight* that can model different priorities among the jobs. Our goal is the design and analysis of an *algorithm* that, for a scheduling problem and any possible instance of that problem, computes a feasible schedule. For a fixed problem, a fixed deterministic algorithm, and an instance I of that problem, we usually denote by $\mathrm{ALG}(I)$ the objective value of the schedule computed by the algorithm for I.

One might ask why we do not just compute an optimal solution by considering every feasible schedule and choosing the best one. While, in principle, this is possible for some problems, there are various reasons why an algorithm "cannot" compute an optimal solution for every instance of the problem.

One reason is that the problem is so inherently complex that for a large instance, for example, the physicist has millions of calculations and there are thousands of computers, an algorithm, even if executed on the fastest supercomputer[1] available today, requires more time than the current age of the universe. For some problems, we can even prove that *every* algorithm for the problem requires such an amount of time for some instance. This is backed up by the famous $\mathsf{P} \neq \mathsf{NP}$ conjecture, which we briefly discuss at the beginning of this thesis. Thus, for such hard problems, there is only little hope that we can design an algorithm that is fast with respect to the *size* of the instance. From an application's perspective, however, having *near* optimal solutions that can be computed fast is often sufficient. This motivates the idea of *approximation algorithms*, which *guarantee* to output good but potentially non-optimal solutions. For a minimization problem, we say that an algorithm is an α-approximation if, for every instance I, it computes a solution with an objective value of at most $\alpha \cdot \mathrm{OPT}(I)$, for some $\alpha \geq 1$. We call the smallest $\alpha \geq 1$ such that the algorithm is an α-approximation its *approximation ratio*, which can also be translated to maximization problems. The goal for such hard problems is to find approximation algorithms with provably small approximation ratios that compute solutions reasonably fast. This latter requirement is defined by a *running time complexity class*, which gives a bound on the time an algorithm is allowed to use to compute a solution for an instance of a specific size. In this thesis, we mostly consider a class that gives a bound that is *polynomial* in the size of the input.

Until now, we have assumed that all *information* of an instance (number of books, pages per book, processing times of calculations on computers, etc.) is available to an algorithm. In such a case, we call the problem an *offline* problem. We consider offline scheduling problems and approximation algorithms in the first part of this thesis.

Another reason an algorithm cannot compute an optimal solution is *uncertainty* about the instance. That is, parts of the instance are initially unknown to the algorithm, but they arrive while the algorithm is solving the instance, that is, scheduling jobs. In this case, we say that a problem is an *online* problem, and an algorithm that solves it is an *online* algorithm. In this thesis, we mainly consider problems where the instance arrives *over time*, and an online algorithm cannot revoke earlier decisions. In the translation company, that could happen if an author submits their book later than expected, say on the 5th of the month, and only when the book arrives, important information such as genre, language, and page number become known. Note that when this happens, we cannot revoke our schedule for the first four days of

[1] We exclude quantum computers here.

the month, as those days and their schedules have already happened. We say in such a case that jobs *arrive online over time*. Another online scenario is when an author sends the next page to the translation company only after the previous page has been translated. The reason for this may be that the authors want to identify quality issues early, or to avoid leaks of the plot twists at the end of their books. We call online scheduling problems (and algorithms) *non-clairvoyant* if the processing requirement of a job remains unknown until the job is completed. Otherwise, the problem and the algorithm are called *clairvoyant*. These are just two examples of online variants that we consider in this thesis.

Intuitively, if an online algorithm wants to compute a good solution, it must be prepared for *every* possible realization of the future that the currently uncertain information will determine, for example, whether more jobs will arrive soon. Thus, it must make defensive and potentially non-optimal decisions at the beginning of the instance. This intuition is the key to many proofs showing that no online algorithm can compute an optimal solution for every instance of some problem. Similar to the offline setting, we can still design online algorithms that compute good approximate solutions. We say that an online algorithm is *α-competitive* for a minimization problem if, for any instance I of the problem, it computes a solution with an objective value of at most $\alpha \cdot \mathrm{OPT}(I)$, for some $\alpha \geq 1$. We call the smallest value $\alpha \geq 1$ such that the algorithm is α-competitive its *competitive ratio*, which can also be translated to maximization problems. Similar to approximation algorithms for offline problems, the goal in the design and analysis of online algorithms is to provide algorithms with small competitive ratios, or to prove that certain competitive ratios cannot be achieved for a problem. In the second part of this thesis, we consider online scheduling problems.

The approximation ratio and the competitive ratio are *worst-case guarantees*: They ensure that regardless of the instance we give to the algorithm, it will output a solution with an objective value of at most α times the optimal objective value. However, this can be quite different from the *average* performance that is relevant for many practical applications. Indeed, an online algorithm has a large competitive ratio if it solves all instances optimally except one, the *worst-case* instance, where it performs poorly. One reason for strong theoretical worst-case instances for online algorithms is the pessimistic assumption that *absolutely no* information about the uncertain future is known.

Going back to the two types of uncertainty identified for the translation company, we can see that some additional information may be available. First, we may know from experience that some authors are more likely to submit their books late than others. Also, we may reject books submitted after the 10th of the month. This information gives an algorithm a smaller set of possibilities of when an author's book will arrive if they are late. Second, if an author submits their book page by page, we can estimate a lower and upper bound on the total number of pages from other books by that author, or as we read each page, we can estimate from the story how far we are from the end. Most of this additional information is rather an *estimate* or a *prediction*, which we generated from experience and historical data. In some larger applications, we could also imagine that such predictions are generated from a machine-learned model. Specifically, we should not expect that they match the ground truth. The type of information that the predictions give is defined by the *prediction model*. If these predictions are reasonably *accurate*, they can help an online algorithm to overcome pessimistic worst-case instances. We call such algorithms that use possibly imperfect predictions *learning-augmented algorithms* or

algorithms with predictions. We study learning-augmented algorithms for online scheduling problems in the third (and final) part of this thesis.

1.1 Outline

In this thesis, we study unrelated machine scheduling and related problems under the three information models mentioned above: We start with offline scheduling problems, where an algorithm knows all information about an instance. Afterwards, we consider various online scheduling problems, where we gradually increase the level of uncertainty via online job arrival and non-clairvoyance. Finally, we study scheduling problems in the learning-augmented framework that lie between the offline and online settings. As a final part of this introduction, we provide a more detailed overview of the chapters contained in these parts. Before we start with the first part, we give formal definitions and other preliminaries in Chapter 2.

Part I – Offline Scheduling

Chapter 3 – Santa Claus and Makespan Minimization: In this first chapter, we study the relationship between two fundamental scheduling problems in terms of their approximability. One problem is minimizing the makespan on unrelated machines without preemption, which corresponds to the physicist's problem. The other problem is the closely related Santa Claus problem, where, compared to the makespan minimization problem, the goal is, roughly, to maximize the minimum completion time. We prove approximation-preserving reductions between both problems, thereby improving our understanding of their difficulty in terms of approximation algorithms. Moreover, we introduce, study, and utilize new generalizations of both problems and give improved approximation algorithms for special cases. This chapter is based on joint work with Étienne Bamas, Nicole Megow, Lars Rohwedder, and Jens Schlöter [Bam+24].

Part II – Online Scheduling

Chapter 4 – Clairvoyant Online Scheduling: We then turn to online scheduling. We first consider unrelated machine scheduling with preemption in the online-time model where jobs arrive online over time with the goal of minimizing the average weighted completion time. We introduce and study two natural and fast algorithms that are based on a simple well-known preemptive scheduling rule. More specifically, one algorithm tackles the non-migratory problem, for which we prove a competitive ratio of at most 5.83, and the other algorithm tackles the migratory problem, for which we prove a competitive ratio of at most 7.24. These results are based on joint work with Nicole Megow and Martin Rapp [LM22; LMR23].

Chapter 5 – Non-Clairvoyant Online Scheduling: We add more uncertainty in the form of non-clairvoyance to the online scheduling problem of the previous chapter. We study the Proportional Fairness algorithm, a natural and fundamental resource allocation rule from economics, for unrelated machine scheduling and a generalization of it. We drastically improve on the previous analysis of this algorithm and give the currently best known

bound of 4.62 on the polynomial-time competitive ratio for preemptive and migratory online scheduling on unrelated machines, even among clairvoyant algorithms. This chapter is based on joint work with Sven Jäger and Nicole Megow [JLM25].

Part III – Learning-Augmented Scheduling

Chapter 6 – Introduction to Learning-Augmented Algorithms: Finally, in the last part of the thesis, we study learning-augmented algorithms for online scheduling problems. In this chapter, we provide a more detailed overview of learning-augmented algorithms and related work.

Chapter 7 – Permutation Predictions for Unknown Processing Times: We introduce a novel prediction model for non-clairvoyant scheduling. Previous works studied prediction models that focus on predicting the unknown processing requirements of the jobs. We propose a more compact prediction model that focuses on essential information to achieve a small objective value. That is, a permutation (or total order) of the jobs in that an optimal solution schedules them on a single machine. We show that permutation predictions have desirable properties, such as being learnable, and admit learning-augmented algorithms with good guarantees. This chapter is based on joint work with Nicole Megow [LM22].

Chapter 8 – Predictions for Unknown Precedence Constraints: We add further requirements to feasible schedules in the form of precedence constraints: certain jobs are only allowed to start when others have been completed. Such constraints could be present in the physicist's simulation, where certain calculations require the results of other calculations. We study precedence-constrained scheduling problems in an online variant, where an algorithm is unaware of these relationships, and new jobs appear only when their predecessors have been completed. We introduce and compare several prediction models and algorithms for this notoriously difficult online problem. This chapter is based on joint work with Alexandra Lassota, Nicole Megow, and Jens Schlöter [Las+23].

Chapter 9 – Predictions for Unknown Processing Speeds: In this chapter, we address a discrepancy between theory and practice: While in scheduling theory we usually assume that the speeds at which the machines process jobs are known, this is usually not the case when actually running such algorithms in practice. For example, the physicist may know how many instructions are needed to complete each calculation. However, it is not upfront clear how much time each computer requires for these, as this may depend on other tasks being executed on that computer in parallel by another user. Similarly, it is not realistic that a translator can consistently translate the same number of pages per hour for many hours. While we show that the lack of information about the processing speeds rules out algorithms with a constant competitive ratio, we introduce new information / prediction models and applicable algorithms that mitigate these issues. This chapter is based on joint work with Nicole Megow and Martin Rapp [LMR23].

Chapter 10 – Predictions for Uncertain Jobs in Online TSP: In this very last chapter of the thesis, we study a slightly different scheduling problem than the previous ones. In the Online Traveling Salesperson Problem (Online TSP or OLTSP), a *server* (the machine)

moves at unit speed in a metric space. *Requests* (jobs) arrive online over time at points in the space. The goal is to route the server over time through all requests and back to its starting position as quickly as possible, that is, to minimize the makespan. To see the connection to unrelated machine scheduling, one can view the server as a moving machine, and the processing time of a request depends on its location and the machine's current location. We consider this online problem in the learning-augmented framework, where the predicted locations and arrival times of requests are given to an algorithm in advance. This chapter is based on joint work with Giulia Bernardini, Alberto Marchetti-Spaccamela, Nicole Megow, Leen Stougie, and Michelle Sweering [Ber+22a].

Chapter 2

Preliminaries

In this chapter, we formally introduce problems, concepts, and definitions that we use throughout this thesis. We assume some basic knowledge of combinatorial optimization, and refer to the books by Schrijver [Sch03] and Korte and Vygen [KV18] for an introduction and overview. For an introduction and overview of scheduling theory, we refer to the book by Pinedo [Pin22].

We use standard mathematical notation. For an positive integer k, we denote by $[k]$ the set $\{1, \dots, k\}$. For a real number x, we denote by x_+ its positive part, that is, $x_+ = x$ if $x \geq 0$ and $x_+ = 0$ if $x < 0$. For a list or set of elements $x_1, x_2, \dots$ indexed by a set I, we denote this list by $(x_i)_{i \in I}$, and, if clear from the context, we write x instead of $(x_i)_{i \in I}$. All logarithms are to base 2 unless explicitly stated otherwise.

2.1 Complexity Theory

We give a high-level overview of basic concepts and definitions in complexity theory. This introduction is based on [Sch03]. We refer to [AB09] for an in-depth overview of complexity theory.

A *deterministic algorithm* is a finite set of instructions that modify data stored in an array of bits. Each instruction can read a fixed predefined number of entries of the list, perform arithmetic operations on it, and store the result in a prescribed position of the array. The initial state of the array is given by the *input*. If the algorithm terminates, the *output* can be extracted from the current state of the array. The *size* of the input is the length of the initial array, that is, the number of bits required to represent it. For example, the input size of a positive integer n is the length of its binary encoding, which is $\lfloor \log n \rfloor + 1$.

A deterministic algorithm runs in *polynomial time* if it terminates after a number of instruction steps that is polynomial in the input size; we say that the algorithm is a *polynomial-time* algorithm. A problem is *polynomial-time* solvable (or can be solved in polynomial time) if there exists a polynomial-time algorithm that solves it.

We can now define the complexity classes P and NP, which are a set of decision problems. Intuitively, a *decision problem* is a question that can be answered with a binary label such as "YES" or "NO". Let Σ be an *alphabet* of size at least 2, which in its simplest form can be $\{0, 1\}$, and let Σ^* denote the set of finite strings, called *words*, which can be built from letters of Σ. The size $\text{size}(x)$ of a word x is its length. A *problem* Π is a subset of Σ^*. In a *decision problem* Π, the task is to decide, for a given word $x \in \Sigma^*$ (the *instance*), whether $x \in \Pi$. From an algorithmic point of view, we can think of x as the input. A decision problem Π over the words Σ^* is called polynomial-time solvable if there exists an algorithm that decides for every word $x \in \Sigma^*$

whether $x \in \Pi$ in time polynomial in size(x). The set of polynomial-time solvable problems is denoted by P.

We now move to the class NP. Intuitively, a decision problem is in NP if for every input with answer "YES", there exists a *certificate* that can be *verified* by a polynomial-time algorithm. Formally, a problem Π is in NP if there is a problem $\Pi' \in \mathsf{P}$ and polynomial function $p : \mathbb{N} \to \mathbb{N}$ such that, for every $w \in \Sigma^*$, it holds that $w \in \Pi$ if and only if there exists a word $x \in \Sigma^*$, the certificate, of size at most $p(\text{size}(w))$ such that $wx \in \Pi'$, where wx denotes the concatenation of w and x.

While it clearly holds that $\mathsf{P} \subseteq \mathsf{NP}$, since we can select $\Pi' = \Pi$ and an empty certificate in the definition of NP, one of the biggest open questions in computer science and mathematics is whether $\mathsf{P} = \mathsf{NP}$. An important role in the relationship of both classes is the concept of NP-*complete* decision problems. To define them, we say that a problem $\Pi \subseteq \Sigma^*$ can be *reduced* to a problem $\Pi' \subseteq \Sigma^*$ if there exists a polynomial-time algorithm that outputs for every input $x \in \Sigma^*$ an output $x' \in \Sigma^*$ such that $x \in \Pi$ if and only if $x' \in \Pi'$. A decision problem Π is NP-*hard* if every problem in NP can be reduced to Π. A decision problem Π is NP-*complete* if it is NP-hard and $\Pi \in \mathsf{NP}$. This definition implies that if there is an NP-complete problem that is also in P, then $\mathsf{P} = \mathsf{NP}$. Intuitively, NP-complete problems are the "hardest" problems in NP. In this thesis, we make the common assumption that $\mathsf{P} \neq \mathsf{NP}$.

Optimization problems can usually be transformed into decision problems: Given an optimization problems which seeks to minimize a rational-valued function $f(y)$ over all $y \in Y$, we can, for any rational number k, define a decision problem which asks whether there exists a $y \in Y$ such that $f(y) \leq k$. Then, given a polynomial-time algorithm for the decision problem, we can usually derive a polynomial-time algorithm for the optimization problem by executing a binary search over values for k. Thus, we say that an optimization problem is in P if the underlying decision problem is in P. Similarly, we can extend the definition of NP-hardness to optimization problems.

Finally, we end this section with a few additional definitions and properties that appear throughout this thesis. Another relevant complexity class of optimization problems is APX. While we omit its formal definition, we note that for an APX-*hard* optimization problem, there exists a constant $c > 1$ such that there exists no polynomial-time c-approximation algorithm, unless $\mathsf{P} = \mathsf{NP}$. An algorithm runs in *quasi-polynomial time* for a problem if there is a $c \in \mathbb{N}$ such that for every input of size n it terminates after at most $2^{O(\log^c n)}$ steps. An algorithm runs in *pseudo-polynomial time* for a problem if, for every input, it terminates after polynomially many steps in the size of the input and the magnitude of the largest numeric value of the input. If a decision problem is *strongly* NP-hard, then there exists no pseudo-polynomial-time algorithm that solves it.

2.2 Unrelated Machine Scheduling

Many parts of this thesis consider variants of the problem of scheduling jobs on unrelated machines. In this section, we give a formal definition of this problem and its variants.

2.2.1 Definitions and Notation

We first introduce the problem of scheduling jobs on unrelated machines and its variants. We are given a set of *n jobs* J. We usually assume that $J = [n]$. Every job $j \in J$ is associated with a *weight* $w_j \geq 0$, a *release date* $r_j \geq 0$, and a *processing requirement* $p_j \geq 0$. We say that the problem has uniform release dates if $r_j = r$ for all $j \in J$, in which case we can assume without loss of generality in this thesis that $r_j = 0$ for all jobs j. Otherwise, the problem has *non-uniform* release dates; we say that the problem is with release dates.

Unrelated Machines. The *unrelated machine environment* is defined by a set of m *unrelated machines* M (we usually assume that $M = [m]$), and every machine $i \in M$ has an associated *speed* $s_{ij} \geq 0$ at which every job $j \in J$ runs on machine i. This means that if we schedule job j on machine i for one time unit, it receives a *processing rate* equal to s_{ij}. For every machine $i \in M$ and job $j \in J$, we denote with $p_{ij} \coloneqq p_j/s_{ij}$ the *processing time* of j on machine i, that is, the total time required for j to complete if it is continuously processed only on machine i.

Important and well-studied special cases of unrelated machines are *related* machines where, on every machine i, the speed $s_i = s_{ij}$ is the same for every job j, *restricted assignment* where all $s_{ij} \in \{0, 1\}$, *identical parallel machines*, where all $s_{ij} = 1$, and the *single machine* where $m = 1$ and all $s_{ij} = 1$.

Schedules and Completion Times. A *schedule* is described by binary variables $x_{ij}(t)$ that indicate whether job j is being processed on machine i at time $t \in \mathbb{R}_{\geq 0}$. We say that a machine i *idles* at time t if $x_{ij}(t) = 0$ for all jobs j. A schedule is *feasible* if at any time instant t, every job j runs on at most one machine, that is, $\sum_{i \in M} x_{ij}(t) \leq 1$, and every machine $i \in M$ is occupied by at most one job, that is, $\sum_{j \in J} x_{ij}(t) \leq 1$. Moreover, a job can only be processed if it has been released, that is, the schedule must satisfy $x_{ij}(t) = 0$ for all $i \in M$ and all times $t < r_j$ for every job j. In the following, we only refer to feasible schedules when considering schedules. We denote by

$$y_j(t) \coloneqq \sum_{i \in M} s_{ij} \cdot x_{ij}(t)$$

the amount of processing that j receives at time t, which we also call *processing rate*. The completion time C_j of job j in a given schedule is the earliest point in time when j received as much processing as required. Formally, that is

$$C_j = \arg\min_{t \geq 0} \left(\int_0^t y_j(t') \, \mathrm{d}t' \geq p_j \right) .$$

We denote by $C_{\max} \coloneqq \max_{j \in J} C_j$ the latest completion time in the schedule. This quantity is also called the *makespan* of the schedule.

Preemption and Migration. A schedule is called *non-migratory* if for every job j there exists a machine i such that $x_{i'j}(t) = 0$ for every machine $i' \in M \setminus \{i\}$ and every time $t \in \mathbb{R}_{\geq 0}$, that is, a job is only being processed on one machine. Otherwise, the schedule is called *migratory*. In this case, we say that migration is allowed. Furthermore, a schedule is called *non-preemptive* if it is non-migratory and the support of $\sum_{i \in M} x_{ij}(t)$ as a function of t is a continuous interval,

that is, a job is never interrupted after being started with processing. Otherwise, the schedule is called *preemptive*, and we say that preemption is allowed. Specifically, every migratory schedule is preemptive.

Scheduling Problems and 3-Field Notation. We use the 3-field notation by Graham et al. [Gra+79] to characterize scheduling problems. A scheduling problem is denoted by $\alpha \mid \beta \mid \gamma$.

- The first field α describes the machine environment. We use $\alpha = R$ for unrelated machines, $\alpha = Q$ for related machines, $\alpha = P$ for parallel identical machines, and $\alpha = 1$ for the single machine.
- The second field β describes further characteristics of the jobs and a feasible schedule. If preemption and migration are allowed, we add "pmtn" to β. If preemption is allowed but migration is not, we add "pmtn, non-mig" to β. If there are non-uniform release dates, we add "r_j" to β.
- The third field γ describes the objective function of our optimization problem. In this thesis, we use $\gamma = C_{\max}$, that is, minimizing the makespan, $\gamma = \sum C_j$, that is, minimizing the sum of completion times, and $\gamma = \sum w_j C_j$, that is, minimizing the sum of weighted completion times. We call the two latter objectives sometimes also *min-sum* objectives. Note that it is equivalent whether to minimize the *sum* or the *average* of the (weighted) completion times.

To denote scheduling with restricted assignment, we use $\alpha = R$ and add "$s_{ij} \in \{0, 1\}$" to β.

More Notation. Given a fixed schedule, we write $p_j(t) \coloneqq p_j - \int_0^t y_j(t')\,\mathrm{d}t'$ for the total remaining processing requirement at time t, and $p_{ij}(t) \coloneqq p_j(t)/s_{ij}$ for all jobs j and machines i. Furthermore, we introduce the notation $J(t) \coloneqq \{j \in J \mid C_j > t \geq r_j\}$ for the set of *available jobs*, $U(t) \coloneqq \{j \in J \mid C_j > t\}$ for the set of *unfinished jobs* at time t, and $W(t) \coloneqq \sum_{j \in U(t)} w_j$ for their total weight.

Time Discretization. We make the common assumption that all release dates r_j, processing requirements p_j, weights w_j, and speeds s_{ij} are rational numbers. Then, given a schedule with a rational allocation x, we can assume by scaling that the schedule preempts, migrates, and completes jobs only at integer times. Moreover, we can conclude that $\sum_{j \in J} w_j C_j = \sum_{t \geq 0} W(t)$ because every job j contributes w_j to the objective value at every integer timeslot before C_j.

2.2.2 Offline Scheduling

In the offline setting, an algorithm knows all information about an instance. We present an overview of basic complexity results for scheduling jobs on unrelated machines with the objective of minimizing the total weighted completion time. Related work on offline scheduling problems with makespan objective is deferred to Chapter 3.

The arguably most famous result for this problem is the Weighted-Shortest-Processing-Time (WSPT) rule, which is also called Smith's rule. It states that an optimal schedule for $1 \mid\mid \sum w_j C_j$ is given by scheduling the jobs without interruptions in order of non-increasing w_j/p_j. This

ratio is also called a job's *density*. Moreover, every optimal schedule for this problem satisfies this property.

Theorem 2.1 (Smith [Smi56])**.** *For the problem* $1 \mid\mid \sum w_j C_j$*, a schedule is optimal if and only if it schedules the jobs one-by-one without interruptions and for all jobs* j' *and* j *it holds that* j' *is scheduled before* j *if* $w_{j'}/p_{j'} \geq w_j/p_j$*.*

Furthermore, WSPT is also optimal for $1 \mid \text{pmtn} \mid \sum w_j C_j$, that is, an optimal solution for this problem may not use preemptions. In the case of unit weights, the rule becomes the Shortest-Processing-Time (SPT) rule that schedules the jobs in non-decreasing order of their length. WSPT can be implemented in polynomial time, and thus, $1 \mid (\text{pmtn}) \mid \sum w_j C_j$ is in P. Other scheduling problems that are in P are listed in the following theorem.

Theorem 2.2. *The following scheduling problems are polynomially-time solvable.*

- $1 \mid r_j, pmtn \mid \sum C_j$ *[Sch68]*
- $R \mid\mid \sum C_j$ *[BJS74; Hor73]*
- $Q \mid pmtn \mid \sum C_j$ *[Gon77]*

In contrast, the following theorem states problems that are known to be strongly NP-hard.

Theorem 2.3. *The following scheduling problems are strongly NP-hard.*

- $1 \mid r_j \mid \sum C_j$ *[LKB77]*
- $1 \mid r_j, pmtn \mid \sum w_j C_j$ *[Lab+84b]*
- $P \mid (pmtn) \mid \sum w_j C_j$ *[GJ79; McN59]*
- $P \mid r_j, pmtn \mid \sum C_j$ *[Bap+07; Bel+15; BK04]*
- $R \mid pmtn, s_{ij} \in \{0, 1\} \mid \sum C_j$ *[Sit17] (not stated explicitly but follows from the construction)*

Finally, we give in Table 2.1 an overview of the currently best-known polynomial-time approximation algorithms for some strongly NP-hard scheduling problems. To this end, we also recall the definition of approximation algorithms from the introduction.

Definition 2.4 (Approximation ratio)**.** An algorithm is a polynomial-time α-approximation for a minimization problem if, for every instance I, it computes in polynomial time a solution to I with an objective value of at most $\alpha \cdot \text{OPT}(I)$. The approximation ratio of an algorithm is the smallest α such that it is an α-approximation algorithm. Similarly, an algorithm is a polynomial-time α-approximation for a maximization problem if, for every instance I, it computes in polynomial time a solution to I with an objective value of at least $\frac{1}{\alpha} \cdot \text{OPT}(I)$, and its approximation ratio is the largest α such that it is an α-approximation algorithm.

For an overview of approximation algorithms, we refer to the book by Williamson and Shmoys [WS11].

2.2.3 Online Scheduling

In online scheduling, an algorithm does not have all information about a scheduling instance, but they arrive while the algorithm is solving the instance. In this thesis, we study the *online-time* model for scheduling problems, where an algorithm needs to continuously, at any time t, make irrevocable decisions about the schedule at time t while only having partial information

Table 2.1: Current bounds on polynomial-time approximation ratios for strongly NP-hard scheduling problems.

Problem	Bound
$P \mid r_j \mid \sum w_j C_j$	$1+\varepsilon$ [Afr+99]
$Q \mid r_j \mid \sum w_j C_j$	$1+\varepsilon$ [CK01]
$Q \mid r_j, \text{pmtn} \mid \sum w_j C_j$	$1+\varepsilon$ [CK01]
$R \mid\mid \sum w_j C_j$	$1.36+\varepsilon$ [Li25]
$R \mid r_j \mid \sum w_j C_j$	1.8786 [IL16]
$R \mid \text{pmtn} \mid \sum w_j C_j$	1.698 [Sit17]
$R \mid r_j, \text{pmtn} \mid \sum w_j C_j$	3 [Sku01]
$R \mid r_j, \text{pmtn}, \text{non-mig} \mid \sum w_j C_j$	1.99971 [IL16]

about the instance. In this thesis, we study the following two online-time models for scheduling problems.

In the first model, which is scheduling with *online job arrival*, an algorithm has at time t no knowledge about jobs j that have not been released yet, that is, jobs j for which $t < r_j$. We study such online algorithms in Chapter 4.

In the second model, which is *non-clairvoyance*, an algorithm has no knowledge about the processing requirement p_j of a job j, and only learns about it at the time when the job has been completed. We consider non-clairvoyant algorithms (and problems) in Chapter 5. If the processing requirements are revealed to the algorithm when the job becomes available, the algorithm (and problem) is called *clairvoyant*. Both models can also be combined; that is, jobs arrive online and they do not reveal their processing requirements when they arrive.

We also recall the definition of competitive ratio from the introduction.

Definition 2.5 (Competitive ratio)**.** An online algorithm is *α-competitive* for a minimization problem if, for every instance I, it computes a solution for I with an objective value of at most $\alpha \cdot \mathrm{OPT}(I)$. The *competitive ratio* of the algorithm is the smallest α for which it is α-competitive.

We give an overview of related work on online scheduling for the total weighted completion time objective in Chapters 4 and 5. For an in-depth overview of online scheduling, we refer to Pruhs et al. [PST04]. For a general introduction to online algorithms, we refer to the book by Borodin and El-Yaniv [BE98].

2.3 Linear Programming Relaxations for Min-Sum Scheduling

In this section, we give an overview of linear programming (LP) relaxations for minimizing the total weighted completion time in different machine environments, which we use in Parts II and III. We defer LP relaxations for the makespan objective to Chapter 3. For an introduction to linear programming and (general) linear optimization, we refer to the book by Bertsimas and Tsitsiklis [BT97].

Single Machine. We start by introducing an LP relaxation for the problem of preemptively minimizing the total weighted completion time on a single machine, $1 \mid \text{pmtn} \mid \sum w_j C_j$. We

have binary values x_{jt} that indicate whether job j is being processed in the integer timeslot $[t, t+1]$. To model a feasible schedule, these values must satisfy $\sum_{j \in J} x_{jt} \leq 1$ for every $t \geq 0$ and $\sum_{t \geq 0} x_{jt} = p_j$. In the following, we assume that $p_j > 0$ for every job j, as otherwise every optimal solution would finish the job at time 0. Thus, we can remove jobs j with $p_j = 0$ without changing the optimal objective value.

$$\begin{aligned}
\min \quad & \sum_{j \in J} w_j C_j^{\mathrm{LP}} && && (\mathrm{LP}_1)\\
\text{s.t.} \quad & C_j^{\mathrm{LP}} = \frac{p_j}{2} + \sum_{t \geq 0} \left(t + \frac{1}{2}\right) \frac{x_{jt}}{p_j} && \forall j \in J \\
& \sum_{t \geq 0} x_{jt} \geq p_j && \forall j \in J && (2.1)\\
& \sum_{j \in J} x_{jt} \leq 1 && \forall t \geq 0 \geq 0 && (2.2)\\
& x_{jt} \geq 0 && \forall j \in J, \forall t \geq 0 \geq 0 && (2.3)
\end{aligned}$$

This *time-indexed* LP formulation has been introduced by Dyer and Wolsey [DW90]. It uses a linear relaxation C_j^{LP} of the completion time C_j for every job j. We call the value

$$M_j = \sum_{t \geq 0} \left(t + \frac{1}{2}\right) \frac{x_{jt}}{p_j}$$

the *mean busy time* of job j in the schedule modelled by $x = (x_{jt})_{j,t}$. This is the average time at which job j is being processed, and further, if C_j is the completion time of j in a non-preemptive schedule where all completions occur at integer times, we have

$$C_j^{\mathrm{LP}} = \frac{p_j}{2} + M_j = \frac{p_j}{2} + \sum_{t=C_j-p_j}^{C_j-1} \left(t + \frac{1}{2}\right) \frac{1}{p_j} = \frac{p_j}{2} + \frac{C_j - p_j}{p_j} p_j + \frac{1}{2} + \frac{(p_j - 1)p_j}{2p_j} = C_j \,.$$

This shows that the objective value of (LP_1) of every non-preemptive solution without unnecessary idle times is equal to the actual objective value of the schedule. Further, Goemans [Goe96] showed that an optimal solution of (LP_1) is given by the the optimal non-preemptive schedule, that is, scheduling jobs in WSPT order. He proved the following theorem.

Theorem 2.6 (Goemans [Goe96]). *An optimal schedule of* (LP_1) *is equal to an optimal schedule of* $1 \mid (pmtn) \mid \sum w_j C_j$*, and they have the same objective value.*

Specifically, this shows that (LP_1) is indeed an LP relaxation for $1 \mid (\mathrm{pmtn}) \mid \sum w_j C_j$.

To conclude the discussion for a single machine, we introduce a slightly weaker LP relaxation, which we call the *mean busy time relaxation*:

$$\begin{aligned}
\min \quad & \sum_{j \in J} w_j \sum_{t \geq 0} \left(t + \frac{1}{2}\right) \frac{x_{jt}}{p_j} && (\mathrm{LP}_1^M)\\
\text{s.t.} \quad & (2.1), (2.2), (2.3)
\end{aligned}$$

Since the objective of this relaxation differs from the objective of (LP_1) by exactly the additive constant $\sum_{j\in J} w_j \frac{p_j}{2}$, we can immediately conclude that the WSPT rule also gives an optimal schedule for (LP_1^M) and thus minimizes the total weighted mean busy time for preemptive schedules. Thus, also (LP_1^M) is an LP relaxation for $1 \mid (\text{pmtn}) \mid \sum w_j C_j$.

Unrelated Machines. We move to the more general unrelated machine environment, where we additionally consider the case of non-uniform release dates. We first consider the straightforward generalization of (LP_1) to unrelated machines. The variable x_{ijt} indicates whether job j is being processed during timeslot $[t, t+1]$ on machine i.

$$\begin{aligned}
\min \quad & \sum_{j\in J} w_j C_j^{\text{LP}} && && (\text{LP}_R^{\text{nm}})\\
\text{s.t.} \quad & C_j^{\text{LP}} \geq \sum_{i\in M}\sum_{t\geq r_j} \left(t+\frac{1}{2}\right)\frac{x_{ijt}s_{ij}}{p_j} + \frac{x_{ijt}}{2} && \forall j \in J &\\
& \sum_{i\in M}\sum_{t\geq r_j} x_{ijt}s_{ij} \geq p_j && \forall j \in J && (2.4)\\
& \sum_{j\in J} x_{ijt} \leq 1 && \forall i \in M, \forall t \geq 0 && (2.5)\\
& \sum_{i\in M} x_{ijt} \leq 1 && \forall j \in J, \forall t \geq 0 && (2.6)\\
& x_{ijt} \geq 0 && \forall i \in M, \forall j \in J, \forall t \geq 0 && (2.7)
\end{aligned}$$

This LP formulation was first studied by Schulz and Skutella [SS02b]. They showed that it is a relaxation of $R \mid r_j \mid \sum w_j C_j$. Later, Jäger [Jäg21] argued that it is even a relaxation of the problem that allows preemption but not migration, $R \mid r_j, \text{pmtn}, \text{non-mig} \mid \sum w_j C_j$.

We use the common notion of *α-relaxation* to quantify the performance of a time-indexed LP relaxation [SS02b; Sku01]: We say that an LP is an *α-relaxation* of a minimization problem, if it is a relaxation of that problem and if, for every instance, the optimal objective value of the problem is at most α times the optimal objective value of the LP relaxation.

Theorem 2.7 (Schulz and Skutella [SS02b])**.** (LP_R^{nm}) *is a 2-relaxation for* $R \mid r_j \mid \sum w_j C_j$.

Moreover, (LP_R^{nm}) can be strengthened via

$$C_j^{\text{LP}} \geq \sum_{i\in M}\sum_{t\geq r_j} x_{ijt} \tag{2.8}$$

to a $\frac{3}{2}$-relaxation for $R \mid\mid \sum w_j C_j$ [SS02b].

Skutella [Sku98] observed that (LP_R^{nm}) is not a relaxation of the problem that allows migration, $R \mid r_j, \text{pmtn} \mid \sum w_j C_j$. We slightly modify his example to obtain the following slightly stronger version.

Lemma 2.8. (LP_R^{nm}) *is not a relaxation for* $R \mid r_j, pmtn \mid \sum w_j C_j$*, even if the machines are related, release dates are uniform, and* $m = 2$.

Proof. Consider an instance composed of two jobs with weights $w_1 = 2$ and $w_2 = 1$ and processing requirements $p_1 = p_2 = 4$, and two related machines with speeds $s_1 = 2$ and $s_2 = 1$.

An optimal solution schedules job 1 on machine 1 and job 2 on machine 2 until time 2. Then, job 1 completes and job 2 migrates to machine 1, where it requires one more time unit to do the remaining 2 units of processing. Thus, the optimal objective value is equal to 7. Via an exchange argument one can show that this schedule also corresponds to an optimal solution of ($\mathrm{LP}_R^{\mathrm{nm}}$), but the LP objective value is equal to $\frac{29}{4}$. Hence, it is no relaxation. □

The issue of ($\mathrm{LP}_R^{\mathrm{nm}}$) is that $\sum_{i\in M}\sum_{t\geq r_j}\left(t+\frac{1}{2}\right)x_{ijt}s_{ij}/p_j$ does not compute the actual mean busy time of job j, as it is the case on single machine, because in this expression different times can contribute with different weights s_{ij} depending on which machine job j runs on. Thus, this value might be larger than the actual mean busy time in the corresponding schedule and, therefore, C_j^{LP} overshoots the actual completion time.

Sitters [Sit17], however, showed that this phenomenon cannot be arbitrarily bad. Specifically, he proved that for the problem $R\,|\,\mathrm{pmtn}\,|\,\sum w_jC_j$, the optimal objective value of ($\mathrm{LP}_R^{\mathrm{nm}}$) is at most 1.81 times the objective value of an optimal schedule. While the proof of the following statement in [Sit17] is only given for uniform release dates, we note that it is easy to check that it also holds for non-uniform release dates.

Theorem 2.9 (Sitters [Sit17]). *For the problem $R\,|\,r_j, pmtn\,|\,\sum w_jC_j$, the optimal objective value of ($\mathrm{LP}_R^{\mathrm{nm}}$) is at most* 1.81 *times the objective value of an optimal schedule.*

Skutella [Sku98] showed another way to derive a relaxation for $R\,|\,r_j,\mathrm{pmtn}\,|\,\sum w_jC_j$. This formulation uses only the "weighted mean busy time" as relaxation of the total weighted completion time, and can be seen as the generalization of (LP_1^M) to unrelated machines:

$$\begin{aligned}\min\quad & \sum_{j\in J} w_jC_j^{\mathrm{LP}} && (\mathrm{LP}_R)\\ \text{s.t.}\quad & C_j^{\mathrm{LP}} \geq \sum_{i\in M}\sum_{t\geq r_j}\left(t+\frac{1}{2}\right)\frac{x_{ijt}s_{ij}}{p_j} && \forall j\in J\\ & (2.4),\,(2.5),\,(2.6),\,(2.7),\,(2.8)\end{aligned}$$

Theorem 2.10 (Schulz and Skutella [SS02b]). (LP_R) *is a* 3*-relaxation for $R\,|\,r_j, pmtn\,|\,\sum w_jC_j$ and a* 2*-relaxation for $R\,|\,pmtn\,|\,\sum w_jC_j$.*

Schulz and Skutella [SS02b] and Sitters [Sit17] presented approximation algorithms that produce non-preemptive schedules, and analyzed them against preemptive and migratory optimal schedules. These results imply the following corollary on the *power of preemption* on unrelated machines.

Corollary 2.11 ([Sku98, Corollary 2.10.11], [Sit17, Corollary 3]). *The optimal objective value for $R\,|\,r_j\,|\,\sum w_jC_j$ is at most* 3 *times the optimal objective value for $R\,|\,r_j, pmtn\,|\,\sum w_jC_j$. For uniform release dates this bound can be improved to* 1.81.

2.4 Dual Fitting

A traditional approach for using the optimal objective value $\mathrm{OPT}_{\mathrm{LP}}$ of an LP relaxation as a lower bound for minimization problems is first to compute an optimal LP solution and then

round this solution to a feasible integral solution. Then, we can compare the objective value of the rounded solution to the optimal LP objective value and derive a bound on the approximation ratio (see, for example, [WS11]). In the online setting, however, one cannot compute an LP solution upfront, as the instance has not been fully revealed. Therefore, there are two common techniques to compare the objective value of an online algorithm to the optimal objective value of an LP relaxation. Both are based on the fundamental property of *weak duality*, which states that if both a (minimization) linear program and its dual are feasible, then the objective value of every feasible solution of the primal program is an upper bound on the objective value of every feasible solution of its dual.

The first technique is the *primal-dual* technique. Here, both a feasible primal and dual solution are constructed simultaneously such that the objective value of the primal solution is at most ρ times the objective value of the dual solution, which immediately implies that the primal solution is a ρ-approximation via weak duality. Moreover, these solutions can be constructed and updated *online*, and decisions of an online algorithm can be guided by these updates. This technique was arguably first used by Dantzig et al. [DFF56] for solving linear programs, and it was also used by Edmonds [Edm65] for solving the maximum matching problem in general graphs. Later, there were many breakthroughs in the area of approximation algorithms using this technique; see Williamson [Wil02] for an early survey. For a survey on primal-dual for online problems, we refer to Buchbinder and Naor [BN09].

The second technique is *dual fitting*, which we mainly use in Parts II and III of this thesis. Here, we typically consider combinatorial algorithms that work independently of LP relaxations. In the analysis, we craft a feasible dual solution and compare its objective value to an algorithm's objective value ALG. Formally, we construct an assignment $\bar{a}$ of dual variables such that (i) its objective value $\mathrm{DP}(\bar{a})$ is at most $\frac{1}{\rho}\mathrm{ALG}$ and (ii) $\bar{a}$ is a feasible dual solution. Then, we can conclude via weak duality that $\mathrm{ALG} \leq \rho \cdot \mathrm{DP}(\bar{a}) \leq \rho \cdot \mathrm{OPT}_{\mathrm{LP}}$.

Jain et al. [Jai+03] arguably first formalized dual fitting and used it to analyze greedy approximation algorithms for the facility location problem. Freund and Rawitz [FR03] observed that it had already been implicitly used earlier for the set cover problem [Chv79; Lov75], a special case of facility location. They also established connections between dual fitting and the combinatorial *local ratio technique* [Bar00; BE85], which previously had been shown to be equivalent to primal-dual [BR01]. Another well-known usage of dual fitting is the analysis of online matching algorithms [BJN07; DJK13; FN21; FNS21; JM22; Meh13; Meh+07; NK13].

In the context of scheduling, Anand et al. [AGK12] and Gupta et al. [GKP12] independently introduced analyses based on dual fitting for flow time objectives with speed augmentation. Subsequently, it became one of the main tools for analyzing scheduling algorithms for the total completion time objective and more general objectives against time-indexed LPs [ALN15; Aza+15; Bha+14; Cho+18; Gar+19; GKL18; GKS21; Gup+21; IK16; IKM15a; IKM14; IKM15b; IKM18; Im+14; IM15; Jäg23; Jäg21; Jai+15; KSS21; Ngu13].

2.5 Matroids and Polymatroids

A set structure that we use in this thesis are *(poly)matroids*. In this section, we give the most basic definitions for matroids and polymatroids, and refer to [Sch03] for more details.

A *matroid* $(E, \mathcal{I})$ is composed of a ground set E and a non-empty downward-closed set system $\mathcal{I} \subseteq 2^E$ that satisfies the *augmentation property*: for all $I, J \in \mathcal{I}$ with $|I| < |J|$, there exists an element $j \in J \setminus I$ such that $I \cup \{j\} \in \mathcal{I}$. Given a matroid $\mathcal{M} = (E, \mathcal{I})$, a set $I \subseteq E$ is called *independent* if $I \in \mathcal{I}$, and *dependent* otherwise. An inclusion-wise maximal independent subset is a *basis* of $\mathcal{M}$, and we denote the set of bases by $\mathcal{B}(\mathcal{M})$. We associate a matroid $\mathcal{M} = (E, \mathcal{I})$ with a rank function $r : 2^E \to \mathbb{Z}_{\geq 0}$, where $r(X)$ describes the maximum cardinality of a subset of X that is independent in $\mathcal{M}$. One of the most basic matroids is the *uniform matroid* of rank r, in which all subsets of E of cardinality at most r are independent.

We now move to polymatroids. Let E be a ground set. For a vector $x \in \mathbb{R}^E$, we write $x(e)$ for the entry of x corresponding to $e \in E$, and $x(S) \coloneqq \sum_{e \in S} x(e)$. For some $X \subseteq E$, we write $b \cdot X$ as the vector $y \in \mathbb{Z}^E$ with $y(e) = b$ for $e \in X$ and $y(e) = 0$ for $e \notin X$. A set function $f : 2^E \to \mathbb{R}$ is *submodular* if for all subsets $A, B \subseteq E$ holds $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$, and *monotone* if for all $A \subseteq B \subseteq E$ holds $f(A) \leq f(B)$.

Given a monotone submodular set function $f : 2^E \to \mathbb{R}_{\geq 0}$ with $f(\emptyset) = 0$, the *polymatroid* over E associated with f is defined as

$$\mathcal{P} = \{x \in \mathbb{R}^E_{\geq 0} : x(S) \leq f(S)\ \forall S \subseteq E\}\ .$$

A basis of a polymatroid $\mathcal{P}$ is a vector $x \in \mathcal{P}$ that satisfies $f(E) = x(E)$. We denote the set of bases of $\mathcal{P}$ by $\mathcal{B}(\mathcal{P})$. For a given polymatroid $\mathcal{P}$ and a constant $k \in \mathbb{R}_{\geq 0}$, the set $\{x \in \mathcal{P} : x(e) \leq k\ \forall e \in E\}$ is again a polymatroid.

Given a monotone submodular integer set function $f : 2^E \to \mathbb{Z}_{\geq 0}$ with $f(\emptyset) = 0$, the *integer polymatroid* over E associated with f is defined as

$$\mathcal{P} = \{x \in \mathbb{Z}^E_{\geq 0} : x(S) \leq f(S)\ \forall S \subseteq E\}\ .$$

An integer polymatroid can be interpreted as the generalization of a matroid to multisets.

Most concepts of matroids translate easily to polymatroids. Every element $x \in \mathcal{P}$ can be seen as an independent (fractional) multiset in which an element e appears with multiplicity $x(e)$. In particular, every matroid is an (integer) polymatroid.

Part I

Offline Scheduling

Chapter 3

Santa Claus and Makespan Minimization

3.1 Introduction

In this chapter, we study the relationship of two prominent optimization problems from resource allocation and scheduling theory: the SantaClaus problem and the Makespan problem. Regarding polynomial-time approximation algorithms, the best-possible approximation ratios are unknown for both problems. These issues are widely recognized as major open problems in the field [Ban17; SW99; WS11; Woe02]. We start by formally defining both problems and providing an overview of the state-of-the-art in terms of approximability.

The Santa Claus Problem. In the SantaClaus problem, we are given a set of m players P and a set of n indivisible resources R. Each resource $j \in R$ has unrelated values $v_{ij} \geq 0$ for each player $i \in P$. The task is to find an assignment of resources to players with the objective to maximize the minimum total value assigned to any player. Formally, we seek for a partition of resources $(R_i)_{i\in[m]}$ that maximizes $\min_{i\in P} \sum_{j\in R_i} v_{ij}$.

From the perspective of each individual player, this objective is arguably the best in terms of *fairness* compared to the other players. Therefore, this problem is also called the *max-min fair allocation problem* in the literature (see, for example, [BD05; CCK09; HS23]). The term "max-min" comes from the fact that the objective maximizes a minimum. The name "Santa Claus" is due to Bansal and Sviridenko [BS06], who stated this problem as Santa's task to distribute gifts to children in a way that makes the least happy child maximally happy.

In terms of polynomial-time approximation algorithms, it is entirely plausible that there exists a polynomial-time algorithm with constant approximation ratio for the problem, as it is only known that it is NP-hard to approximate the problem with a factor better than 2 [BD05]. On the other hand, the state-of-the-art is a polynomial-time n^{ε}-approximation for any constant $\varepsilon > 0$ and a quasi-polynomial-time $\text{poly}(\log n)$-approximation due to Chakrabarty et al. [CCK09]. Therefore, the big open question for the SantaClaus problem asks whether a constant-factor approximation in polynomial time is possible.

The Makespan Problem. In the Makespan problem, we are given a set of m machines M and a set of n jobs J. Every job $j \in J$ has size $p_{ij} \geq 0$ on machine $i \in M$. The task is to find an assignment of jobs to machines that minimizes the maximum load over all machines, which is called the makespan. Here, the load of a machine is the total size of jobs assigned to that machine. Formally, we seek for a partition of jobs $(J_i)_{i\in[m]}$ that minimizes $\max_{i\in M} \sum_{j\in J_i} p_{ij}$.

Note that this definition mimics the SantaClaus problem up to one detail in the objective function: instead of a max-min objective, we have a min-max objective for Makespan. Therefore, both problems are also considered to be "dual" to each other. While the above definition uses terminology from resource allocation, this problem is equal to the fundamental scheduling problem of finding a non-preemptive schedule on unrelated machines with the objective of minimizing the makespan, in 3-field notation $R \,||\, C_{\max}$.

Lenstra et al. [LST90] gave a beautiful 2-approximation algorithm based on rounding a sparse vertex solution of the so-called *assignment LP* (cf. Section 3.1.4). The rounding has been slightly improved to the factor $2 - \frac{1}{m}$ [SV05], but despite substantial research efforts, this upper bound remains undefeated. In terms of lower bounds, Lenstra et al. showed that it is NP-hard to approximate the problem with a factor better than $\frac{3}{2}$ [LST90]. Thus, the big open question is whether computing a better-than-2 approximation in polynomial time is possible for Makespan.

The Restricted Case and the Two-Value Case. Positive evidence towards affirmative answers for these notoriously difficult open questions comes from intensively studied special cases for both problems: the *restricted assignment* case and the *two-value* case. In the former, each resource / job can only be assigned to a specific subset of players / machines, but has the same value / size for all of those. In the latter, as the name suggests, there are only two non-trivial values $u < w$ for resources / jobs. The *trivial values* are $v_{ij} = 0$ in SantaClaus, which means that no solution has a benefit of assigning resource j to player i, and $p_{ij} = \infty$ in Makespan, which means that no reasonable solution can assign job j to machine i. We note here that if there only is one non-trivial value, the task becomes finding an assignment such that every player / machine receives a minimum / maximum number of resources / jobs, hence it reduces to the polynomial-time solvable b-matching problem.

In the restricted SantaClaus problem, the values satisfy $v_{ij} \in \{0, v_j\}$. The inapproximability of 2 from the general case still holds for restricted SantaClaus, even for restricted two-value SantaClaus [CTW18]. The currently best-known polynomial-time approximation algorithm has an approximation factor of at most $4 + \varepsilon$ for any $\varepsilon > 0$ [DRZ20].

Also the restricted Makespan problem, where $p_{ij} \in \{p_j, \infty\}$, has been studied extensively. While the inapproximability below $\frac{3}{2}$ also holds for this special case [LST90], the barrier of 2 has been overcome only partially: with non-constructive integrality gap bounds for the configuration LP [Sve12], better-than-2 approximations in quasi-polynomial time [JR20b], and for the restricted two-value case [Ann19; CKL15].

By comparing the formal definitions of SantaClaus and Makespan, one might immediately suspect that formal reductions between both problems, such as in terms of approximability, must be known. However, despite the community's belief [BR23; Ban17], nothing has been proven yet. We prove the first formal reductions between both problems regarding polynomial-time approximations. Before outlining our results, we highlight various aspects that connected both problems in the past.

Historical Relationships between both Problems. Various techniques have been previously transferred between both problems:

(i) **Configuration LP.** The configuration LP [BS06] (cf. Section 3.1.4) is the basis of many results for the restricted assignment variant of both problems.

(ii) **Haxell.** The local search technique by Haxell [Hax95] for hypergraph matching has been adopted and shown to be very powerful for both problems. First, it has been picked up for restricted SantaClaus [AKS17; AFS12; CM18b; CM19; DRZ20; HS23; PS16] and later it has been further developed for restricted Makespan [Ann19; JR19; JR20b; Sve12].

(iii) **Lovász Local Lemma.** Chakrabarty et al. [CKL15] transferred the technique of rounding the configuration LP via Lovász Local Lemma (LLL) used for restricted SantaClaus [Fei08; HSS11] to restricted Makespan with two job sizes and thereby provided the first better-than-2 approximation in polynomial time.

(iv) **Hardness Reductions.** The reduction for establishing NP-hardness of better-than-2 approximations for SantaClaus [BD05] is essentially the same as the earlier construction for the NP-hardness of better-than-$\frac{3}{2}$-approximations for Makespan [LST90].

(v) **Additive Guarantees.** The LP rounding by Lenstra et al. [LST90] achieves an additive approximation within the maximum (finite) processing time $p_{\max}$, which can be translated into a multiplicative 2-approximation. Bezáková and Dani [BD05] show the same additive approximation for SantaClaus: each player is guaranteed a value at least in a range of $v_{\max}$ within the optimal objective value. Note that for the max-min objective this does not translate into a multiplicative guarantee.

3.1.1 Our Results

We confirm the existence of a formal relationship between the Makespan and the SantaClaus problem with respect to their approximability. Our first result is a one-sided relation between the two major open questions: any better-than-2 approximation algorithm for Makespan implies a constant approximation algorithm for SantaClaus. More specifically, we prove that for every $\alpha \geq 2$, if there exists a polynomial-time $(2 - \frac{1}{\alpha})$-approximation algorithm for Makespan, then there exists a polynomial-time $(\alpha + \varepsilon)$-approximation algorithm for SantaClaus for any $\varepsilon > 0$ (Section 3.2). For values of $\alpha < 2$, computing a $(2 - \frac{1}{\alpha})$-approximation for the Makespan problem is NP-hard and the implication would still hold, even though clearly uninteresting.

For the two-value cases, we can also prove the reverse relation. Furthermore, we improve upon the factor of $1 + \varepsilon$ in the reduced approximation factor. This implies the following equivalence of Makespan and SantaClaus in terms of approximability for the two-value cases: for any $\alpha \geq 2$, there exists an α-approximation algorithm for two-value SantaClaus if and only if there exists a $(2 - \frac{1}{\alpha})$-approximation algorithm for two-value Makespan (Section 3.3).

As mentioned above, there are many improved results known for the restricted cases of SantaClaus and Makespan. One might suspect that reductions within this special case could potentially lead to further improvements of approximation factors. For example, *if* the existence of an α-approximation for restricted SantaClaus *would* imply an $(2 - \frac{1}{\alpha})$-approximation for restricted Makespan, then we could derive a polynomial-time $(1.75+\varepsilon)$-approximation algorithm for restricted Makespan for any $\varepsilon > 0$ via the result by Davies et al. [DRZ20], which would be the first polynomial-time better-than-2 approximation algorithm for restricted Makespan.

Using our current techniques, however, this seems unclear, since the aforementioned reductions do not maintain the characteristics of the restricted case, that is, the reduction of a restricted Makespan instance may not be a restricted SantaClaus instance.

Fortunately, it turns out that we prove similar reductions *within* a slight generalization of the restricted SantaClaus and restricted Makespan problems. The high-level idea for these generalizations is that we allow resources / jobs to be copied to multiple machines, also potentially multiple times. The set of possible configurations in which a resource / job can be copied is defined via a polymatroid. More specifically, every resource / job is associated with an integer polymatroid over the set of players / machines, and the bases of this polymatroid define the set of possible configurations. We call these generalizations the matroid SantaClaus and matroid Makespan problem. The formal definitions are given in Section 3.4.

For the restricted two-value cases of these generalizations, we can again prove equivalence in terms of approximability. We show that for any $\alpha \geq 2$, there exists a polynomial-time α-approximation algorithm for the restricted two-value matroid SantaClaus problem if and only if there exists a polynomial-time $(2 - \frac{1}{\alpha})$-approximation algorithm for the restricted two-value matroid Makespan problem (Section 3.5).

Finally, we show in Section 3.4.2 that the classic additive rounding theorems of the assignment LP by Bezáková and Dani [BD05] and Lenstra et al. [LST90] can be lifted to these generalizations.

3.1.2 Implications and Corollaries

By applying our reductions to existing results, we have two immediate implications to the state-of-the-art of the Makespan problem.

Corollary 3.1. *For every $\varepsilon > 0$, there exists a polynomial-time $(2 - 1/n^{\varepsilon})$-approximation algorithm and a quasi-polynomial-time $(2 - 1/\mathrm{poly}(\log n))$-approximation algorithm for two-value Makespan.*

This result follows from the algorithm of Chakrabarty et al. [CCK09] and our reduction for the general two-value problems (cf. Theorem 3.12). This approximation guarantee partially improves upon the best-known polynomial-time approximation factor of $2 - \frac{1}{m}$ for the two-value Makespan problem [SV05].

Corollary 3.2. *For every $\varepsilon > 0$, there exists a polynomial-time $(1.75 + \varepsilon)$-approximation algorithm for the restricted two-value matroid Makespan problem.*

As the matroid Makespan problem is a generalization, this statement holds in particular true for the restricted Makespan problem; thus, improving upon the previously best polynomial-time approximation factor of $1 + \frac{2}{\sqrt{5}} + \varepsilon \approx 1.8945$ by Annamalai [Ann19]. The corollary follows from combining the 4-approximation algorithm of Bamas et al. [Bam+24] for the restricted two-value matroid SantaClaus problem and our reduction for the restricted two-value matroid problems (cf. Theorem 3.22).

3.1.3 Binary Search Framework

A standard tool for proving approximation guarantees for SantaClaus and Makespan is a so-called *guessing framework* or *binary search framework* (see, for example, [HS87; LST90]), which we now introduce for SantaClaus. There is a straightforward analogue for Makespan.

To this end, consider a SantaClaus instance I with optimal objective value $\mathrm{OPT}(I)$ and let $\alpha \geq 1$. We can without loss of generality assume that all values are integers and that $\mathrm{OPT}(I) \in [0, T_{\max}]$ with $T_{\max} = \sum_{i \in P, j \in R} v_{ij}$. Given any integer T, the *α-decision variant* asks whether there exists a solution with objective value at least T, and if so, it asks for a solution with objective value at least T/α. In other words, an algorithm that solves the α-decision variant must compute, if there exists at solution with value at least T, a solution with objective value at least T/α.

We first argue that if there exists a polynomial-time algorithm for the α-decision variant, then there exists a polynomial-time α-approximation algorithm for SantaClaus. This is because we can define a binary search that finds $\mathrm{OPT}(I)$ as follows. If the algorithm, given the current guess T, outputs no solution of value at least T/α, we can conclude that $\mathrm{OPT}(T) < T$, and repeat the process using a smaller guess. Otherwise, we can repeat with a larger guess. Since all values are integers, this process terminates after $O(\log T_{\max})$ iterations, which is polynomial in the size of the instance. After $\mathrm{OPT}(I)$ is determined, the algorithm for the α-decision variant outputs a solution with objective value at least $\mathrm{OPT}(I)/\alpha$.

Moreover, since the objective of SantaClaus is linear in the values v, we can scale all values by T and obtain the following tool.

Proposition 3.3. *If there exists a polynomial-time algorithm that, given a SantaClaus instance I with $\mathrm{OPT}(I) \geq 1$, computes a solution for I with objective value at least $\frac{1}{\alpha}$, then there exists a polynomial-time α-approximation algorithm for SantaClaus.*

3.1.4 Linear Programming Relaxations

There are two relevant linear programming relaxations for both SantaClaus and Makespan. We state both for the decision variants of the problems. That is, given a value T, the decision problem asks whether there exists a solution with objective value at least T for SantaClaus or at most T for Makespan. Therefore, the LP relaxations have no objective functions and only try to find a feasible solution. We state both relaxations for Makespan, but the adaption for SantaClaus is straightforward.

The assignment LP uses for every machine i and job j a (relaxed) binary variable x_{ij} to indicate whether job j is assigned to machine i. The set of feasible solutions $x = (x_{ij})_{i \in M, j \in J}$ must satisfy the following constraints:

$$\sum_{j \in J} p_{ij} \cdot x_{ij} \leq T \quad \forall i \in M$$

$$\sum_{i \in M} x_{ij} = 1 \quad \forall j \in J$$

$$x_{ij} \geq 0 \quad \forall i \in M, j \in J$$

The first set of constraints guarantees that every machine receives a load of at most T, and the second set of constraints ensures that every job is assigned to exactly one machine. For the Makespan problem, one can additionally strengthen this relaxation by forbidding even fractional assignments of jobs j to machines i if their size on i is larger than T:

$$x_{ij} = 0 \quad \forall i \in M, j \in J \text{ where } p_{ij} > T \,.$$

This was used by Lenstra et al. when they introduced this relaxation for their aforementioned 2-approximation [LST90]. For SantaClaus, this relaxation was first used by Bezáková and Dani [BD05].

For the configuration LP, we first introduce for every machine i the set of feasible *configurations* $\mathcal{C}_i$, that is, sets of jobs with a total size of at most T on machine i. Formally, $\mathcal{C}_i = \{S \subseteq J \mid \sum_{j \in S} p_{ij} \leq T\}$. The configuration LP then uses for every machine i and configuration $C \in \mathcal{C}_i$ a (relaxed) binary variable x_{iC} to indicate whether jobs of configuration C should be assigned to machine i. The set of feasible solutions $x = (x_{iC})_{i \in M, C \in \mathcal{C}_i}$ must satisfy the following constraints:

$$\begin{aligned} \sum_{C \in \mathcal{C}_i} x_{iC} &= 1 \quad \forall i \in M \\ \sum_{i \in M} \sum_{C \in \mathcal{C}_i : j \in C} x_{iC} &= 1 \quad \forall j \in J \\ x_{iC} &\geq 0 \quad \forall i \in M, C \in \mathcal{C}_i \end{aligned}$$

The first set of constraints ensures that exactly one configuration is selected for every machine, and the second set of constraints guarantees that every job appears in exactly one selected configuration. This relaxation was introduced for the SantaClaus problem by Bansal and Sviridenko [BS06], who also proved that it is stronger than the assignment LP. Furthermore, they showed that, despite its exponential number of variables, if there exists a feasible solution of value T, then a solution of value at most $(1 + \varepsilon)T$ can be computed in polynomial time for any $\varepsilon > 0$.

3.1.5 Further Related Work

Makespan. The first polynomial-time 2-approximation algorithm for Makespan is due to Lenstra et al. [LST90]. Their rounding of the assignment LP has later been slightly improved by Shchepin and Vakhania [SV05] to an approximation ratio of at most $2 - \frac{1}{m}$. This matches the integrality gap of the assignment LP, even in the restricted case. It is worth noting that there are other 2-approximation algorithms known, for example, by a different rounding technique for the assignment LP [ST93], or a simpler, combinatorial approach [GMW07].

For the restricted Makespan problem, Svensson [Sve12] proved in his seminal work that the configuration LP has an integrality gap of at most $\frac{33}{17}$, and thus, is stronger than the assignment LP. Jansen and Rohwedder [JR20b] improved this bound to $\frac{11}{6}$ and further presented an $(\frac{11}{6} + \varepsilon)$-approximation algorithm that terminates with a feasible solution in quasi-polynomial time for any $\varepsilon > 0$. In the two-value case of the restricted Makespan problem, it is known that the configuration LP has an integrality gap of at most $\frac{5}{3}$ [JLM18]. For this problem, Chakrabarty

et al. [CKL15] presented the first polynomial time better-than-2 approximation algorithm. Annamalai [Ann19] improved this factor further to $1 + \frac{2}{\sqrt{5}} + \varepsilon \approx 1.8945$.

The integrality gap of the configuration LP for the general Makespan problem is equal to 2 (for an arbitrary number of machines). The upper bound follows from the positive results by Lenstra et al. and the fact that the configuration LP is stronger than the assignment LP. The lower bound of 2 already holds in the special case where every job has a non-trivial size on at most two machines [VW14]. This special case is also called *graph balancing*, as one can imagine that the machines are vertices and the jobs are edges of a graph, and orienting an edge to a vertex corresponds to assigning the job to the machine. For the restricted graph balancing problem, Ebenlendr et al. [EKS14] developed an $\frac{7}{4}$-approximation algorithm, which rounds a solution of a strengthened version of assignment LP. Furthermore, the assignment LP has an integrality gap equal to $\frac{7}{4}$, and, similar to the restricted Makespan problem, the configuration LP for restricted graph balancing has a strictly smaller integrality gap of at most 1.749 [JR19]. The hardness of approximation for factors better than $\frac{3}{2}$ even holds for the restricted graph balancing problem [EKS14].

If the machines are identical, that is, $p_j = p_{ij}$ for all jobs j, then, for any $\varepsilon > 0$, there exists an $(1+\varepsilon)$-approximation algorithm [HS87]. This special case is already NP-hard to solve optimally for two machines via a straightforward reduction from the Partition problem.

If the number of machines m is fixed and not part of the input, there exists, for any $\varepsilon > 0$, a $(1+\varepsilon)$-approximation algorithm for the general Makespan [HS76].

Santa Claus. For the general SantaClaus problem, the state-of-the-art polynomial-time approximation algorithm by Chakrabarty et al. has an approximation factor of at most n^{ε} for any $\varepsilon > 0$ [CCK09]. Their algorithm can also be configured to a $\text{poly}(\log n)$-approximation algorithm that runs in quasi-polynomial-time $n^{O(\log n/\log\log n)}$. The same result was previously achieved by Bateni et al. [BCG09] for the max-min degree arborescence problem, a special case of SantaClaus where all values satisfy $v_{ij} \in \{0, 1, \infty\}$, for every resource j there is at most one player i with $v_{ij} = \infty$, and for every player i there is at most one resource j with $v_{ij} = \infty$. Recently, Bamas and Rohwedder gave an improved quasi-polynomial-time approximation factor of $\text{poly}(\log\log n)$ for this problem [BR23].

In the restricted setting, Bansal and Sviridenko [BS06] showed an $\frac{\log\log n}{\log\log\log n}$-approximation algorithm by rounding the configuration LP randomly and applying Lovász Local Lemma (LLL). Feige [Fei08] then showed by a non-constructive argument that the integrality gap of the configuration LP is constant, which later was turned into the first constant-factor approximation algorithm for restricted SantaClaus by making LLL constructive [HSS11]. A second line of research for this problem was initiated by Asadpour et al. [AFS12], who established a connection between analyzing the restricted SantaClaus problem against the configuration LP and a classical result by Haxell [Hax95] for hypergraph matching. Several results [AKS17; CM18b; CM19; PS16] in this direction converged to the current state-of-the-art polynomial-time approximation algorithm with an approximation factor of $4 + \varepsilon$ for any $\varepsilon > 0$ [DRZ20]. The upper bound on the integrality gap of the configuration LP has been further improved to $\frac{53}{15} \approx 3.534$ [AFS12; CM18a; CM19; HS23; JR20a].

If there are only two non-trivial values in restricted SantaClaus problem, Chan, Tang and Wu [CTW18] showed that the integrality gap of the configuration LP is at most $3 + \varepsilon$ for any

$\varepsilon > 0$, and further gave a quasi-polynomial-time algorithm with an approximation factor of at most $3 + 4u$, where the two values are 1 and $u \in (0, 1)$.

For the special case of the SantaClaus problem where every resource has non-trivial values for two players (the graph balancing special case of SantaClaus), the integrality gap of the configuration LP is equal to 2 [VW14], and there is a polynomial-time approximation algorithm matching this factor [CCK09; VW14].

Matroid Variants. Davies et al. [DRZ20] introduced along the way to their approximation algorithm a different matroid generalization of (restricted) SantaClaus. One can see this problem as being halfway between the traditional restricted problem and our matroid generalization of restricted SantaClaus. Indeed, in their problem there is one resource of large value, that can be copied to multiple players subject to a matroid constraint, and many traditional resources of small value that each can only be assigned to one player. In particular, our matroid SantaClaus problem generalizes their matroid variant.

Further, a special case of the matroid Makespan problem has already been considered in the past. Azar et al. [ACL18] give a 2-approximation for the Makespan problem where each job j needs to be processed by at least k_j different machines (possibly in parallel), which is the special case of our matroid Makespan problem where each job j is equipped with a uniform matroid of rank k_j. Interestingly, Azar et al. [ACL18] mention that the rounding theorem of Lenstra et al. [LST90] cannot be applied in the matroid setting, because it crucially relies on a counting argument that does not hold anymore. In our 2-approximation rounding theorem of the assignment LP for the matroid Makespan problem, we use the rounding technique of Shmoys and Tardos [ST93], which does not need this argument.

3.2 Reduction of Santa Claus to Makespan

We prove our first main result, the reduction of the problem of finding a constant-factor approximation algorithm for SantaClaus to the problem of finding a better-than-2 approximation algorithm for Makespan.

Theorem 3.4. *For any $\alpha \geq 2$ and $\varepsilon > 0$, if there exists a polynomial-time $(2 - \frac{1}{\alpha})$-approximation algorithm for Makespan, then there exists a polynomial-time $(\alpha + \varepsilon)$-approximation algorithm for SantaClaus.*

Using the binary search framework and Proposition 3.3, this theorem is implied by the following lemma, which we prove in this section.

Lemma 3.5. *For any $\alpha \geq 2$ and $\varepsilon > 0$, given an instance I_S of SantaClaus with $\mathrm{OPT}(I_S) \geq 1$, we can construct in polynomial time an instance I_M of Makespan such that, given a $(2 - \frac{1}{\alpha})$-approximate solution for I_M, we can compute in polynomial time a solution for I_S with an objective value of at least $\frac{1}{\alpha+\varepsilon}$.*

The proof of this lemma is split into two parts: Lemma 3.6 and Lemma 3.9. Before moving to those, we give a high-level idea for the reduction.

In the Makespan instance I_M, we introduce for every player a gadget (several machines and jobs) that encodes different configurations (sets) of resources, each giving the player a total

value of at least 1. The gadget structure forces any solution for I_M to *select* one configuration for each player. Further, a solution can *use* a resource of the selected configuration for a player, which we then can translate back to a solution for I_S by giving the used resource to that player. To ensure that this interpretation is well-defined, we connect these gadgets in a way that prevents any solution for I_M for using a distinct resource in selected configurations of two or more different players.

The construction roughly satisfies the following properties. An optimal solution for I_M has to select for every player a configuration in a way that they can use all of their resources. This can be translated to a solution for I_S with objective value at least 1. Note that this essentially mimics the idea of the configuration LP for SantaClaus. On the other side, a 2-approximation for I_M can select configurations without using any resources, hence we cannot guarantee any approximation factor for I_S. A $(2 - \frac{1}{\alpha})$-approximation for I_M, however, must use for every selected configuration resources of total value at least $\frac{1}{\alpha}$, which translates to a solution for I_S with objective value at least $\frac{1}{\alpha}$.

In fact, this high-level description matches the full proof if we allow exponential time. This is because there can be exponentially many configurations of resources that give a player a value of at least 1. We overcome this issue in the next section, where we introduce an intermediate rounded instance and for every player a *polynomial* number of relevant configurations. In Section 3.2.2, we then formalize the above intuition and prove Lemma 3.5.

3.2.1 Reduction to a Polynomial Number of Configurations

Consider a SantaClaus instance I_S with players P and n resources R. We define the set of *value types* as $\mathcal{T} = \{v_{ij} : i \in P, j \in R\}$ that contains all distinct resource values that occur in the instance. We call a function $c : \mathcal{T} \to \{0, 1, \dots, n\}$ a *configuration*, and define the *total value* of c as $|c| = \sum_{v \in \mathcal{T}} c(v) \cdot v$. One can also see a configuration as a multiset of value types.

Given a configuration c_i for a player i of a SantaClaus instance I_S, we say that a resource assignment $(R_i)_{i \in P}$ for I_S that assigns the set of resources R_i to player i *matches* the configuration c_i if $|\{j \in R_i : v_{ij} = v\}| = c_i(v)$ for every value type $v \in \mathcal{T}$.

We use $\mathcal{C}_i$ to refer to a set of configurations for a player $i \in P$ and call $\mathcal{C} = (\mathcal{C}_i)_{i \in P}$ a *collection of configurations*. A resource assignment $(R_i)_{i \in P}$ matches a collection of configurations $\mathcal{C}$ if, for each player i, there exists a configuration $c \in \mathcal{C}_i$ such that R_i matches c. Given a SantaClaus instance I_S and a collection of configurations $\mathcal{C} = (\mathcal{C}_i)_{i \in P}$, we use $\mathrm{OPT}_{\mathcal{C}}(I_S)$ to refer to the optimal objective value for instance I_S among those solutions that match $\mathcal{C}$.

The main result of this section is the following lemma.

Lemma 3.6. *For every $\varepsilon > 0$ and a given instance I_S of SantaClaus with $\mathrm{OPT}(I_S) \geq 1$, we can construct a rounded instance I'_S and a collection of configurations $\mathcal{C}$ such that the number of configurations for each player is polynomial in the input size of I_S and $\mathrm{OPT}_{\mathcal{C}}(I'_S) \geq \frac{1}{1+\varepsilon}$. Further, every solution for I'_S of objective value T is a solution for I_S with objective value at least T.*

The lemma essentially allows us to consider only solutions that *partially match* the constructed collection of configurations $\mathcal{C}$. That are solutions $(R_i)_{i \in P}$ in which for every player i there is some $c_i \in \mathcal{C}_i$ such that $|\{j \in R_i : v_{ij} = v\}| \leq c_i(v)$ for every value type $v \in \mathcal{T}$. If we find

such a solution that α-approximates $\mathrm{OPT}_{\mathcal{C}}(I'_S)$, we immediately get a $(\alpha+\varepsilon)$-approximation for $\mathrm{OPT}(I_S)$. The remaining section is dedicated to the proof of Lemma 3.6.

Let I_S be a SantaClaus instance with the set P of m players, the set R of n resources and $\mathrm{OPT}(I_S) \geq 1$. We first describe the construction of I'_S and $\mathcal{C}$. Let $\varepsilon > 0$ be a sufficiently small constant and $\kappa = \lceil 1/\varepsilon^3 \rceil$. We construct the SantaClaus instance I'_S by executing the following steps:

1. Use the same set of players and resources as in I_S.
2. Round each resource values v_{ij} down to the closest power of $\frac{1}{1+\varepsilon}$. If $v_{ij} \geq 1$, then we set $\bar{v}_{ij} = 1$. Furthermore, we round all v_{ij} with $v_{ij} < \frac{1}{(1+\varepsilon)n}$ to 0. In summary, each $\bar{v} \in \mathcal{T}$ is either a power of $\frac{1}{1+\varepsilon}$ of value at least $\frac{1}{(1+\varepsilon)n}$ or 0.

Next, we construct a collection of configurations $\mathcal{C}$ for the rounded instance I'_S by executing the following steps. Since these steps reduce the number of possible configurations per player to a polynomial, an algorithm creating the configurations can just compute them via enumeration.

3. For each player $i \in P$, we have a set $\mathcal{C}_i$ of configurations c such that, for every value type $\bar{v} \in \mathcal{T}$, either $c(\bar{v}) = 0$, $c(\bar{v}) = \lceil (1+\varepsilon)^\ell \rceil$ or $c(\bar{v}) = \lfloor (1+\varepsilon)^\ell \rfloor$ for some $\ell \in \mathbb{N}_0$ with $(1+\varepsilon)^\ell \leq n$.
4. Let $\bar{v}_1 \geq \ldots \geq \bar{v}_\tau$ be the rounded value types in $\mathcal{T}$. We partition $\mathcal{T}$ into κ *value classes* $\mathcal{T}_1, \ldots, \mathcal{T}_\kappa$ where $\mathcal{T}_\ell := \{\bar{v}_{\ell+s\cdot\kappa} : s = 0, 1, \ldots\}$. For every player i, we further restrict the set of configurations $\mathcal{C}_i$ to configurations c that satisfy for every $1 \leq \ell \leq \kappa$ and for every $\bar{v}, \bar{v}' \in \mathcal{T}_\ell$ with $\bar{v} > \bar{v}'$ that either $c(\bar{v}) < c(\bar{v}')$ or $c(\bar{v}) = 0$ or $c(\bar{v}') = 0$. That is, the number of occurrences of a value type $\bar{v} \in \mathcal{T}_\ell$ of one value class that actually occur in a configuration (that is, $c(\bar{v}) > 0$) increase with decreasing value $\bar{v}$.

Having I'_S and $\mathcal{C}$ constructed, we prove below two auxiliary lemmas that immediately imply the first part of Lemma 3.6. For the second part, note that the values of I'_S are only rounded down. Thus, every solution for I'_S of objective value T is a solution for I_S with objective value at least T.

Lemma 3.7. *For every player i, the size of $\mathcal{C}_i$ is polynomial in the size of I_S.*

Proof. Because of the Step 2, the number of value types in I'_S is in $O(\log_{1+\varepsilon} n) \subseteq O(\frac{1}{\varepsilon}\log n)$. By Step 3, the number of distinct function values $c(\bar{v})$ over all configurations $c \in \mathcal{C}_i$ and all value types $\bar{v} \in \mathcal{T}$ is in $O(\log_{1+\varepsilon} n) \subseteq O(\frac{1}{\varepsilon}\log n)$ as well. By Step 4, we can for every $1 \leq \ell \leq \kappa$ represent the entries of $c \in \mathcal{C}_i$ that correspond to the same value class $\mathcal{T}_\ell$ in terms of a vector with $O(\frac{1}{\varepsilon}\log n)$ entries such that all non-zero entries strictly increase in value. Here, each entry of the vector corresponds to a value type $\bar{v} \in \mathcal{T}_\ell$, in decreasing order, and the entry values represent the corresponding function values $c(\bar{v})$. We can represent such a vector by the set of indices whose corresponding entries have a non-zero value and by the set of non-zero values that occur in the vector, because, given these two sets, we can reconstruct the corresponding unique vector. Since there are at most $2^{O(\log(n)/\varepsilon)}$ different sets of non-zero values that can occur in the vector and at most $2^{O(\log(n)/\varepsilon)}$ different sets of non-zero entries, the number of such vectors is $2^{O(\log(n)/\varepsilon)} \cdot 2^{O(\log(n)/\varepsilon)} \subseteq n^{O(1/\varepsilon)}$.

Finally, we can compose the vectors of every value class to a vector of size $n^{O(\kappa/\varepsilon)} = n^{O(1/\varepsilon^4)}$ for all value types $\mathcal{T}$ using the unique partition of value classes. Since ε is constant, the total number of vectors representing $\mathcal{C}_i$ is polynomial in the size of I_S. □

Lemma 3.8. *It holds that* $\mathrm{OPT}_{\mathcal{C}}(I'_S) \geq \frac{1}{1+\varepsilon}$.

Proof. Let $\mathcal{C}'$ denote the collection of configurations that is created by only executing Step 3 of the construction and let $\mathcal{C}$ denote the final collection of configurations. We separately prove $\mathrm{OPT}_{\mathcal{C}'}(I'_S) \geq 1/(1+\varepsilon)^3$ and $\mathrm{OPT}_{\mathcal{C}}(I'_S) \geq \frac{1}{1+\varepsilon} \cdot \mathrm{OPT}_{\mathcal{C}'}(I'_S)$. Then, for any sufficiently small $\varepsilon' > 0$, we can choose $\varepsilon = \varepsilon'/5$ and conclude $\mathrm{OPT}_{\mathcal{C}}(I'_S) \geq \frac{1}{1+\varepsilon'}$.

We first show $\mathrm{OPT}_{\mathcal{C}'}(I'_S) \geq 1/(1+\varepsilon)^3$. Consider an optimal solution for I_S. For a player i, let R_i denote the resources that are assigned to i in this optimal solution. Clearly $v(R_i) \geq 1$. Discarding all resources in R_i with value smaller than $\frac{1}{(1+\varepsilon)n}$ reduces the value of R_i by a factor of at most $1+\varepsilon$. Rounding the remaining resource values down to powers of $1+\varepsilon$ reduces the value by another factor of $1+\varepsilon$. To make sure that the remaining resources in R_i with their rounded values match a configuration in $\mathcal{C}'_i$, we might have to remove a $1+\varepsilon$ fraction of the resources for each value type from R_i. This reduces the value of R_i by another factor of $1+\varepsilon$. The remaining value is at least $1/(1+\varepsilon)^3$. By doing this for every player i, we obtain a solution for I'_S that matches $\mathcal{C}'$ (recall that $\mathcal{C}'$ does not enforce the monotonicity of Step 4) with an objective value of at least $1/(1+\varepsilon)^3$, which implies $\mathrm{OPT}_{\mathcal{C}'}(I'_S) \geq 1/(1+\varepsilon)^3$.

Finally, we prove $\mathrm{OPT}_{\mathcal{C}}(I'_S) \geq \frac{1}{1+\varepsilon} \cdot \mathrm{OPT}_{\mathcal{C}'}(I'_S)$. To that end, fix an optimal solution for I'_S among those solutions that match $\mathcal{C}'$. Fix any player i and let $c'_i \in \mathcal{C}'_i$ denote the configuration that is selected for player i in the optimal solution for I'_S. We argue that we can find a configuration $c_i \in \mathcal{C}_i$ that

(i) has a total value that is at least a $\frac{1}{1+\varepsilon}$ fraction of the value of c'_i, and

(ii) satisfies $c_i(\overline{v}) \leq c'_i(\overline{v})$ for all $\overline{v} \in \mathcal{T}$.

This gives us a feasible solution for I'_S that matches $\mathcal{C}$ and has an objective value of at least $\frac{1}{1+\varepsilon} \cdot \mathrm{OPT}_{\mathcal{C}'}(I'_S)$, and thus proves the statement.

We start by building a configuration c_i independently for every value class $\mathcal{T}_\ell$, for $1 \leq \ell \leq \kappa$. First, we iteratively construct a subset $S_\ell \subseteq \mathcal{T}_\ell$ of value types as follows: Start with the largest $\overline{v} \in \mathcal{T}_\ell$ such that $c'_i(\overline{v}) > 0$ and add $\overline{v}$ to S_ℓ. Then, find the largest $\overline{v}' \in \mathcal{T}_\ell$ with $\overline{v} > \overline{v}'$ and $c'_i(\overline{v}) < c'_i(\overline{v}')$. Add $\overline{v}'$ to S_ℓ and repeat from $\overline{v}'$ until we do not find another value type to add. Based on S_ℓ, define configuration c_i as $c_i(\overline{v}) = c'_i(\overline{v})$ if $\overline{v} \in S_\ell$ and $c_i(\overline{v}) = 0$ otherwise.

Note that by the choice of the sets S_ℓ, the configuration c_i is contained in $\mathcal{C}_i$ and satisfies (ii).

It remains to prove that c_i also satisfies (i). Fix a value class ℓ and an arbitrary value type $\overline{v}_j \in S_\ell$, and let $\overline{v}_{j'}$ denote the next smaller value type in S_ℓ. Recall that the rounded value types are indexed in decreasing order, and let s be the integer such that $j' = j + \kappa \cdot (s+1)$. If $\overline{v}_j$ is already the smallest value type in S_ℓ, we set s to the largest integer such that $\ell + \kappa \cdot s \leq \tau$. (Recall that τ is the number of value types in I'_S.)

We show that

$$c_i(\overline{v}_j) \cdot \overline{v}_j \geq \frac{1}{1+\varepsilon} \cdot \sum_{s'=0}^{s} c'_i(\overline{v}_{j+s'\cdot\kappa}) \cdot \overline{v}_{j+s'\cdot\kappa} \,. \tag{3.1}$$

If this inequality holds for all $\overline{v}_j \in S_\ell$, and for all $1 \leq \ell \leq \tau$, then (i) follows.

To prove the inequality, observe that, by the choice of S_ℓ, all integers $0 \leq s' \leq s$ satisfy $c'_i(\bar{v}_{j+s'\cdot\kappa}) \leq c'_i(\bar{v}_j)$. Furthermore, $\bar{v}_{j+s'\cdot\kappa} = \bar{v}_j/(1+\varepsilon)^{\kappa\cdot s'}$ by the rounding of the value types. This gives us

$$\begin{aligned}\sum_{s'=0}^{s} c'_i(\bar{v}_{j+s'\cdot\kappa}) \cdot \bar{v}_{j+s'\cdot\kappa} &\leq c'_i(\bar{v}_j) \cdot \sum_{s'=0}^{s} \bar{v}_{j+s'\cdot\kappa} \\ &= c_i(\bar{v}_j) \cdot \bar{v}_j \cdot \sum_{s'=0}^{s} \frac{1}{(1+\varepsilon)^{\kappa\cdot s'}} \leq c_i(\bar{v}_j) \cdot \bar{v}_j \cdot \frac{1}{1-\frac{1}{(1+\varepsilon)^\kappa}} \qquad (3.2)\end{aligned}$$

using the geometric series. Since

$$\kappa \geq \frac{1+\varepsilon}{\varepsilon^2} \geq \frac{1}{\varepsilon} \cdot \frac{1}{\ln(1+\varepsilon)} \geq \frac{\ln(1+\frac{1}{\varepsilon})}{\ln(1+\varepsilon)} = \log_{1+\varepsilon}\left(\frac{1+\varepsilon}{\varepsilon}\right),$$

we conclude that (3.2) is at most $c_i(\bar{v}_j) \cdot \bar{v}_j \cdot (1+\varepsilon)$, which implies (3.1). This concludes the proof of $\mathrm{OPT}_{\mathcal{C}}(I'_S) \geq \mathrm{OPT}_{\mathcal{C}'}(I'_S)/(1+\varepsilon)$, and the proof of the lemma. □

3.2.2 Construction of a Makespan Instance

Using Lemma 3.6, we can assume that we are given both a SantaClaus instance I_S and a collection of configurations $\mathcal{C}$. We now prove the following lemma, which, together with Lemma 3.6, implies Lemma 3.5.

Lemma 3.9. *Let I_S be a SantaClaus instance and let $\mathcal{C}$ be a collection of configurations with $\mathrm{OPT}_{\mathcal{C}}(I_S) \geq 1$. For any $\alpha \geq 1$, we can construct in polynomial time a Makespan instance I_M such that, given a $(2-\frac{1}{\alpha})$-approximate solution for I_M, we can compute in polynomial time a solution for I_S with value at least $\frac{1}{\alpha}$. The running times are polynomial in the size of $(I_S, \mathcal{C})$.*

We now prove this lemma. Fix an instance I of the SantaClaus problem and a collection of configurations $\mathcal{C}$ for I with $\mathrm{OPT}_{\mathcal{C}}(I) \geq 1$. We proceed by constructing the Makespan instance I' as follows:

1. Remove all configurations from $\mathcal{C}$ of value strictly less than 1.
2. For every player i introduce a *player-job* j_i in I'.
3. For every resource j introduce a *resource-machine* m_j in I'.
4. For every player i and every configuration $c \in \mathcal{C}_i$ introduce a *configuration-machine* m^i_c in I', where the size of the player-job j_i is equal to 1 on every configuration-machine m^i_c and ∞ on all other machines.
5. For every player i, every configuration $c \in \mathcal{C}_i$ and every value type $v \in \mathcal{T}$ introduce a set $J^i_{c,v}$ of $c(v)$ many *configuration-jobs*. A configuration-job in $J^i_{c,v}$ has size 1 on every resource-machine m_j if resource j has value type v for player i (that is, $v_{ij} = v$), size $\frac{v}{|c|}$ on the configuration-machine m^i_c, and ∞ on all other machines.

Since we assumed that $\mathrm{OPT}_{\mathcal{C}}(I_S) \geq 1$, the first step does not affect $\mathrm{OPT}_{\mathcal{C}}(I_S)$. See Figure 3.1 for an example of this construction. In the following, we prove two auxiliary lemmas that imply Lemma 3.9.

Lemma 3.10. *The optimal objective value of I_M is at most* 1.

Proof. Fix a solution of I_S that is optimal among the solutions that match $\mathcal{C}$. Consider a player i of instance I_S and let $c_i \in \mathcal{C}_i$ be the matched configuration for player i in the fixed solution. Let R_i be the set of resources assigned to player i.

We construct a solution for I_M as follows. We assign job j_i to machine $m^i_{c_i}$, giving it a load equal to 1. Further, we assign the configuration-jobs $J^i_{c_i,v}$ of configuration c_i to resource-machines $\{m_j : j \in R_i\}$ such that every resource-machine receives at most one job. Such an assignment must exist, because configuration c_i is matched by the fixed solution for I_S. For every configuration $c' \in \mathcal{C}_i \setminus \{c_i\}$ and for every $v \in \mathcal{T}$, we assign every configuration-job in $J^i_{c',v}$ to machine $m^i_{c'}$, giving those machines a load equal to $\sum_{v\in\mathcal{T}} c'(v)\frac{v}{|c'|} = 1$. Since in the given solution every resource j is assigned to at most one player i, and since we have assigned at most one configuration-job to machine m_j, every resource-machine also has a load of at most 1. Hence, the makespan of the constructed solution for I_M is at most 1. □

Using Lemma 3.10, we can conclude that a $(2 - \frac{1}{\alpha})$-approximation algorithm computes a solution with a makespan of at most $2 - \frac{1}{\alpha}$ for instance I_M. Thus, the following lemma implies Lemma 3.9.

Lemma 3.11. *For any $\alpha \geq 1$, given a solution for I_M with a makespan of at most $2 - \frac{1}{\alpha}$, we can construct in polynomial time a solution for I_S where every player receives a total value of at least $\frac{1}{\alpha}$.*

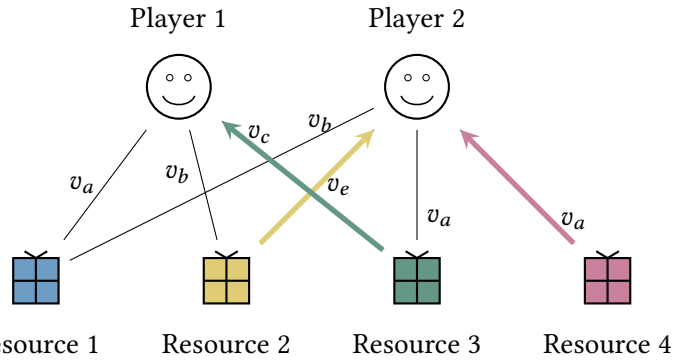
Proof. Suppose that we are given such a solution for I_M. We first construct a solution for I_S. Fix a player i and assume that j_i is assigned to machine m^i_c for some configuration $c \in \mathcal{C}_i$. Let N_i be the set of configuration-jobs (of potentially different values) of configuration c of player i that are *not* assigned to m^i_c. Thus, every job in N_i is assigned to one resource-machine. Also, every resource-machine has at most one assigned job because every job has a size of at least 1 on these machines, and the given solution has a makespan of less than 2. Let R_i be the set of resources for which the corresponding resource-machines receive a job of N_i. For the solution of I_S, we assign resources R_i to player i. After doing this for every player, we distribute unassigned resources arbitrarily.

We now argue that the total value of resources in R_i for player i is at least $\frac{1}{\alpha}$, which concludes the proof. Let $J^i_c = \cup_{v\in\mathcal{T}} J^i_{c,v}$ be the set of all configuration-jobs of configuration c of player i, and note that their total size on machine m^i_c is equal to $\sum_{v\in\mathcal{T}} c(v)\frac{v}{|c|} = 1$. Thus, $J^i_c \setminus N_i$ are those configuration-jobs that are assigned to m^i_c, and since the makespan of the given solution is at most $2 - \frac{1}{\alpha}$, their load contributed to m^i_c can be at most $1 - \frac{1}{\alpha}$. This is because job j_i is also assigned to m^i_c and has size 1. We conclude that

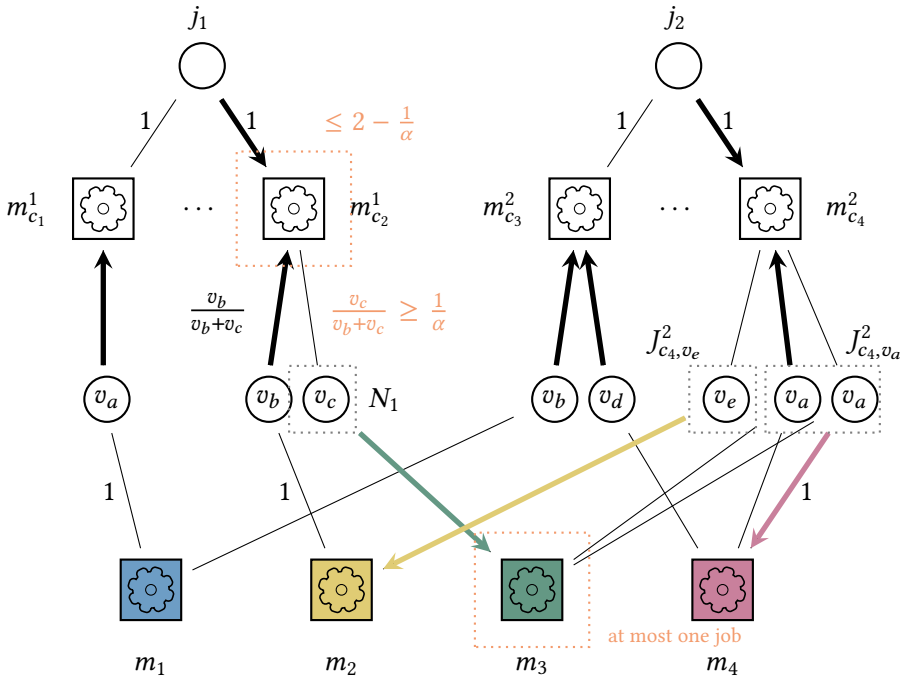
$$\sum_{j\in N_i : j\in J^i_{c,v}} \frac{v}{|c|} = 1 - \sum_{j\in J^i_c : j\in J^i_{c,v}} \frac{v}{|c|} \geq 1 - \left(1 - \frac{1}{\alpha}\right) = \frac{1}{\alpha}\,.$$

Since $|c| \geq 1$ by Step 1, we conclude that player i receives a total value of at least

$$\sum_{j\in R_i} v_{ij} = \sum_{j\in N_i : j\in J^i_{c,v}} v \geq \sum_{j\in N_i : j\in J^i_{c,v}} \frac{v}{|c|} \geq \frac{1}{\alpha}\,.$$



(a) α-approximation in SantaClaus instance I_S. Players are visualized by smileys and resources by gifts. Resource 1 can be assigned arbitrarily.



(b) $(2 - \frac{1}{\alpha})$-approximation in Makespan instance I_M. Machines are visualized by squares with gears and jobs by circles. The depicted solution selects configuration c_2 for Player 1 and configuration c_4 for Player 2. Note that Player 2 has the same value for resources 3 and 4.

Figure 3.1: The construction of Lemma 3.9: the given SantaClaus instance I_S in (a) and the constructed Makespan instance I_M in (b). In both pictures, edges indicate non-trivial values of resources / jobs for players / machines. The pictures show the reduction of an approximate solution for I_M to an approximate solution for I_S, which is used in Lemma 3.11. The SantaClaus instance I_S has 5 value types $\mathcal{T} = \{v_a, v_b, v_c, v_d, v_e\}$. In (b), there are only two configuration $\{c_1, c_2\} \subseteq \mathcal{C}_1$ for Player 1 and two configurations $\{c_3, c_4\} \subseteq \mathcal{C}_2$ for Player 2 depicted.

A visualization of this argument is given in Figure 3.1. This concludes the proof. □

3.3 Equivalence of Two-Value Problems

We move to the two-value cases. Recall that in the two-value case of SantaClaus and Makespan, all resource values and job sizes are in $\{u, w, 0\}$ and $\{u, w, \infty\}$, respectively, with $u, w \geq 0$. For these special cases, we prove an equivalence between establishing a better-than-2 approximation algorithm for Makespan and a constant-factor approximation algorithm for SantaClaus. More specifically, we show the following theorem in this section.

Theorem 3.12. *For any $\alpha \geq 2$, there exists an α-approximation algorithm for two-value SantaClaus if and only if there exists a $(2 - \frac{1}{\alpha})$-approximation algorithm for two-value Makespan.*

We prove this equivalence by showing the implication in both directions separately in the following two subsections.

3.3.1 Reduction of Santa Claus to Makespan

For this direction, we show that we can apply a similar reduction as in Theorem 3.4. Recall that the additional factor of $1 + \varepsilon$ comes from reducing the number of relevant configurations to polynomial in Lemma 3.6. However, we can observe that for the two-value case, there are naturally only polynomially many relevant configurations. To see this, suppose that $w = 1$ and $u = \frac{1}{b}$ for some integer b. Then, a player reaches a value of at least 1 whenever they receive one resource of value w or at least b resources of value u. Thus, the remaining task is to prove a similar argument in the general two-value case, and ensure that in the reduced Makespan instance we only have two non-trivial values. The desired implication then follows from the following lemma and Proposition 3.3.

Lemma 3.13. *Let I_S be a two-value SantaClaus instance with $\mathrm{OPT}(I_S) \geq 1$. For any $\alpha \geq 2$, we can construct in polynomial time a two-value Makespan instance I_M such that, given an $(2 - \frac{1}{\alpha})$-approximate solution for I_M, we can compute in polynomial time a solution for I_S with an objective value of at least $\frac{1}{\alpha}$.*

Proof. Let I_S be an instance of the two-value SantaClaus problem with $\mathrm{OPT}(I_S) \geq 1$ and $v_{ij} \in \{0, u, w\}$. We assume without loss of generality that $u \leq w$. We consider three exhaustive cases.

In the first case, we assume that $w < \mathrm{OPT}(I_S)/\alpha$. Then, we can use the algorithm of Bezáková and Dani [BD05] to compute in polynomial time a solution in which every player receives a total value of at least

$$\mathrm{OPT}(I_S) - \max_{i \in P, j \in R} v_{ij} = \mathrm{OPT}(I_S) - w > \left(1 - \frac{1}{\alpha}\right) \cdot \mathrm{OPT}(I_S) \geq \frac{1}{\alpha} \cdot \mathrm{OPT}(I_S)\,,$$

using $\alpha \geq 2$.

In the second case, we assume that $w \geq \mathrm{OPT}(I_S)/\alpha$ and that there is an optimal solution for I_S in which every player i receives a resource j of value $v_{ij} = w$. Then, we can set u to 0 and compute such a promised solution where every player receives a resource of value w in

polynomial time by solving a bipartite matching problem. Since $w \geq \mathrm{OPT}(I_S)/\alpha \geq \frac{1}{\alpha}$, we are done.

In the third case, we assume that $w \geq \mathrm{OPT}(I_S)/\alpha$ and that in every optimal solution for I_S there is some player i that does not receive a resource j of value $v_{ij} = w$. We can conclude that such a player must receive at least $b = \lceil 1/u \rceil$ many resources j' for which they have a value of $v_{ij'} = u$, because $\mathrm{OPT}(I_S) \geq 1$. Then, we construct a new instance I'_S by copying I_S and adjusting the resource values to $w' = 1$ and $u' = \frac{1}{b}$. Observe that $\mathrm{OPT}(I'_S) \geq 1$ and that any solution for I'_S in which every player either receives a resource of value 1 or b resources of value $\frac{1}{b}$ gives an objective value of at least 1. We can therefore define a collection of configurations $\mathcal{C}$ in which every player has these two configurations. Then, we can use Lemma 3.9 (by noting that in the constructed Makespan instance every job has either size 1 or $\frac{1}{b}$) to compute a solution for I'_S in which every player receives a total value of at least $\frac{1}{\alpha}$. Since $u \geq \frac{1}{b} = u'$ and $w \geq \frac{1}{\alpha} \cdot \mathrm{OPT}(I_S)$, this means that we can use the same solution for instance I and guarantee that every player receives a total value of at least $\frac{1}{\alpha}$. □

3.3.2 Reduction of Makespan to Santa Claus

For this direction, we cannot rely on Lemma 3.9. However, given a makespan instance I_M, we can use a "dual" construction of a SantaClaus instance I_S. Further, we can use similar observations as in the previous section. Suppose that $\mathrm{OPT}(I_M) \leq 1$ and that the two sizes are $w = 1$ and $u = \frac{1}{b}$ for some integer b. Then, a machine with load at most 1 can either have assigned one job of size w, or at most b jobs of size u. This keeps the number of relevant configurations small, and the implication follows from Proposition 3.3 and the following lemma.

Lemma 3.14. *Let I_M be a two-value Makespan instance with $\mathrm{OPT}(I_M) \leq 1$. For any $\alpha \geq 2$, we can construct in polynomial time a two-value SantaClaus instance I_S such that, given an α-approximate solution for I_S, we can compute in polynomial time a solution for I_M with an objective value of at most $2 - \frac{1}{\alpha}$.*

The remainder of this section is dedicated to the proof of this lemma. To this end, fix a two-value Makespan instance I_S with $\mathrm{OPT}(I_M) \leq 1$ and sizes in $\{u, w, \infty\}$ with $u \leq w$. For a fixed machine, we call jobs of size u small and jobs of size w big.

First, we observe that we can without loss of generality assume that $w > \frac{1}{2}\mathrm{OPT}(I_M)$. Otherwise, the algorithm by Lenstra et al. [LST90] gives us a solution of objective value at most $\mathrm{OPT}(I_M) + w \leq \frac{3}{2} \cdot \mathrm{OPT}(I_M)$. Since $\alpha \geq 2$, this solution satisfies the lemma for every possible α.

We now construct a two-value SantaClaus instance I_S as follows:

1. Let $b := \min\{\lfloor 1/u \rfloor, n\}$, where n is the number of jobs in I_M. That is, b denotes the maximal number of small jobs that can be assigned to every machine in an optimal solution with $\mathrm{OPT}(I_M) \leq 1$. By our assumption that $w \geq \frac{1}{\alpha} \cdot \mathrm{OPT}(I_S)$, there can be at most one big job placed on every machine.

2. For every machine i, we introduce a *machine-player* q_i, one *big resource* r_i, and b *small resources* $r_i^1, \ldots, r_i^b$. The value of the big resource r_i for player q_i is equal to w, the value of every small resource r_i^ℓ for player q_i is equal to u.

3. For every job j, we introduce a *job-player* $\hat{q}_j$. Furthermore, for every machine i, we set the value of resource r_i for $\hat{q}_j$ to w if $p_{ij} = w$ and to 0 otherwise. For every small resource r_i^ℓ, we set the value for $\hat{q}_j$ to w if $p_{ij} = u$ and to 0 otherwise.

Note that in I_M, every machine-player q_i has only values in $\{0, u, w\}$, and every job-player $\hat{q}_j$ has only values in $\{0, w\}$. Thus, I_M is a two-value SantaClaus instance. Further, for every machine, the number of introduced resources is at most the total number of jobs in I_S plus one, asserting that I_M is of polynomial size. An illustration of this construction is depicted in Figure 3.2.

We continue by proving two auxiliary lemmas on the constructed instance I_M, which together imply Lemma 3.14. To this end, let $t = w + b \cdot u - 1$ and note that $t \leq 1$ holds by construction. Also, observe that

$$t = w + b \cdot u - 1 \leq w + b \cdot u - b \cdot u = w \,. \tag{3.3}$$

Using this, we prove the following lemma.

Lemma 3.15. *The optimal objective value of I_S is at least t.*

Proof. Fix an optimal solution of I_M, and recall that we assume $\mathrm{OPT}(I_M) \leq 1$. In the following, we construct a solution for I_S.

Fix a machine i of instance I_M. If the given solution for I_M assigns a job j of size w to i, we assign resource r_i to job-player $\hat{q}_j$. If the given solution for I_M assigns a job j of size u to machine i, we assign an arbitrary unassigned small resource r_i^ℓ to job-player $\hat{q}_j$. All yet unassigned resources are assigned to their corresponding machine-players.

We continue by separately proving that each player in instance I' receives a value of at least t, which implies the lemma.

First, consider the job-players. Since every job j is assigned to exactly one machine in I_M, the job-player $\hat{q}_j$ receives exactly one resource in the constructed solution for I_S. These resources have value w for those players, giving them a sufficiently large value of $w \geq t$.

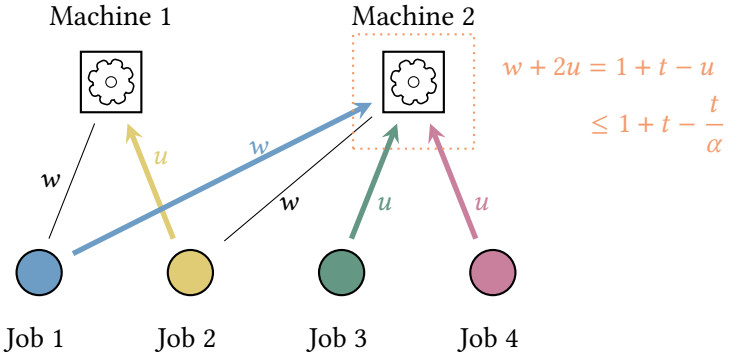
Next, we consider the machine-players. Fix a machine i. Since we assume $\mathrm{OPT}(I_M) \leq 1$, every machine i receives jobs of total size at most 1 in the solution for instance I_M. For our constructed solution to instance I_S, this means that the subset N_i of resources $r_i, r_i^1, \ldots, r_i^b$ that are *not* assigned to machine-player q_i satisfies $\sum_{r \in N_i} v_{q_i,r} \leq 1$. This implies that the value assigned to machine-player i is a least

$$w + b \cdot u - \sum_{r \in N_i} v_{q_i,r} \geq w + b \cdot u - 1 = t \,,$$

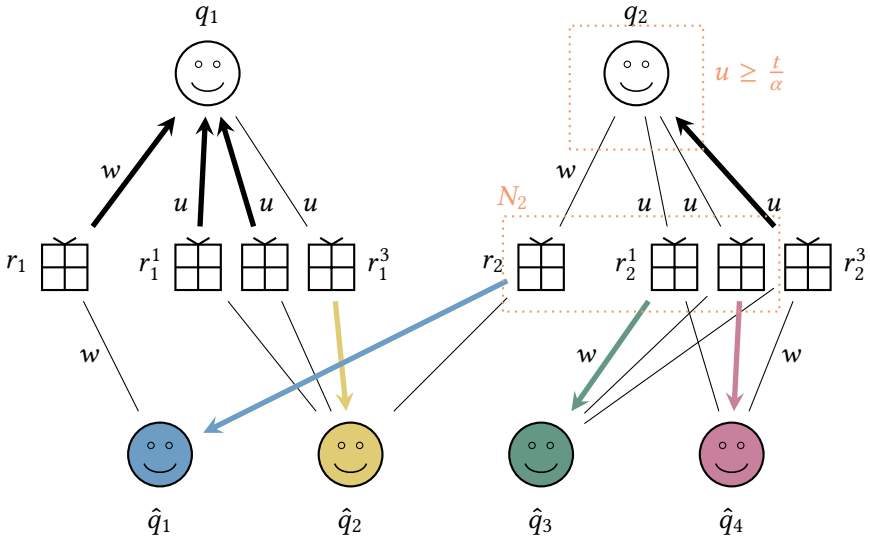
which implies $\mathrm{OPT}(I_S) \geq t$. □

Using Lemma 3.15, we can conclude that an α-approximation algorithm computes a solution for instance I_S where every player receives a total value of at least t/α . Thus, the following lemma implies Lemma 3.14.

Lemma 3.16. *For any $\alpha \geq 2$, given an solution for I_S where every player receives a total value of at least t/α, we can construct in polynomial time a solution for I_M of makespan at most $2 - \frac{1}{\alpha}$.*



(a) $(2 - \frac{1}{\alpha})$-approximation in two-value Makespan instance I_M. Machines are visualized by squares with gears and jobs by circles.



(b) α-approximation in two-value SantaClaus instance I_S. Players are visualized by smileys and resources by gifts.

Figure 3.2: The construction of Lemma 3.14: the given two-value Makespan instance I_M in (a) and the constructed two-value SantaClaus instance I_S in (b). In both pictures, edges indicate non-trivial values of resources / jobs for players / machines. The pictures show the reduction of an approximate solution for I_S to an approximate solution for I_M, which is used in Lemma 3.16. Note that in this example, $b = 3$ and $t = w + 3u - 1$.

Proof. Clearly, in the given solution for I_S, every job-player receives at least one resource of positive value, as otherwise the objective value would be zero. We modify the given solution in a way such that each job-player receives exactly one resource of positive value. If a job-player receives more than one resource, we select an arbitrary resource of positive value and reassign all other resources to their corresponding machine-players. Since the only positive value of resources for job-players is w, the single resource that remains assigned to a job-player gives them a value of at least t because of (3.3). Thus, the modified solution for I_S still has an objective value of at least t/α.

Now, we can construct a solution for I_M as follows. If one of the resources $r_i, r_i^1, \dots, r_i^b$ belonging to machine i has been assigned to a job-player $\hat{q}_j$, we assign job j to machine i. By the above assumption this assignment is feasible.

It remains to argue about the load of every machine in I_M. Fix a machine i. In the solution for instance I_S, the corresponding machine-player q_i receives resources of value at least t/α. This means that the subset N_i of resources $r_i, r_i^1, \dots, r_i^b$ that are *not* assigned to machine-player q_i satisfies

$$\sum_{r \in N_i} v_{q_i,r} \leq w + b \cdot u - \frac{t}{\alpha} = 1 + t - \frac{t}{\alpha} .$$

Since by construction $t \leq 1$, we have $t - t/\alpha \leq 1 - \frac{1}{\alpha}$, which implies $\sum_{r \in N_i} v_{q_i,r} \leq 2 - \frac{1}{\alpha}$. We conclude the proof by observing that, by construction, the load of machine i in the constructed solution for I_M is equal to $\sum_{r \in N_i} v_{q_i,r} \leq 2 - \frac{1}{\alpha}$.

A visualization of this argument is given in Figure 3.2. □

3.4 Matroid Allocation Problems

In this section, we introduce generalizations of SantaClaus and Makespan and prove basic results for them. The key feature of these generalizations, in the case of Makespan, is that a single job can be assigned to multiple machines at the same time, possibly also multiple times to one machine. It is instructive to think of assigning *copies* of this job to machines. The configurations in which a job j can be copied to machines are specified by bases $\mathcal{B}(\mathcal{P}_j)$ of an integer polymatroid $\mathcal{P}_j$ over the set of machines: given a basis x_j, we copy the job $x_j(i)$ times to machine i. We now give formal definitions.

The Matroid Santa Claus Problem. In the matroid SantaClaus problem, there are sets of m players P and n resources R with values $v_{ij} \geq 0$ for all $j \in R$ and $i \in P$. Further, for every resource $j \in R$ there is an integer polymatroid $\mathcal{P}_j$ over P. The task is to allocate each resource $j \in R$ to a basis $x_j \in \mathcal{B}(\mathcal{P}_j)$ and let each player i profit from the resource j with value $v_{ij} \cdot x_j(i)$. The goal is to maximize the minimum total value any player receives, that is, $\min_{i \in P} \sum_{j \in R} v_{ij} \cdot x_j(i)$.

The Matroid Makespan Problem. In the matroid Makespan problem, there are sets of m machines M and n jobs J with sizes $p_{ij} \geq 0$ for all $j \in J$ and $i \in M$. Further, for every job $j \in J$ there is an integer polymatroid $\mathcal{P}_j$ over M. The task is to allocate each job $j \in J$ to a basis

$x_j \in \mathcal{B}(\mathcal{P}_j)$, which means that j contributes load $p_{ij} \cdot x_j(i)$ to the total load of machine i. The goal is to minimize the maximum total load over all machines, that is, $\max_{i \in M} \sum_{j \in J} p_{ij} \cdot x_j(i)$.

These new matroid allocation problems generalize SantaClaus and Makespan, respectively. In fact, the variants that use polymatroids of rank 1 correspond to the respective traditional problems.

Note that in matroid SantaClaus, it is equivalent to require that $x_j \in \mathcal{P}_j$ for each resource j instead of $x_j \in \mathcal{B}(\mathcal{P}_j)$, since we can always increase x_j to reach a basis without making the solution worse. In matroid Makespan this is not the case. These properties are analogue to the traditional problems.

Finally, we note that special cases such as the restricted and the two-value case naturally generalize to the matroid problems.

The proofs in this section rely on properties of the polymatroid intersection problem, which we briefly define. In the polymatroid intersection problem, we are given two polymatroids $\mathcal{P}$ and $\mathcal{P}'$ over the same universe E, and the task is to find an element $y \in \mathcal{P} \cap \mathcal{P}'$ that maximizes $\sum_{e \in E} y(e)$. For further details, we refer to [Sch03, Chapter 41].

3.4.1 Reducing the Number of Polymatroids in the Restricted Case

In the restricted case, both matroid problems can be reduced to instances where the number of polymatroids is equal to the number of distinct job sizes / resource values. This is because we can sum up polymatroids associated with jobs / resources of equal size / value to a single one, and then decompose a basis for such a summed polymatroid into bases for the original polymatroids via polymatroid intersection. Formally, we get the following proposition.

Proposition 3.17. *For any $\alpha \geq 1$, if there exists a polynomial-time α-approximation algorithm for restricted matroid Makespan (matroid SantaClaus) with h jobs (resources), then there exists a polynomial-time α-approximation algorithm for restricted matroid Makespan (matroid SantaClaus) with $p_j \in \{w_1, \ldots, w_h\}$ ($v_j \in \{w_1, \ldots, w_h\}$) and $w_1, \ldots, w_h \geq 0$.*

Proof. Let I be an instance of the restricted matroid Makespan problem with machines E and h distinct processing times $p_1, \ldots, p_h$. Let J_ℓ, $\ell \in [h]$, denote the set of jobs with processing times p_ℓ. Further, let $\mathcal{P}_j^\ell$ with $\ell \in [h]$ and $j \in J_\ell$ denote the corresponding polymatroids over E and let f_j^ℓ be the associated submodular function.

We construct an instance I' of the restricted matroid Makespan problem with h jobs by using the same set of machines E and creating the polymatroids $\mathcal{P}_\ell$ for $\ell \in [h]$ with the monotone submodular function $f_\ell(S) = \sum_{j \in J_\ell} f_j^\ell(S)$ for every subset $S \subseteq E$. Note that $\mathcal{P}_\ell = \sum_{j \in J_\ell} \mathcal{P}_j^\ell$ [Sch03]. For $\ell \in [h]$, the goal in instance I' is to find vectors $x_\ell \in \mathcal{B}(\mathcal{P}_\ell)$ such that $\max_{e \in E} \sum_{\ell \in [h]} p_\ell \cdot x_\ell(e)$ is minimized. We prove that this reduction preserves the approximation factor.

Consider a solution of instance I that selects the bases x_j^ℓ for job $j \in J_\ell$ with $\ell \in [h]$ and consider the vectors x'_ℓ with $x'_\ell(e) = \sum_{j \in J_\ell} x_j^\ell(e)$ for all $e \in E$. Using again $\mathcal{P}_\ell = \sum_{j \in J_\ell} \mathcal{P}_j^\ell$, we have $x'_\ell \in \mathcal{P}_\ell$ for all $\ell \in [h]$. In particular, $x'_\ell(E) = \sum_{e \in E} \sum_{j \in J_\ell} x_j^\ell(e) = \sum_{j \in J_\ell} f_j^\ell(E) = f_\ell(E)$, so x'_ℓ is a basis of $\mathcal{P}_\ell$. Thus, $(x'_1, \ldots, x'_h)$ is a feasible solution for instance I'. Furthermore,

$$\mathrm{OPT}(I') \leq \max_{e \in E} \sum_{\ell \in [h]} x'_\ell(e) \cdot p_\ell = \max_{e \in E} \sum_{\ell \in [h]} \sum_{j \in J_\ell} x_j^\ell(e) \cdot p_\ell = \mathrm{OPT}(I) \ .$$

Consider some solution $(y'_1, \dots, y'_h)$ to instance I', that is, $y'_\ell \in \mathcal{B}(\mathcal{P}_\ell)$ and $f_\ell(E) = y'_\ell(E)$ for all $\ell \in [h]$. We construct a solution to I by decomposing each y'_ℓ, $\ell \in [h]$, into bases $y^\ell_j \in \mathcal{B}(\mathcal{P}^\ell_j)$ such that $y'_\ell(e) = \sum_{j \in J_\ell} y^\ell_j(e)$ holds for all $e \in E$. As $\mathrm{OPT}(I') \leq \mathrm{OPT}(I)$, this implies that the reduction preserves the approximation factor. If such decomposition would not exist for some $\ell \in [h]$, then, by construction of the submodular function f_ℓ, we would arrive at a contradiction to $f_\ell(E) = y'_\ell(E)$.

To find the decomposition for an $\ell \in [h]$ in polynomial time, consider the polymatroids $\hat{\mathcal{P}}^\ell_j$ that are just copies of the original polymatroids $\mathcal{P}^\ell_j$ on pairwise disjoint copies $\hat{E}_j$ of the ground set E. For each $\ell \in [h]$, we decompose the solution y'_ℓ of instance I' into bases of the copy polymatroids, which then implies a decomposition into bases of the original polymatroids. For each $e \in E$, let C_e denote the set of copies of e introduced by the ground set copies. We want to find a basis $\hat{y}^\ell_j$ for every $j \in J_\ell$ such that $\sum_{\hat{e} \in C_e} \hat{y}^\ell_j(\hat{e}) = y'_\ell(e)$ holds for all $e \in E$ and $\ell \in [h]$. For an element $e \in E$ and $\ell \in [h]$, consider the polymatroid $\mathcal{X}^\ell_e$ on the ground set C_e implied by bases $\mathcal{B}(\mathcal{X}^\ell_e) = \{x \in \mathbb{Z}^{C_e}_{\geq 0} : x(C_e) = y'_\ell(e)\}$ and let $\mathcal{X}^\ell$ denote the union of these polymatroids. Furthermore, let $\hat{\mathcal{P}}^\ell$ denote the union of the polymatroids $\hat{\mathcal{P}}^\ell_j$. The largest element in the intersection of $\mathcal{X}^\ell$ and $\hat{\mathcal{P}}^\ell$ gives us the decomposition. We can compute the largest element in the intersection in polynomial time using algorithms for polymatroid intersection (cf. [Sch03, Chapter 41]).

The statement for matroid SantaClaus can be shown with the same reduction and proof; only the inequality $\mathrm{OPT}(I') \leq \mathrm{OPT}(I)$ trivially changes to $\mathrm{OPT}(I') \geq \mathrm{OPT}(I)$. □

3.4.2 Rounding Theorems for the Assignment LP

In this section, we show that the rounding theorems by Bezáková and Dani [BD05], Lenstra et al. [LST90], and Shmoys and Tardos [ST93] for the traditional SantaClaus and Makespan problems can be lifted to our matroid generalizations. To this end, we mainly employ a rounding technique for the assignment LP introduced by Shmoys and Tardos [ST93].

We start with the matroid SantaClaus problem. We first introduce the assignment LP for this problem. We have one fractional variable $x_j(i)$ for every resource j and every player i that indicates how many (fractional) copies of resource j player i receives in the solution. Let f_j be the submodular function associated with resource j's polymatroid. Then, the assignment LP defines a set of feasible solutions that give every player a total value of at least T via the following set of linear constraints:

$$\sum_{j \in R} x_j(i) \cdot v_{ij} \geq T \qquad \forall i \in P \tag{3.4}$$

$$\sum_{i \in S} x_j(i) \leq f_j(S) \qquad \forall S \subseteq P \; \forall j \in R \tag{3.5}$$

$$\sum_{i \in P} x_j(i) = f_j(P) \qquad \forall j \in R \tag{3.6}$$

$$x_j \geq 0 \qquad \forall j \in R \tag{3.7}$$

Note that Constraint (3.5) ensures that x_j is contained in the fractional relaxation of j's polymatroid $\mathcal{P}_j$. Moreover, Constraint (3.6) requires that x_j is a basis of the fractional relaxation of $\mathcal{P}_j$.

Our rounding theorem for the matroid SantaClaus problem now guarantees, for a given fractional solution, an integral solution with a close objective value.

Theorem 3.18. *Given an instance I of the matroid SantaClaus problem and a feasible solution x for the assignment LP that gives every player a total value of at least T, we can compute in polynomial time a solution for I with objective value at least $T - \max_{i\in P, j\in R} v_{ij}$.*

Proof of Theorem 3.18. Let $P = \{1, \dots, m\}$ and $R = \{1, \dots, n\}$. We first define for every player i an ordering σ_i over the resources by the a non-increasing order of their values. That is, $v_{i\sigma_i(1)} \geq v_{i\sigma_i(2)} \geq \cdots \geq v_{i\sigma_i(n)}$. We assume without loss of generality that $v_{i\sigma_i(n)} = 0$, which can always be achieved by adding a dummy resource. For every resource $j \in R$, we denote its associated polymatroid by $\mathcal{P}_j$ and the corresponding submodular function by f_j.

We now define a polymatroid intersection problem based on x. The universe E and the two polymatroids $\mathcal{P}$ and $\mathcal{P}'$ are based on following bipartite graph. On the left side of the graph, we have a set of vertices W with one vertex w_j for each resource j, and on the right side we have a set of vertices U with one vertex u_{ij} for each player i and the resource that appears in jth position in the ordering σ_i of player i. The set of edges is given by

$$E = \{(w_{\sigma_i(j)}, u_{ij})\}_{i\in P, j\in R} \cup \{(w_{\sigma_i(j)}, u_{i(j+1)})\}_{i\in P, j\in R\setminus\{n\}} .$$

For some edge $e \in E$, we denote by e_w the resource corresponding to its left side vertex, and by e_u the player corresponding to its right side endpoint.

For the polymatroid intersection problem, we use the set of edges E as universe. The first polymatroid $\mathcal{P}$ is defined via the submodular function f that satisfies for every $S \subseteq E$

$$f(S) \coloneqq \sum_{j=1}^{n} f_j\left(\bigcup_{e\in S: e_w = j} e_u\right) .$$

The second polymatroid $\mathcal{P}'$ is defined via the submodular function f' that itself is defined via the right side of the bipartite graph. We define below for every vertex $u \in U$ a *degree constraint* $d(u)$, and then have for every $S \subseteq E$

$$f'(S) \coloneqq \sum_{u\in U: (w,u)\in S} d(u) .$$

We define the degree constraints using the following process for every player i. We start with

$$d(u_{i1}) \coloneqq \left\lfloor x_{\sigma_i(1)}(i) \right\rfloor ,$$

and define the *remainder* $R_1 \coloneqq x_{\sigma_i(1)}(i) - d(u_{i1})$ (for ease of notation we define $R_0 = 0$). Then, we define recursively the degree constraint $d(u_{ij})$ and remainders as follows. (We use the notation $\{a\} \coloneqq a - \lfloor a \rfloor$ for every $a \in \mathbb{R}_{\geq 0}$ for the fractional part of a.)

$$\begin{aligned} d(u_{ij}) &\coloneqq \left\lfloor R_{j-1} + x_{\sigma_i(j)}(i) \right\rfloor \text{ , and} \\ R_j &\coloneqq \{R_{j-1} + x_{\sigma_i(j)}(i)\} = R_{j-1} + x_{\sigma_i(j)}(i) - \left\lfloor R_{j-1} + x_{\sigma_i(j)}(i) \right\rfloor \\ &= R_{j-1} + \{x_{\sigma_i(j)}(i)\} - \left\lfloor R_{j-1} + \{x_{\sigma_i(j)}(i)\} \right\rfloor \text{ .} \end{aligned}$$

We now argue that there exists an element $y \in \mathcal{P} \cap \mathcal{P}'$ such that for every $S \subseteq E$ holds that $\sum_{e \in S} y(e) = f'(S)$, that is, y is a basis of $\mathcal{P}'$. To see this, fix a player i. The first resource (in the order defined by player i) can be assigned to vertex u_{i1} up to an extent of $\left\lfloor x_{\sigma_i(1)}(i) \right\rfloor$, which is represented in our graph as taking $\left\lfloor x_{\sigma_i(1)}(i) \right\rfloor$ copies of the edge $(w_{\sigma_i(1)}, u_{i1})$, that is, $y((w_{\sigma_i(1)}, u_{i1})) = \left\lfloor x_{\sigma_i(1)}(i) \right\rfloor$. Note that this makes the degree constraint at u_{i1} tight, as $y((w_{\sigma_i(1)}, u_{i1})) = d(u_{i1})$. The remaining fraction of $x_{\sigma_i(1)}(i)$ can be assigned to player i by taking the edge $(w_{\sigma_i(1)}, u_{i2})$ fractionally by $\{x_{\sigma_i(1)}(i)\}$, that is, $y((w_{\sigma_i(1)}, u_{i2})) = \{x_{\sigma_i(1)}(i)\}$. Note that this fraction is equal to R_1. Then, we move on to the second resource, and we take the edge $(w_{\sigma_i(2)}, u_{i2})$ by the maximal amount possible until the degree constraint $d(u_{i2})$ of vertex u_{i2} becomes tight. That is, we set $y((w_{\sigma_i(2)}, u_{i2})) = d(u_{i2}) - R_1$, because the amount equal to R_1 at vertex u_{i2} is already taken by $(w_{\sigma_i(1)}, u_{i2})$. We again see that there might be some leftover of the value $x_{\sigma_i(2)}(i)$ that was not assigned to u_{i2} and that is precisely equal to $x_{\sigma_i(2)}(i) - d(u_{i2}) - R_1 = R_2$. But this amount we assign to u_{i3} We continue this assignment until the last resource. Note that our definition of remainder R_j is precisely this small leftover of $x_{\sigma_i(j)}(i)$ that carries over to the edge going to vertex $u_{i(j+1)}$ (note that this remainder is always less than 1). One slight caveat at the end is that a some small amount of the fractional assignment of $x_{\sigma_i(n)}(i)$ might be thrown away, but this will be no issue as we assume that $v_{i\sigma_i(n)} = 0$.

Thus, the assignment y makes all degree constraints tight, implying $y \in \mathcal{B}(\mathcal{P}')$. Moreover, observe that since we distribute for every resource j in y a value of at most $\sum_{i \in P} x_{\sigma_i(j)}(i)$ to edges in $\{(w_{\sigma_i(j)}, u_{ij}), (w_{\sigma_i(j)}, u_{i,\min\{n,j+1\}})\}_{i \in P}$, we have that $y \in \mathcal{P}$.

We showed that by construction of the polymatroid intersection problem, there always exists an element $y \in \mathcal{P} \cap \mathcal{P}'$ such that $y \in \mathcal{B}(\mathcal{P}')$. Since $\mathcal{P}$ and $\mathcal{P}$ are integer polymatroids, the integrality of the polymatroid intersection polytope (cf. Corollary 46.1a in [Sch03]) implies that there exists an integral solution $z \in \mathcal{P} \cap \mathcal{P}'$ such that $z \in \mathcal{B}(\mathcal{P}')$. Further, we can compute such an integral solution by finding the maximum cardinality multiset of edges in G that belongs to the polymatroids intersection, which can be algorithmically done in polynomial time using a b-matching algorithm in a multigraph extension of G.

We can now interpret such an integral solution z as an integral assignment of resources to players in the matroid SantaClaus instance I: we assign $z((w_{\sigma_i(j)}, u_{ij})) = d(u_{ij})$ copies of resource j to player i. In this assignment, every player i receives a total value of at least

$$\sum_{j=1}^{n} \left\lfloor R_{j-1} + x_{\sigma_i(j)}(i) \right\rfloor \cdot v_{i\sigma_i(j)} \text{ .}$$

To conclude the theorem, we now compute for every player i the difference Δ_i between the total value player i receives in the given fractional solution x and the above value. We have that

$$\begin{aligned}\Delta_i &\leq \sum_{j=1}^{n} \left(x_{\sigma_i(j)}(i) - \left\lfloor R_{j-1} + x_{\sigma_i(j)}(i) \right\rfloor\right) \cdot v_{i\sigma_i(j)} \\ &= \sum_{j=1}^{n} \left(\{x_{\sigma_i(j)}(i)\} - \left\lfloor R_{j-1} + \{x_{\sigma_i(j)}(i)\} \right\rfloor\right) \cdot v_{i\sigma_i(j)} \ .\end{aligned}$$

Looking at each term of the sum individually, we notice that either $\left\lfloor R_{j-1} + \{x_{\sigma_i(j)}(i)\} \right\rfloor = 0$, or that $\left\lfloor R_{j-1} + \{x_{\sigma_i(j)}(i)\} \right\rfloor = 1$. In the first case, the next remainder R_j is equal to $R_{j-1} + \{x_{\sigma_i(j)}(i)\} < 1$. In the second case, we have that

$$R_j = R_{j-1} + \{x_{\sigma_i(j)}(i)\} - \left\lfloor R_{j-1} + \{x_{\sigma_i(j)}(i)\} \right\rfloor = R_{j-1} + \{x_{\sigma_i(j)}(i)\} - 1 \ .$$

Let R' denote the set of all indices where the second case happens. Then, we can write

$$\Delta_i \leq \sum_{j=1}^{n} \{x_{\sigma_i(j)}(i)\} \cdot v_{i\sigma_i(j)} - \sum_{j \in R'} v_{i\sigma_i(j)} \ .$$

Now, note that if $j \notin R'$ then $\{x_{\sigma_i(j)}(i)\} = R_j - R_{j-1}$, and that if $j \in R'$ then $\{x_{\sigma_i(j)}(i)\} = 1 + R_j - R_{j-1}$. Using this observation, we obtain

$$\begin{aligned}\Delta_i &\leq \sum_{j=1}^{n} (R_j - R_{j-1}) \cdot v_{i\sigma_i(j)} + \sum_{j \in R'} v_{i\sigma_i(j)} - \sum_{j \in R'} v_{i\sigma_i(j)} \\ &= \sum_{j=1}^{n} (R_j - R_{j-1}) \cdot v_{i\sigma_i(j)} = \sum_{j=1}^{n-1} (v_{i\sigma_i(j)} - v_{i\sigma_i(j+1)}) \cdot R_j \leq v_{i\sigma_i(1)} \ ,\end{aligned}$$

where we use the facts that $R_0 = 0$, $v_{i\sigma_i(n)} = 0$, and $R_j \leq 1$ for all resources j.

Therefore, our computed integral assignment gives every player i a total value of at least

$$\sum_{j=1}^{n} \left\lfloor R_{j-1} + x_{\sigma_i(j)}(i) \right\rfloor v_{i\sigma_i(j)} \geq \left(\sum_{j=1}^{n} x_{\sigma_j(i)}(i) \cdot v_{i\sigma_j(i)}\right) - \Delta_i \geq T - v_{i\sigma_i(1)} \geq T - \max_{i \in P, j \in R} v_{ij} \ ,$$

which concludes the proof of the theorem. □

We continue with the matroid Makespan problem. Similarly to the assignment LP for the matroid Makespan problem, we have one variable $x_j(i)$ for every job j and every machine i.

Then, the assignment LP defines a set of feasible solutions with makespan at most T via the following constraints:

$$\sum_{j \in J} x_j(i) \cdot p_{ij} \leq T \qquad \forall i \in M \tag{3.8}$$

$$\sum_{i \in S} x_j(i) \leq f_j(S) \qquad \forall S \subseteq M \, \forall j \in J \tag{3.9}$$

$$\sum_{i \in M} x_j(i) = f_j(M) \qquad \forall j \in J \tag{3.10}$$

$$x_j(i) = 0 \qquad \forall j \in J, i \in M \text{ s.t. } p_{ij} > T \tag{3.11}$$

$$x_j \geq 0 \qquad \forall j \in J \tag{3.12}$$

Constraints (3.8), (3.9), and (3.10) correspond to (3.4), (3.5), and (3.6) of the matroid SantaClaus problem. Constraint (3.11) however is specific to the Makespan problem. The idea is that, since no integral solution can assign job j to machine i if $p_{ij} > T$, we can safely disallow even a fractional assignment of j to i. This idea is also called *parametric pruning* and already appeared in the proofs of the rounding theorems for the non-matroid Makespan problem [LST90; ST93].

Given a fractional assignment that satisfies the constraints of the assignment LP, our rounding theorem for the matroid Makespan problem now guarantees an integral solution with a close objective value.

Theorem 3.19. *Given an instance I of the matroid Makespan problem and given a feasible solution x for the assignment LP with makespan at most T, we can compute in polynomial time a solution for I with makespan at most $T + \max\{p_{ij} \mid i \in M, j \in J \text{ s.t. } p_{ij} < T\}$.*

Since the proof of Theorem 3.19 is very similar to the proof of Theorem 3.18, we omit it here and refer to [Bam+24].

3.4.3 Reducing Restricted Santa Claus to Restricted Two-Value Santa Claus

The goal of this section is to prove the following lemma, which allows us to heavily reduce restricted matroid SantaClaus instances to instances with only one matroid and one polymatroid.

Lemma 3.20. *For any $\alpha \geq 2$, if there is a polynomial-time algorithm that, given an instance of restricted two-value matroid SantaClaus problem with one matroid of value $v_1 = \infty$ and one (integer) polymatroid of value $v_2 = 1$ and a number $b \in \mathbb{N}$, finds a solution of value at least b or determines that there is no solution of value αb, then there is also:*

1. *a polynomial-time α-approximation algorithm for (any instance of) the restricted two-value matroid SantaClaus problem and*
2. *a polynomial-time 2α-approximation algorithm for (any instance of) the restricted matroid SantaClaus problem.*

The above lemma implies the following reduction.

Corollary 3.21. *For any $\alpha \geq 2$, if there is a polynomial-time α-approximation algorithm for the restricted two-value matroid* SantaClaus *problem, then there also is a polynomial-time 2α-approximation algorithm for the restricted matroid* SantaClaus *problem.*

Proof of Lemma 3.20. We start by proving the first result of the lemma. Let u and w with $w \geq u$ be the two sizes of a given restricted two-value matroid SantaClaus instance I. By Proposition 3.17, we can assume that there are exactly two resources in I, one for each size, with corresponding polymatroids $\mathcal{P}_u$ and $\mathcal{P}_w$. Let f_u and f_w be the associated submodular functions. There are three possible cases.

If $\mathrm{OPT}(I)/\alpha \leq u$, then it suffices to give at least one resource to any player to obtain an α-approximate solution. This requires to check whether the vector of ones is in the union of $\mathcal{P}_u$ and $\mathcal{P}_w$, which can be done in polynomial time [Sch03].

If $u < \mathrm{OPT}(I)/\alpha \leq w$, then, in an α-approximate solution, it suffices to give to each player either one resource of value w or $\frac{1}{\alpha \cdot u}\mathrm{OPT}(I)$ resources of value u. We define a new instance I_2 over the same set of players with one matroid of value $v_1 = \infty$ and one polymatroid of value $v_2 = 1$. The independent sets of the matroid are the sets of players that can be covered by at least one resource of value w each in the original instance. The polymatroid is defined by the submodular function $f_2 = f_u$. Clearly, in this case, we have that $\mathrm{OPT}(I_2) \geq \frac{1}{u}\mathrm{OPT}(I)$, and a solution of value T in instance I_2 can immediately be translated to a solution of value $\min\{\mathrm{OPT}(I)/\alpha, Tu\}$ in the original instance. So an α-approximation on instance I_2 gives us an α-approximation on the original instance.

In the last case where $\mathrm{OPT}(I)/\alpha > w$, we first assume without loss of generality by scaling that u and w are integers. Then, we define an integer polymatroid $\mathcal{P}_3$ over the set of players P with the integer submodular function $f_3(S) = u \cdot f_u(S) + w \cdot f_w(S)$; this should be thought of splitting the resources of value w (respectively u) into w (respectively u) individual resources of value 1 each. We can then compute the largest integer b such that $b \cdot P \in \mathcal{P}_3$ (where $b \cdot P$ is the $|P|$ dimensional vector with all entries equal to b) by binary search and checking whether the submodular function $f_3'(S) = f_3(S) - b \cdot |S|$ is non-negative. The latter can be done by computing the minimum value of $f_3'(S)$ in polynomial time (cf. Theorem 45.1 in [Sch03]). By the construction of f_3, we can decompose $b \cdot P$ into a fractional solution for instance I with an objective value equal to $\mathrm{OPT}(I)$. Using Theorem 3.18, we can then round in polynomial time this fractional solution into an integral solution with an objective value of at least

$$\mathrm{OPT}(I) - \max\{u, w\} \geq \left(1 - \frac{1}{\alpha}\right) \cdot \mathrm{OPT}(I) \geq \frac{1}{\alpha}\mathrm{OPT}(I)\,,$$

using $\alpha \geq 2$. This concludes the proof of the first point of the lemma.

For the second point, using the binary search framework (cf. Proposition 3.3), we can assume that we know the optimal objective value $\mathrm{OPT}(I)$. We call a resource j *heavy* if $v_j \geq \frac{1}{2\alpha}\mathrm{OPT}(I)$ and *light* otherwise. Let H and L be the set of heavy and light resources, respectively. We then define an instance I' with the same set of players and with one matroid of value ∞, whose independent sets are the sets of players that can be covered by at least one heavy resource each, which is a matroid by the matroid union theorem [Sch03, Chapter 42]. Again, assuming that all values v_j are integers, we define one polymatroid of value 1 associated to the submodular function $f'(S) := \sum_{j \in L} v_j \cdot f_j(S)$. In this new instance, it is clear that $\mathrm{OPT}(I') \geq \mathrm{OPT}(I)$. Hence

an α-approximate solution to instance I' can be transformed into an α-approximate solution to instance I, in which the heavy resources are assigned integrally, and the light resources fractionally. Using Theorem 3.18, we can round this fractional assignment in polynomial time into an integral assignment with an objective value of at least

$$\frac{1}{\alpha}\mathrm{OPT}(I) - \max_{j \in L} v_j \geq \frac{1}{\alpha}\mathrm{OPT}(I) - \frac{1}{2\alpha}\mathrm{OPT}(I) = \frac{1}{2\alpha}\mathrm{OPT}(I) \ .$$

This concludes the proof of the second point of the lemma. □

3.5 Equivalence of Restricted Two-Value Matroid Problems

In this final section of this chapter, we prove the equivalence of matroid SantaClaus and matroid Makespan in terms of approximation within the restricted two-value case. We prove the following theorem.

Theorem 3.22. *For any $\alpha \geq 2$, there exists an α-approximation algorithm for restricted two-value matroid SantaClaus if and only if there exists a $(2 - \frac{1}{\alpha})$-approximation algorithm for restricted two-value matroid Makespan.*

We prove this equivalence by showing the implication in both directions separately in the following two subsections. Both directions heavily rely on *polymatroid duality*, which we briefly explain now. For a given integer polymatroid $\mathcal{P}$ over a universe E with associated submodular function $f : 2^E \to \mathbb{Z}_{\geq 0}$ and some vector $z \in \mathbb{Z}^E_{\geq 0}$ with $x \leq z$ for all $x \in \mathcal{P}$, the *dual polymatroid* $\overline{\mathcal{P}}$ of $\mathcal{P}$ with respect to z is defined via the integer set function $g : 2^E \to \mathbb{Z}_{\geq 0}$ with

$$g(S) = \sum_{e \in S} z(e) + f(E \setminus S) - f(E)$$

for every $S \subseteq E$. This function is submodular, monotone, and satisfies $g(\emptyset) = 0$, hence $\overline{\mathcal{P}}$ is in fact a polymatroid. In particular, a value oracle for g can be simulated in polynomial time using a value oracle for f. The important property we exploit below is that the set of bases of $\mathcal{P}$ and $\overline{\mathcal{P}}$ are dual to each other in the following sense: for every $x \in \mathcal{B}(\mathcal{P})$ it follows $g(E) = \sum_{e \in E} z(e) + f(\emptyset) - f(E) = \sum_{e \in E} z(e) - x(E)$, and therefore $z - x \in \mathcal{B}(\overline{\mathcal{P}})$.

For an integer b and a set S, we again write $b \cdot S$ to denote the $|S|$-dimensional vector where each entry is equal to b.

3.5.1 Reduction of Matroid Santa Claus to Matroid Makespan

We start with reducing matroid SantaClaus to matroid Makespan. This direction in Theorem 3.22 follows from the following lemma combined with the binary search framework (Proposition 3.3).

Lemma 3.23. *Let $\alpha \geq 2$ and I be an instance of the restricted two-value matroid SantaClaus problem such that $\mathrm{OPT}(I) \geq 1$. Then, we can compute an instance I' of the restricted two-value matroid Makespan problem, such that, given a $(2-\frac{1}{\alpha})$-approximate solution for I', we can compute a solution for I with an objective value of at least $\frac{1}{\alpha}$.*

Proof. Let I be an instance of the restricted two-value matroid SantaClaus problem with players E and two resources with associated polymatroids $\mathcal{P}_1, \mathcal{P}_2$ and values $v_1, v_2 \geq 0$ such that $\mathrm{OPT}(I) \geq 1$. By Lemma 3.20, we can assume that we are given a simplified case of the problem. For convenience, we slightly reformulate it as follows: given $v_1 = 1$ (instead of ∞), $v_2 = \frac{1}{b} \leq v_1$ (instead of 1) for $b \in \mathbb{N}$, and $\mathrm{OPT}(I) \geq 1$, find a solution of value at least $\frac{1}{\alpha}$.

Consider the truncated polymatroids $\mathcal{P}'_1 = \{x \in \mathcal{P}_1 : x(e) \leq 1 \ \forall e \in E\}$ and $\mathcal{P}'_2 = \{x \in \mathcal{P}_2 : x(e) \leq b \ \forall e \in E\}$. Let $\overline{\mathcal{P}}_1$ be the dual polymatroid of $\mathcal{P}'_1$ with respect to the vector $1 \cdot E$, and let $\overline{\mathcal{P}}_2$ be the dual polymatroid of $\mathcal{P}'_2$ with respect to the vector $b \cdot E$. We compose an instance I' of matroid Makespan using machines E, one job of size $p_1 = 1$ with polymatroid $\overline{\mathcal{P}}_1$ and one job of size $p_2 = \frac{1}{b}$ with polymatroid $\overline{\mathcal{P}}_2$.

We first show that $\mathrm{OPT}(I') \leq 1$. Fix an optimal solution for I that selects bases $x_1 \in \mathcal{B}(\mathcal{P}_1)$ and $x_2 \in \mathcal{B}(\mathcal{P}_2)$. Since $\mathrm{OPT}(I) \geq 1$, we have $v_1 \cdot x_1(e) + v_2 \cdot x_2(e) \geq 1$ for every $e \in E$. Further, $v_1 = 1$ and $v_2 = \frac{1}{b}$ that implies $x_1(e) + \frac{1}{b} \cdot x_2(e) \geq 1$. Our goal is to dualize bases of $\mathcal{P}'_1$ and $\mathcal{P}'_2$ to obtain bases of $\overline{\mathcal{P}}_1$ and $\overline{\mathcal{P}}_2$, which are feasible for I'. To this end, we first construct vectors $x'_1 \in \mathcal{P}'_1$ and $x'_2 \in \mathcal{P}'_2$ such that $x'_1(e) + \frac{1}{b} \cdot x'_2(e) \geq 1$ for all $e \in E$. This can be done by restricting x_1 to values of at most 1 and x_2 to values of at most b, and then selecting any bases that dominate these intermediate vectors. Now, we define $\overline{x}_1(e) = 1 - x'_1(e)$ and $\overline{x}_2(e) = b - x'_2(e)$ for all $e \in E$. By the construction of $\overline{\mathcal{P}}_1$ and $\overline{\mathcal{P}}_2$, this solution is feasible for I', that is, $\overline{x}_j \in \mathcal{B}(\overline{\mathcal{P}}_j)$ for $j \in \{1, 2\}$. We further have for every machine $e \in E$

$$\overline{x}_1(e) + \frac{1}{b} \cdot \overline{x}_2(e) = 2 - \left(x'_1(e) + \frac{1}{b} \cdot x'_2(e)\right) \leq 1 \,,$$

showing that $\mathrm{OPT}(I') \leq 1$.

Second, we prove the stated bound on the objective value of an approximate solution. Fix an $(2 - \frac{1}{\alpha})$-approximate solution for I' that selects bases $\overline{y}_1 \in \mathcal{B}(\overline{\mathcal{P}}_1)$ and $\overline{y}_2 \in \mathcal{B}(\overline{\mathcal{P}}_2)$. We construct a solution for I by defining $y'_1(e) = 1 - \overline{y}_1(e)$ and $y'_2(e) = b - \overline{y}_2(e)$ for every $e \in E$, meaning that $y'_j \in \mathcal{B}(\mathcal{P}'_j)$, and then choose an arbitrary basis $y_j \in \mathcal{B}(\mathcal{P}_j)$ that dominates y'_j, for $j \in \{1, 2\}$. We have for every player $e \in E$

$$y_1(e) + \frac{1}{b} \cdot y_2(e) \geq y'_1(e) + \frac{1}{b} \cdot y'_2(e) = 2 - \left(\overline{y}_1(e) + \frac{1}{b} \cdot \overline{y}_2(e)\right) \geq 2 - \left(2 - \frac{1}{\alpha}\right) \cdot \mathrm{OPT}(I') \geq \frac{1}{\alpha} \,.$$

Since $v_2 = \frac{1}{b}$ and $v_1 = 1$, we conclude $v_1 \cdot y_1(e) + v_2 \cdot y_2(e) \geq \frac{1}{\alpha}$ for every $e \in E$, which implies that the value of the constructed solution y_1 and y_2 is at least $\frac{1}{\alpha}$. □

3.5.2 Reduction of Matroid Makespan to Matroid Santa Claus

For the other direction, from matroid Makespan to matroid SantaClaus, we cannot make use of Lemma 3.20 to simplify the given instance whenever the two values satisfy $u < w < 1$. However, we can use a similar scaling trick as for the general two-value reduction in Section 3.3.2. Theorem 3.22 then follows from the following lemma applied to the binary search framework (Proposition 3.3).

Lemma 3.24. *Let $\alpha \geq 2$ and I be an instance of the restricted matroid Makespan problem with two jobs such that $\mathrm{OPT}(I) \leq 1$. Then we can compute an instance I' of the restricted matroid*

SantaClaus problem with two resources, such that, given an α-approximate solution for I', we can compute a solution for I with an objective value of at most $2 - \frac{1}{\alpha}$.

Proof. Given an instance I with $\mathrm{OPT}(I) \leq 1$ of the restricted matroid Makespan problem with machines E and two jobs with sizes $p_1, p_2 \geq 0$ and polymatroids $\mathcal{P}_1, \mathcal{P}_2$, we construct an instance I' of the restricted matroid SantaClaus problem as follows.

Let $k_1 = \lfloor 1/p_1 \rfloor$ and let $k_2 = \lfloor 1/p_2 \rfloor$. We first consider the polymatroids $\mathcal{P}'_1 = \{x \in \mathcal{P}_1 : x(e) \leq k_1 \ \forall e \in E\}$ and $\mathcal{P}'_2 = \{x \in \mathcal{P}_2 : x(e) \leq k_2 \ \forall e \in E\}$. Let f'_1 and f'_2 be the associated submodular functions of these polymatroids. Since $\mathrm{OPT}(I) \leq 1$, any optimal solution $x_j \in \mathcal{B}(P_j)$ satisfies $x_j(e) \leq k_j$ for all $e \in E$, and therefore, $x_j \in \mathcal{B}(P'_j)$, for $j \in \{1, 2\}$. Thus, $f_j(E) = x_j(E) = f'_j(E)$. Let $\overline{\mathcal{P}}_j$ be the dual polymatroid of $\mathcal{P}'_j$ with respect to the vector $k_j \cdot E$, for $j \in \{1, 2\}$. We compose instance I' using players E and two resources with polymatroids $\overline{\mathcal{P}}_1, \overline{\mathcal{P}}_2$ and resource values p_1, p_2.

Let $t = k_1 \cdot p_1 + k_2 \cdot p_2 - 1$. We show that $\mathrm{OPT}(I') \geq t$. Fix an optimal solution for I that selects bases $x_1 \in \mathcal{B}(\mathcal{P}_1)$ and $x_2 \in \mathcal{B}(\mathcal{P}_2)$. For each $j \in \{1, 2\}$, we define a vector $\overline{x}_j$ with $\overline{x}_j(e) = k_j - x_j(e)$ for all $e \in E$, and conclude that $\overline{x}_j \in \mathcal{B}(\overline{\mathcal{P}}_j)$, because $x_j \in \mathcal{B}(\mathcal{P}'_j)$. This means that $\overline{x}_1$ and $\overline{x}_2$ are a feasible solution for I'. Using $\mathrm{OPT}(I) \leq 1$, for every player $e \in E$ it holds that

$$p_1 \cdot \overline{x}_1(e) + p_2 \cdot \overline{x}_2(e) = (1 + t) - (p_1 \cdot x_1(e) + p_2 \cdot x_2(e)) = (1 + t) - \mathrm{OPT}(I) \geq t \,,$$

showing that $\mathrm{OPT}(I') \geq t$.

We finally prove the stated bound on the objective value of an approximate solution. Fix an α-approximate solution for I' that selects bases $\overline{y}_1 \in \overline{\mathcal{P}}_1$ and $\overline{y}_2 \in \overline{\mathcal{P}}_2$. We construct a solution $y_j \in \mathcal{B}(\mathcal{P}_j)$ for instance I by setting $y_j(e) = k_j - \overline{y}_j(e)$ for every $e \in E$ and $j \in \{1, 2\}$. The construction of the dual polymatroid $\overline{\mathcal{P}}_j$ implies $y_j \in \mathcal{B}(\mathcal{P}'_j)$ for $j \in \{1, 2\}$. We further have $y_j \in \mathcal{B}(\mathcal{P}_j)$, because $\mathcal{P}'_j \subseteq \mathcal{P}_j$ and $y_j(E) = f'_j(E) = f_j(E)$. Moreover, for every machine $e \in E$ it holds that

$$p_1 \cdot y_1(e) + p_2 \cdot y_2(e) = (1 + t) - \big(p_1 \cdot \overline{y}_1(e) + p_2 \cdot \overline{y}_2(e)\big) \leq (1 + t) - \frac{1}{\alpha} \cdot \mathrm{OPT}(I') \leq 1 + t - \frac{t}{\alpha}.$$

Since by construction $t \leq 1$, we have $t - t/\alpha \leq 1 - \frac{1}{\alpha}$, which implies that the makespan of the constructed solution y_1 and y_2 is at most $2 - \frac{1}{\alpha}$. □

3.6 Concluding Remarks

For the two notorious open problems in scheduling theory, we prove Makespan to be at least as difficult as SantaClaus; more precisely, a better-than-2 approximation for Makespan would imply a constant approximation for SantaClaus. In the two-value case, both problems appear equivalent with respect to approximability. The obvious open question is whether there is also a Makespan-to-SantaClaus reduction (for restricted or the general case). Here we note that for restricted Makespan, all efforts to refine the local search method in order to give a better-than-2 approximation have failed so far. Also with our new reduction techniques it seems that it would require additional ideas to handle this problem.

Finally, we comment on an alternative matroid scheduling variant with matroid constraints on the *items* allocated to a specific machine/player. In the *machine-matroid* Makespan problem, each machine would be given a matroid on the jobs. All jobs must be assigned such that each machine receives an independent set of its matroid. The *player-matroid* SantaClaus can be defined similarly.

Kawase et al. [Kaw+21] consider such matroid partition problems for various objective functions showing complexity results. Further, two special-matroid examples for Makespan have been studied, namely, bag-constrained scheduling [DW17; GJK19] (single partition matroid) and scheduling with capacity constraints [Che+16] (uniform matroids). The approximability lower bound $\Omega(\log^{1/4} n)$ by [DW17] holds for the restricted setting and even translates to an inapproximability bound for machine-matroid Makespan for identical machines with machine-dependent matroids. We are not aware of any similarly strong lower bounds for the SantaClaus variant.

Bibliographic Note

This chapter is based on joint work with Étienne Bamas, Nicole Megow, Lars Rohwedder, and Jens Schlöter [Bam+24]. Thus, some parts of this chapter are identical with [Bam+24].

Part II

Online Scheduling

Chapter 4

Clairvoyant Online Scheduling

4.1 Introduction

In this chapter, we introduce and analyze two clairvoyant online algorithms for minimizing the total weighted completion time on unrelated machines. This means that jobs arrive online over time and, whenever a job arrives, all of its characteristics become known to the algorithm. Both algorithms preempt jobs, one computes a migratory schedule, and the other one a non-migratory schedule.

The results of this section are motivated by the PreemptiveWSPT rule. On a single machine, this algorithm schedules the available job j with the highest density w_j/p_j at any time. In particular, if job k is released while another job j is running, the rule preempts j and processes k if $w_k/p_k > w_j/p_j$. Schulz and Skutella, as well as Goemans, Wein, and Williamson, proved that PreemptiveWSPT has a competitive ratio equal to 2 for $1 \mid r_j, \text{pmtn} \mid \sum w_j C_j$ [SS02a]. Megow and Schulz [MS04] showed that on parallel identical machines, a natural generalization, which schedules at any time the at most m available jobs with highest densities, also has a competitive ratio equal to 2. Moreover, Goemans [Goe96] proved that PreemptiveWSPT minimizes the total weighted mean busy time on a single machine.

4.1.1 Our Results

We present and analyze two generalizations of the PreemptiveWSPT rule for online scheduling on unrelated machines.

- The first algorithm is based on the MinIncrease paradigm to assign arriving jobs to machines, where it then schedules assigned jobs by PreemptiveWSPT. In particular, it computes a non-migratory schedule. In Section 4.3, we prove that it has a competitive ratio of at most 5.83 for $R \mid r_j, \text{pmtn}, \text{non-mig} \mid \sum w_j C_j$.

- The second algorithm computes a migratory schedule. On a high-level, it computes at any time a set of at most m jobs and assigns them to machines so that the *sum* of their densities in this assignment is maximized. In Section 4.4, we formally define this algorithm and show that it is 7.24-competitive for $R \mid r_j, \text{pmtn} \mid \sum w_j C_j$.

Both analyses are based on dual fitting (cf. Section 2.4). Next, we provide an overview of state-of-the-art results and techniques used for clairvoyant online scheduling for total completion time objectives.

4.2 Related Work on Clairvoyant Online Scheduling

The arguably first analysis of a clairvoyant online algorithm for the total completion time objective is by Schrage [Sch68], who showed that the Shortest-Remaining-Processing-Time rule (SRPT), which schedules at any time the job with the smallest remaining processing time, minimizes the sum of flow times on a single machine, where the flow time of a job is defined as $C_j - r_j$. Thus, SRPT also computes an optimal solution for $1 \mid r_j, \text{pmtn} \mid \sum C_j$. Other early results are 2-competitive algorithms for $1 \mid r_j \mid \sum C_j$ via delaying jobs j to $\max\{r_j, p_j\}$ and $r_j + p_j$, and then scheduling them greedily via SPT (called DelayedSPT) by Hoogeveen and Vestjens [HV96] and Stougie (cited in [Ves97]), respectively. Anderson and Potts [AP04] extended these ideas to the weighted objective.

For scheduling on unrelated machines, two main techniques have been very successful. The first technique is the doubling framework by Hall et al. [Hal+97]. Roughly speaking, it divides time into geometric intervals and schedules in every interval a set of jobs of largest total weight to completion, while completely ignoring newly arriving jobs. Since computing such sets is in general NP-hard, we distinguish between whether bounds admit polynomial-time algorithms, unless P = NP. For $R \mid r_j \mid \sum w_j C_j$, Hall et al. [Hal+97] prove bounds of 4 and 8 (polynomial-time), which soon later have been improved by Chakrabarti et al. [Cha+96] to 2.886 and 5.771 (polynomial-time), respectively, via choosing the interval offset randomly. Rather recently, Bienkowski et al. [BKL21] gave an improved deterministic 3-competitive algorithm and a randomized 2.443-competitive algorithm, without polynomial-time restriction.

The second technique uses the MinIncrease paradigm, also known as greedy assignment. The main idea is to immediately assign a job to a machine when it arrives (*immediate dispatch*), and to schedule assigned jobs on a machine using some single machine policy. The assignment of a job is chosen in a way such that a certain cost function is minimized, which in its simplest form is the actual increase of the objective value when assigning the job to a machine. Specifically, a job is never reassigned, hence every algorithm using this framework computes a non-migratory schedule. This idea was arguably first used in the design of algorithms that minimize the makespan for jobs that arrive one-by-one in a list and have to be immediately assigned. For m identical parallel machines, Graham [Gra66; Gra69] proved a competitive ratio of $2 - \frac{1}{m}$ when assigning the job to the currently least loaded machine, which indeed is an assignment that increases the current makespan the least. This idea was extended later to $\Theta(\log n)$-competitive algorithms for restricted assignment [ANR95] and general unrelated machines [Asp+97]. For min-sum objectives in the online-time model, around the same time, Avrahami and Azar [AA03; AA07] used MinIncrease to minimize the total flow time with speed augmentation and Megow et al. [MUV04; MUV05] to minimize the total weighted completion time in a stochastic setting, both on parallel identical machines. Chekuri et al. [Che+04] analyzed the algorithm of Avrahami and Azar [AA07] for more general objectives. Later, Chadha et al. [Cha+09] extended it and gave the first constant competitive algorithm for the total weighted flow time objective on unrelated machines with speed augmentation, which subsequently was again improved and generalized [AGK12; GKP10; IM11]. For minimizing the total weighted completion time on unrelated machines, Gupta et al. [Gup+21] showed that a combination of MinIncrease and DelayedWSPT has a competitive ratio of at most 7.216 for $R \mid r_j \mid \sum w_j C_j$, and Jäger [Jäg21] proved that MinIncrease with WSRPT machine policy is 4-competitive when

Table 4.1: Current bounds on the competitive ratio of clairvoyant online problems subject to deterministic polynomial-time algorithms.

Problem	Lower bound	Upper bound
$1 \mid r_j, \text{pmtn} \mid \sum C_j$	1	1 [Sch68]
$1 \mid r_j, \text{pmtn} \mid \sum w_j C_j$	1.104 [JS23]	1.226 [JS23]
$1 \mid r_j \mid \sum w_j C_j$	2 [HV96]	2 [AP04; HV96]
$P \mid r_j, \text{pmtn} \mid \sum C_j$	1.105 [XC12]	1.25 [Sit10]
$P \mid r_j, \text{pmtn} \mid \sum w_j C_j$	1.114 [XC12]	$1.791 + o(1)$ [Sit10], 2 [MS04]
$P \mid r_j \mid \sum C_j$	1.309 [Ves97]	1.546[a] [Sch22]
$P \mid r_j \mid \sum w_j C_j$	1.309	$1.791 + o(1)$ [Sit10], 2.618 [CW09]
$R \mid r_j, \text{pmtn}, \text{non-mig} \mid \sum w_j C_j$	1.114	4 [Jäg21], 5.83 (Thm. 4.1)
$R \mid r_j, \text{pmtn} \mid \sum w_j C_j$	1.114	4.62 (Thm. 5.27), 7.24 (Thm. 4.5)
$R \mid r_j \mid \sum w_j C_j$	1.309	7.216 [Gup+21]

[a] This bound only holds if the number of machines is even.

allowing preemption but not migration. The WSRPT machine policy schedules at any time t the assigned job j with largest weight to remaining processing time ratio on that machine, that is, $w_j / p_{ij}(t)$. This latter result is tight, because it is known that MinIncrease with WSPT machine policy has a competitive ratio of at least 4 for uniform release dates [CQ12; Gup+21].

Moreover, there is a 2-competitive algorithm for a special case of related machines for that the ordered speed vector $s_1 \geq \cdots \geq s_m$ does not decrease too quickly [Liu+09].

We give an overview of state-of-the-art competitive ratios in Table 4.1.

4.3 Non-Migratory Preemptive Online Scheduling on Unrelated Machines

We study a variant of MinIncrease for the online problem $R \mid r_j, \text{pmtn}, \text{non-mig} \mid \sum w_j C_j$. Our algorithm uses the PreemptiveWSPT machine policy and assigns an arriving job to a machine that minimizes the increase in the current objective. Inspired by the analyses of previous MinIncrease variants [AGK12; Gup+21; Jäg21], we prove the following theorem.

Theorem 4.1. *The MinIncrease PreemptiveWSPT algorithm has a competitive ratio of at most* $3 + 2\sqrt{2} \approx 5.8284$ *for minimizing the total weighted completion time on unrelated machines with release dates, preemption, and without migration,* $R \mid r_j, \text{pmtn}, \text{non-mig} \mid \sum w_j C_j$.

As mentioned above, Jäger [Jäg21] showed that the MinIncrease WSRPT algorithm has a competitive ratio of 4. While the bound on the competitive ratio of our algorithm is worse, our machine policy has the advantage that it only requires to know for every machine i the fixed non-increasing order of the jobs' densities $(d_{ij})_j$.

The remainder of this section is dedicated to the proof of Theorem 4.1, which uses dual fitting. We first formalize the algorithm. Fix an instance and let $\kappa > 1$ be a real number that we will set later. We assume without loss of generality by scaling the instance that all processing requirements, speeds and release dates are integer multiples of κ.

Let $J_i(j)$ be the set of available jobs that are assigned to machine i at time r_j, excluding job j. As this definition is ambiguous if there are two jobs j and j' with $r_j = r_{j'}$ being assigned to i, we assume that we assign them in the order of their index. The increase of the objective value of the algorithm due to assigning job j to machine i at time r_j equals

$$Q(i,j) = w_j\left(r_j + p_{ij} + \sum_{\substack{j' \in J_i(j) \\ d_{ij'} \geq d_{ij}}} p_{ij'}(r_j)\right) + p_{ij} \sum_{\substack{j' \in J_i(j) \\ d_{ij'} < d_{ij}}} w_{j'} \,.$$

Thus, the algorithm assigns job j to machine $g(j) = \arg\min_i Q(i,j)$, where ties are broken arbitrarily. We now perform a dual fitting argument. As we have seen in Section 2.3, the LP formulation ($\mathrm{LP}_R^{\mathrm{nm}}$) is a relaxation of our problem. Its dual (without (2.6)) can be written as follows.

$$\begin{aligned} \max \quad & \sum_{j \in J} a_j - \sum_{i \in M} \sum_{t \geq 0} b_{it} && && (\mathrm{DLP}_{\mathrm{R}}^{\mathrm{nm}}) \\ \text{s.t.} \quad & \frac{a_j}{p_{ij}} \leq b_{it} + w_j \cdot \left(\frac{t + 1/2}{p_{ij}} + \frac{1}{2}\right) && \forall i \in M, \forall j \in J, \forall t \geq r_j && (4.1) \\ & a_j, b_{it} \geq 0 && \forall i \in M, \forall j \in J, \forall t \geq 0 \end{aligned}$$

Let C_j be the completion time of j in the algorithm's schedule and ALG be the algorithm's objective value. Let $U_i(t)$ be the set of unfinished jobs at time t that are assigned to machine i when they arrive, that is, $U_i(t) = \{j \in J \mid g(j) = i \wedge t < C_j\}$. Note that $U_i(t)$ includes unreleased jobs. We define the following assignment:

- $\bar{a}_j \coloneqq Q(g(j), j)$ for every job j and
- $\bar{b}_{it} \coloneqq \sum_{j \in U_i(\kappa \cdot t)} w_j$ for every machine i and time t.

The following two lemmas show that this assignment is feasible and that its dual objective value captures a fraction of the algorithm's objective value.

Lemma 4.2. *It holds that* $\sum_{j \in J} \bar{a}_j - \sum_{i \in M} \sum_{t \geq 0} \bar{b}_{it} = \left(1 - \frac{1}{\kappa}\right) \cdot \mathrm{ALG}$.

Proof. The definition of $Q(g(j), j)$ implies $\sum_{j \in J} \bar{a}_j = \sum_{j \in J} Q(g(j), j) = \mathrm{ALG}$. Since we assumed that all release dates, speeds and processing requirements are integer multiples of κ, all preemptions occur at integer multiples of κ and therefore also all job completions. Thus, $\sum_{t \geq 0} \sum_{j \in U_i(\kappa \cdot t)} w_j = \frac{1}{\kappa} \sum_{t \geq 0} \sum_{j \in U_i(t)} w_j$ for every machine i, and

$$\sum_{i \in M} \sum_{t \geq 0} \bar{b}_{it} = \sum_{i \in M} \sum_{t \geq 0} \sum_{j \in U_i(\kappa \cdot t)} w_j = \frac{1}{\kappa} \sum_{i \in M} \sum_{t \geq 0} \sum_{j \in U_i(t)} w_j = \frac{1}{\kappa} \cdot \mathrm{ALG} \,,$$

which implies the desired equality. □

Lemma 4.3. *The solution* $\left(\frac{1}{\kappa+1} \bar{a}_j\right)_j$ *and* $\left(\frac{1}{\kappa+1} \bar{b}_{it}\right)_{i,t}$ *is feasible for* ($\mathrm{DLP}_{\mathrm{R}}^{\mathrm{nm}}$).

Proof. Since our defined variables are non-negative by definition, it suffices to show that this assignment satisfies (4.1). Fix a job j, a machine i and a time $t \geq r_j$. We assume that no new job

arrives after j, since such a job may only increase $\bar{b}_{it}$ while $\bar{a}_j$ stays unchanged. Let $j_1, \dots, j_z$ be the jobs of $J_i(j)$ indexed in WSPT order by densities $d_{ij} = w_j/p_{ij}$. Defining

- $H \coloneqq \{j' \in J_i(j) : d_{ij'} \geq d_{ij}\} = \{j_1, \dots, j_r\}$ and
- $L \coloneqq \{j' \in J_i(j) : d_{ij'} < d_{ij}\} = \{j_{r+1}, \dots, j_z\}$,

and using $\bar{a}_j = Q(g(j), j) \leq Q(i, j)$ and $\kappa + 1 > 2$ yields

$$\frac{a_j}{p_{ij}} = \frac{\bar{a}_j}{(\kappa+1)p_{ij}} \leq \frac{d_{ij}}{\kappa+1}\left(r_j + \sum_{j' \in H} p_{ij'}(r_j)\right) + \frac{w_j}{2} + \sum_{j' \in L} \frac{w_{j'}}{\kappa+1} .$$

Thus, asserting (4.1) reduces to proving

$$\frac{d_{ij}}{\kappa+1}\left(r_j + \sum_{j' \in H} p_{ij'}(r_j)\right) + \sum_{j' \in L} \frac{w_{j'}}{\kappa+1} \leq d_{ij} \cdot t + b_{it} . \tag{4.2}$$

Observe that the total processing time of all jobs in $J_i(j)$ that are completed before time $\kappa \cdot t$ is at most $\kappa \cdot t$. Further, $r_j + \kappa \cdot t \leq (\kappa+1)t$. Now consider the case that machine i processes a job j_k at time $\kappa \cdot t$. If $j_k \in H$, using $d_{ij} \leq w_{j_\ell}/p_{ij_\ell} \leq w_{j_\ell}/p_{ij_\ell}(r_j)$ for all $j_\ell \in H$ gives

$$\begin{aligned}
&\frac{d_{ij}}{\kappa+1}\left(r_j + \sum_{\ell=1}^{k-1} p_{ij_\ell}(r_j)\right) + \frac{d_{ij}}{\kappa+1}\sum_{\ell=k}^{r} p_{ij_\ell}(r_j) + \sum_{j' \in L} \frac{w_{j'}}{\kappa+1} \\
&\leq d_{ij} \cdot t + \frac{1}{\kappa+1}\sum_{\ell=k}^{r} w_{j_\ell} + \sum_{j' \in L} \frac{w_{j'}}{\kappa+1} \leq d_{ij} \cdot t + \frac{\bar{b}_{it}}{\kappa+1} = d_{ij} \cdot t + b_{it} .
\end{aligned}$$

The last inequality holds since all jobs in $J_i(j)$ that are processed after job j_{k-1} are unfinished at time $\kappa \cdot t$ and assigned to i in the algorithm's schedule, hence part of $U_i(\kappa \cdot t)$. If $j_k \in L$, using $w_{j_\ell} < d_{ij} \cdot p_{ij_\ell}$ for all $j_\ell \in L$ implies

$$\begin{aligned}
&\frac{d_{ij}}{\kappa+1}\left(r_j + \sum_{\ell=1}^{r} p_{ij_\ell}(r_j)\right) + \frac{1}{\kappa+1}\sum_{\ell=r+1}^{k-1} w_{j_\ell} + \frac{1}{\kappa+1}\sum_{\ell=k}^{z} w_{j_\ell} \\
&\leq \frac{d_{ij}}{\kappa+1}\left(r_j + \sum_{\ell=1}^{r} p_{ij_\ell}(r_j) + \sum_{\ell=r+1}^{k-1} p_{ij_\ell}\right) + \frac{1}{\kappa+1}\sum_{\ell=k}^{z} w_{j_\ell} \leq d_{ij} \cdot t + \frac{1}{\kappa+1}\sum_{\ell=k}^{z} w_{j_\ell} \leq d_{ij} \cdot t + b_{it} .
\end{aligned}$$

If no job is running at time $\kappa \cdot t$, we conclude that all jobs in $J_i(j)$ must already be completed, because the algorithm does not idle unnecessarily, and we assumed that no job is released after j. By using $w_{j_\ell} < d_{ij} \cdot p_{ij_\ell}$ for all $j_\ell \in L$ we have

$$\frac{d_{ij}}{\kappa+1}\left(r_j + \sum_{\ell=1}^{r} p_{ij_\ell}(r_j)\right) + \sum_{\ell=r+1}^{z} w_{j_\ell} \leq \frac{d_{ij}}{\kappa+1}\left(r_j + \sum_{\ell=1}^{r} p_{ij_\ell}(r_j) + \sum_{\ell=r+1}^{z} p_{ij_\ell}\right) \leq d_{ij} \cdot t .$$

Since we established (4.2) in every case, we conclude the lemma. □

We finally prove Theorem 4.1. Weak duality and Lemma 4.3 imply that the objective value of ($\mathrm{DLP}_{\mathrm{R}}^{\mathrm{nm}}$) of our assignment is a lower bound on the optimal objective value. Lemma 4.2 gives

$$\mathrm{OPT} \geq \sum_{j \in J} a_j - \sum_{i \in M} \sum_{t \geq 0} b_{it} = \frac{1}{\kappa + 1} \left(\sum_{j \in J} \bar{a}_j - \sum_{i \in M} \sum_{t \geq 0} \bar{b}_{it} \right) = \left(\frac{1 - 1/\kappa}{\kappa + 1} \right) \cdot \mathrm{ALG} \,.$$

We conclude that the algorithm has a competitive ratio of at most $3 + 2\sqrt{2} \approx 5.8284$ by choosing $\kappa := 1 + \sqrt{2}$.

4.4 Migratory Preemptive Online Scheduling on Unrelated Machines

In this section, we move to the migratory setting. We study a simple and efficient online algorithm for the problem $R \,|\, r_j, \mathrm{pmtn} \,|\, \sum w_j C_j$.

We first review the best-known deterministic polynomial-time algorithm for this problem, which is the *doubling framework* by Hall et al. [Hal+97]. It achieves a competitive ratio of at most 8. While they only state this bound for the non-preemptive problem, we quickly argue that it extends also to the preemptive case; this could be folklore. On a high level, their algorithm divides time into geometric intervals, and schedules in every interval $[2^\ell, 2^{\ell+1}]$ the set S_ℓ of jobs of maximum weight that are released before time 2^ℓ and can be feasibly scheduled non-preemptively to completion within that interval. This ensures that the total weight of jobs completed by the algorithm by time $2^{\ell+1}$ is at least the total weight of all jobs completed by an optimal non-preemptive schedule by time 2^ℓ. This implies that, at any time, the total weight of unfinished jobs in the algorithm's schedule is at most 4 times that of the optimal's schedule, yielding a competitive ratio of at most 4. This argument works analogously for preemptive schedules. Since computing such sets of maximum total weight is NP-hard, Hall et al. propose an approximation algorithm that finds a set S_ℓ of maximum weight that can be non-preemptively scheduled in the *doubled* interval $[2^{\ell+1}, 2^{\ell+2}]$. Thus, the bound on the competitive ratio also doubles to 8. This subroutine uses the approximation framework for the generalized assignment problem by Shmoys and Tardos [ST93], and thus requires solving a linear program. Since the approximation compares against a solution of maximum weight that allows a preemptive schedule, the subroutine can also be integrated in the framework for the preemptive setting. This argumentation can analogously be applied to the randomized variant of the doubling framework by Chakrabarti et al. [Cha+96].

Theorem 4.4 (Hall et al. [Hal+97], Chakrabarti et al. [Cha+96])**.** *The doubling framework yields a deterministic polynomial-time online algorithm with a competitive ratio of at most* 8*, and a randomized polynomial-time online algorithm with a competitive ratio of at most* 5.78*, for minimizing the total weighted completion time on unrelated machines with release dates, preemption, and migration,* $R \,|\, r_j, pmtn \,|\, \sum w_j C_j$.

While the doubling framework can be seen as applying the WSPT rule in every geometric interval, we propose an algorithm that resembles the PreemptiveWSPT rule more directly.

Our algorithm computes at any time an assignment of at most m jobs to machines that maximizes the total density, that is, the sum of the densities for this job-to-machine assignment.

This can be done efficiently by computing, at any time t, a maximum-weight matching A_t between alive jobs $j \in J(t) = \{j \in J \mid r_j \leq t \leq C_j\}$ and machines $i \in M$ with edge weights $d_{ij} := w_j s_{ij}/p_j$. Note that we need to recompute the matching only if the set $J(t)$ changes, that is, when a job arrives or completes. Thus, the algorithms makes only a polynomial number of migrations and preemptions and therefore runs in polynomial time. As this algorithm essentially follows the PreemptiveWSPT paradigm, we do not give it another name. Our main result is the following theorem.

Theorem 4.5. *PreemptiveWSPT has a competitive ratio of at most 7.24 for minimizing the total weighted completion time on unrelated machines with release dates, preemption, and migration,* $R \mid r_j, pmtn \mid \sum w_j C_j$.

The remainder of this section is dedicated to the proof of Theorem 4.5. Fix an instance and a schedule of the algorithm for this instance. Let C_j be the completion time of j in the algorithm's schedule and ALG be the algorithm's objective value. Given rational input, we assume by scaling that all processing requirements and speeds are integers. Thus, the schedule of PreemptiveWSPT only preempts and completes jobs at integer times.

We perform a dual fitting argument. Since we allow migration, ($\mathrm{LP}_R^{\mathrm{nm}}$) is not a relaxation of our problem (cf. Lemma 2.8). However, Theorem 2.9 gives that the optimal objective value of ($\mathrm{LP}_R^{\mathrm{nm}}$) is at most 1.81 times the objective value OPT of an optimal solution. Moreover, this statement holds even for the following LP relaxation with a slightly stronger objective function (cf. [Sit17]). We state this relaxation in the setting where every machine is slowed down by a factor of $\frac{1}{\kappa}$ for a constant $\kappa \geq 1$. Thus, by the scalability of completion times, the optimal objective value of the following LP is at most $1.81\kappa \cdot \mathrm{OPT}$.

$$\begin{aligned}
\min \quad & \sum_{j \in J} w_j \sum_{i \in M} \sum_{t \geq r_j} \left(t + \frac{1}{2}\right) \frac{x_{ijt} s_{ij}}{p_j} + x_{ijt} && && (\widetilde{\mathrm{LP}}_R^{\mathrm{nm}}(\kappa)) \\
\text{s.t.} \quad & \sum_{i \in M} \sum_{t \geq r_j} x_{ijt} s_{ij} \geq p_j && \forall j \in J \\
& \sum_{j \in J} x_{ijt} \leq \frac{1}{\kappa} && \forall i \in M, \forall t \geq 0 \\
& x_{ijt} \geq 0 && \forall i \in M, \forall j \in J, \forall t \geq 0
\end{aligned}$$

The dual of ($\widetilde{\mathrm{LP}}_R^{\mathrm{nm}}(\kappa)$) can be written as follows.

$$\begin{aligned}
\max \quad & \sum_{j \in J} a_j - \sum_{i \in M} \sum_{t \geq 0} b_{it} && && (\widetilde{\mathrm{DLP}}_R^{\mathrm{nm}}(\kappa)) \\
\text{s.t.} \quad & \frac{a_j s_{ij}}{p_j} - \frac{w_j s_{ij}}{p_j} t \leq \kappa \cdot b_{it} + w_j && \forall i \in M, \forall j \in J, \forall t \geq r_j \\
& a_j, b_{it} \geq 0 && \forall i \in M, \forall j \in J, \forall t \geq 0
\end{aligned}$$

Before presenting our dual setup, we first define for every machine i and any time t

$$\beta_{it} := \begin{cases} d_{ij} & \text{if machine } i \text{ is matched to job } j \in J(t) \text{ in } A_t \\ 0 & \text{otherwise,} \end{cases}$$

and for every job j and any time t

$$\gamma_{jt} \coloneqq \begin{cases} d_{ij} & \text{if job } j \text{ is matched to machine } i \in M \text{ in } A_t \\ 0 & \text{otherwise.} \end{cases}$$

Now, we define the following assignment of dual variables:

- $\bar{a}_j \coloneqq w_j C_j$ for every job j,
- $\bar{b}_{it} \coloneqq \frac{1}{2} \sum_{t' \geq t} \beta_{it'}$ for every machine i and time t.

For a job j and a time t, let $\sigma_j(t)$ be the machine assignment of j at time t according to A_t. If j is not assigned to any machine at time t, we set $\sigma_j(t) \coloneqq \perp$, and further define $s_{\perp,j} \coloneqq 0$. Since no job receives more processing than required, we have the following observation.

Observation 4.6. *For every j and time t, it holds that $\sum_{t'=t}^{C_j} s_{\sigma_j(t'),j} \frac{w_j}{p_j} \leq w_j$.*

The following two lemmas show that this assignment is feasible and that its dual objective value captures a fraction of the algorithm's objective value.

Lemma 4.7. *It holds that $\frac{1}{2}\mathrm{ALG} \leq \sum_{j \in J} \bar{a}_j - \sum_{i \in M} \sum_{t \geq 0} \bar{b}_{it}$.*

Proof. Clearly, $\sum_{j \in J} \bar{a}_j = \mathrm{ALG}$. Moreover, Observation 4.6 implies

$$\sum_{i \in M} \bar{b}_{it} = \frac{1}{2} \sum_{j \in U(t)} \sum_{t'=t}^{C_j} s_{\sigma_j(t'),j} \frac{w_j}{p_j} \leq \frac{1}{2} W(t) \,,$$

which gives $\sum_{i \in M} \sum_{t \geq 0} \bar{b}_{it} \leq \frac{1}{2}\mathrm{ALG}$, and thus concludes the proof. □

Lemma 4.8. *The solution $(\bar{a}_j)_j$ and $(\bar{b}_j)_j$ is feasible for $(\widetilde{\mathrm{DLP}}_R^{\mathrm{nm}}(\kappa))$ if $\kappa = 2$.*

Proof. First note that the dual assignment is non-negative. To verify the dual constraint, fix a machine i, a job j and a time $t \geq r_j$. The definition of $\bar{a}_j$ yields $\bar{a}_j \frac{s_{ij}}{p_j} - w_j t \frac{s_{ij}}{p_j} \leq \sum_{t'=t}^{C_j} \frac{w_j s_{ij}}{p_j} = \sum_{t'=t}^{C_j} d_{ij}$. Since $\sum_{t'=t}^{C_j} \gamma_{jt'} \leq w_j$ by Observation 4.6, it remains to validate $d_{ij} \leq \beta_{it'} + \gamma_{jt'}$ for every $t \leq t' \leq C_j$. We distinguish five cases:

(i) If $(i,j) \in A_{t'}$, then $d_{ij} = \beta_{it'} = \gamma_{jt'}$.

(ii) If $(i,j') \in A_{t'}$ and $(i',j) \in A_{t'}$ s.t. $i' \neq i$ (and thus $j' \neq j$), we know by the optimality of $A_{t'}$ that $d_{ij} \leq d_{ij} + d_{i'j'} \leq d_{i'j} + d_{ij'} = \gamma_{jt'} + \beta_{it'}$.

(iii) If $(i',j) \in A_{t'}$ and i is not matched in $A_{t'}$, we conclude $d_{ij} \leq d_{i'j} = \gamma_{jt'}$.

(iv) If $(i,j') \in A_{t'}$ and j is not matched in $A_{t'}$, we conclude $d_{ij} \leq d_{ij'} = \beta_{it'}$.

(v) The case where $s_{ij} > 0, w_j > 0$, but both i and j are unmatched in $A_{t'}$ contradicts the optimality of $A_{t'}$, as $t' \leq C_j$. Otherwise, $d_{ij} = 0$, and the inequality holds since the right side is non-negative.

This concludes the proof of the lemma. □

We finally prove Theorem 4.5. Weak duality, Lemma 4.7, and Lemma 4.8 imply

$$1.81 \cdot 2 \cdot \mathrm{OPT} \geq \sum_{j \in J} \bar{a}_j - \sum_{i \in M} \sum_{t \geq 0} \bar{b}_{it} \geq \frac{1}{2} \cdot \mathrm{ALG}\,,$$

which concludes that PreemptiveWSPT has a competitive ratio of at most 7.24.

4.5 Concluding Remarks

In this section, we presented two natural extensions of the well-known PreemptiveWSPT single-machine scheduling rule to unrelated machines. We showed that these have a competitive ratio of at most 5.83 and 7.24 for preemptively minimizing the total weighted completion time on unrelated machines with online job arrival, without and with migration, respectively.

For online clairvoyant $R \mid r_j, \mathrm{pmtn}, \text{non-mig} \mid \sum w_j C_j$, there exists a 4-competitive algorithm [Jäg21], and it remains open whether a better competitive ratio among polynomial-time algorithms is possible. For online clairvoyant $R \mid r_j, \mathrm{pmtn} \mid \sum w_j C_j$, the best-known polynomial-time algorithm has a competitive ratio of at most 5.78 but requires randomization [Cha+96]. In the next chapter, we give a 4.62-competitive polynomial-time online algorithm for this problem, which is even non-clairvoyant. Thus, we believe that it should be possible to prove a better competitive ratio than 4.62 for a polynomial-time online algorithm by exploiting clairvoyance.

If the polynomial-time constraint is lifted, a deterministic 3-competitive and randomized 2.443-competitive online algorithm is possible for $R \mid r_j \mid \sum w_j C_j$ [BL19], which also transfer to both, $R \mid r_j, \mathrm{pmtn}, \text{non-mig} \mid \sum w_j C_j$ and $R \mid r_j, \mathrm{pmtn} \mid \sum w_j C_j$ (similar to Theorem 4.4).

Bibliographic Note

This chapter is based on joint work with Nicole Megow and Martin Rapp [LM22; LMR23]. Specifically, Theorem 4.1 and its proof appear very similarly in [LM22]. The generalization of PreemptiveWSPT presented in Section 4.4 has been introduced in [LMR23], but there it is only proved that the algorithm is 8-competitive. Thus, some parts of this chapter are identical with [LM22; LMR23].

Chapter 5

Non-Clairvoyant Online Scheduling

5.1 Introduction

In this chapter, we study online algorithms that cope with both uncertainties, *online job arrival* and *non-clairvoyance*, while aiming to minimize the total weighted completion time. Recall that a non-clairvoyant online algorithm has no knowledge about a job's processing requirement until it completes.

To capture a variety of *preemptive* scheduling environments, we adopt the abstract perspective of *instantaneous resource allocation*. Specifically, at any time t, an algorithm can process a job j at a *rate* $y_j(t)$ such that the *rate vector* $y(t)$ over all n jobs satisfies polyhedral constraints given by a packing polytope $\mathcal{P} = \{y \in \mathbb{R}^n_{\geq 0} \mid B \cdot y \leq 1\}$, where $B \in \mathbb{Q}^{D \times n}_{\geq 0}$. A job is considered complete when it has received the total required processing. This abstract problem, termed *Polytope Scheduling Problem (PSP)*, was introduced in a seminal paper by Im et al. [IKM18]. It generalizes many well-studied preemptive scheduling problems, such as unrelated machine scheduling, multidimensional scheduling [Gho+11], or broadcast scheduling [BCS08]. The power of the general PSP formulation is that it abstracts away the specific details of the scheduling environment and extracts the essence of "packing over time".

We consider a simple and well-known mechanism, known as *Proportional Fairness* (PF), which dates back to Nash [Nas50] and is widely studied in the context of fair allocation and markets [JV10; Jal+23; Mou03; PTV21]. At any time t it allocates processing rates $y_j(t)$ to the available jobs j such that the *Weighted Nash Social Welfare* [KN79] is maximized, where the rates are interpreted as utilities. This has desirable properties from the perspective of *fairness*, such as Pareto-efficiency and envy-freeness [IKM18; TV24; Var76]. Im et al. [IKM18] were the first who studied this allocation rule in the context of scheduling. They proved that PF is a 128[1]-competitive non-clairvoyant algorithm for PSP, while the best-known non-clairvoyant lower bound for this problem is only 2. Interestingly, this lower bound already holds in the simple case of single-machine scheduling, a classical result by Motwani et al. [MPT94]. It is not hard to see that Round-Robin, the classical non-clairvoyant algorithm that splits a single machine evenly among all available jobs [MPT94], corresponds to PF for the unweighted single-machine problem. This algorithm achieves the optimal competitive ratio of 2 for $1 \mid \text{pmtn} \mid \sum C_j$ [MPT94].

A fundamental PSP is scheduling on unrelated machines with preemption and migration, $R \mid r_j, \text{pmtn} \mid \sum w_j C_j$. At any time a job can be assigned to at most one machine and on every machine there can be at most one job assigned. These matching constraints determine

[1]The factor stated in [IKM18] is 64, but the proof contained an algebraic mistake and is fixed by losing a factor 2 [IKM].

the polytope for the this PSP. Any fractional rate vector in this polytope can be efficiently transformed to a preemptive and migratory schedule for one time unit by the Birkhoff–von Neumann Theorem. We emphasize that both preemption [MPT94] and migration are necessary for competitive non-clairvoyant algorithms for this problem; we provide a general lower bound for non-migratory algorithms (cf. Theorem 5.3).

Little is known about non-clairvoyantly scheduling jobs with online arrival on unrelated machines with preemption and migration. The bound of 128 on the competitive ratio of PF [IKM18] holds for this problem. We can also prove that an algorithm designed for minimizing the total weighted flow time [Im+14] is 32-competitive for our total weighted completion time objective. There remains a substantial gap to the lower bound of 2, and to the best of our knowledge, even for the simpler cases of restricted assignment and related machines no substantially better constants are known.

In the *clairvoyant* setting, where a job's processing requirement becomes known at the job's release date, better constants are known; we refer to Section 4.2 for an overview of current bounds.

This chapter uses tools and concepts from *convex optimization*. For a general introduction to convex optimization, we refer to the book by Boyd and Vandenberghe [BV14], and for a modern overview of solution algorithms for convex programs to the book by Vishnoi [Vis21].

5.1.1 Our Results

For the general polytope scheduling problem (PSP), we simplify and substantially improve the original analysis of the non-clairvoyant Proportional Fairness algorithm (PF) by Im et al. [IKM18]. By exploiting the monotonicity of unfinished weight more directly and by optimizing scaling constants, we reduce the bound on the competitive ratio from 128 down to 27 (cf. Section 5.3).

Our main technical contribution is to provide new techniques that lead to further improvements for major special cases of PSP. On a high level, these fall into two categories: (i) we show and exploit a monotonicity property of PF and (ii) we prove and utilize a superadditivity property of the objective function value of an optimal offline solution. A more detailed overview of our techniques is provided in the following subsection; here we state and discuss the results obtained through each of them. Tables 5.1 and 5.2 summarize our non-clairvoyant scheduling results distinguished by online and and simultaneous job arrival, respectively.

Results via Monotonicity. Our first results concern PSP variants for which PF computes *monotone rates*, a subclass introduced by Im et al. [IKM18]. Informally speaking, in MonPSP the completion of a job does not harm any other job's processing, as it does not cause a decrease of that job's rate.

Definition 5.1 (PF-monotone PSP [IKM18])**.** Given job sets $J' \subseteq J$ with corresponding rates y' and y computed by PF, we say the PSP is *PF-monotone* (short MonPSP) if for every $j' \in J'$ it holds that $y'_{j'} \geq y_{j'}$.

We exploit this structural property rigorously to prove that PF has a competitive ratio of at most 4 for MonPSP (cf. Section 5.4). This substantially reduces the previously known bound

Table 5.1: Current bounds on the competitive ratio of online min-sum scheduling problems subject to polynomial-time algorithms.

Problem	old (nclv.)	old (clv.)	our upper bound
PSP	$128^{\ddagger}$ [IKM18]	128	$27^{\ddagger}$
MonPSP	$25.74^{*\ddagger}$ [IKM18]	25.74	$4^{\ddagger}$
$R \mid r_j, \text{pmtn} \mid \sum w_j C_j$	32^{*} [Im+14]	$5.78^{\dagger}$ [Cha+96]	4.62
$R \mid r_j, \text{pmtn}, s_{ij} \in \{0,1\} \mid \sum C_j$	25.74	$5.78^{\dagger}$	3
$Q \mid r_j, \text{pmtn} \mid \sum w_j C_j$	25.74	$5.78^{\dagger}$	4
$Q \mid r_j, \text{pmtn} \mid \sum C_j$	25.74	$5.78^{\dagger}$	3

* These bounds are implications of flow time results.
† Uses randomization. Besides the results of this section, the best known deterministic bound is 7.24 (cf. Theorem 4.5).
‡ The time complexity of PF depends on the encoding of the polytope.

of 25.74 [IKM18]. Related machine scheduling as well as multidimensional scheduling with certain utility functions are known to be special cases of MonPSP [IKM18]. We further show that the restricted assignment problem is also a MonPSP. Moreover, we argue that for restricted assignment as well as related machine scheduling, PF runs in strongly polynomial time.

It seems tempting to hope that MonPSP captures also general unrelated machine scheduling. However, as was suggested in [IKM15b], this is false; we give a counterexample in Section 5.4.3.

Results via Superadditivity. For our second group of results, we introduce a subclass of PSP, which is, in contrast to PF-monotonicity, not defined in terms of PF. For a fixed instance, the defining property makes assumptions on the optimal objective value when all jobs are released simultaneously as a function of the processing requirements $p = (p_1, \dots, p_n)$, denoted by $\mathrm{OPT}_0(p)$.

Definition 5.2 (α-superadditive PSP)**.** We say that a PSP is *α-superadditive* if for every partition $p = \sum_{\ell=1}^{L} p^{(\ell)}$ it holds that $\sum_{\ell=1}^{L} \mathrm{OPT}_0(p^{(\ell)}) \leq \alpha \cdot \mathrm{OPT}_0(p)$.

Based on this definition, we introduce a new analysis framework that builds on a decomposition of PF's schedule into structured instances on that we exploit α-superadditivity. We show that PF has a competitive ratio of at most $2\alpha + 1$ for α-superadditive PSP. For uniform release dates, this bound reduces to 2α.

We now outline important implications of this result. First, by showing that unrelated machine scheduling is 1.81-superadditive, we conclude that PF is a 4.62-competitive algorithm for $R \mid r_j, \text{pmtn} \mid \sum w_j C_j$. This improves upon the best known competitive ratios for clairvoyant polynomial-time algorithms, which are 7.24 (Theorem 4.5) for deterministic algorithms and 5.78 [Cha+96] for randomized algorithms. Remarkably, this is not only the first polynomial-time improvement within nearly thirty years, but it is even obtained by a deterministic non-clairvoyant algorithm. Second, we show that minimizing the total completion time is 1-superadditive both on related machines and for restricted assignment. This implies that PF achieves the best-possible competitive ratio of 2 for these problems. To the best of our

Table 5.2: Current bounds on the competitive ratio of non-clairvoyant min-sum scheduling problems.

Problem	old upper bound	lower bound	our upper bound
$R \mid \text{pmtn} \mid \sum w_j C_j$	32 [Im+14]	2	3.62
$R \mid \text{pmtn}, s_{ij} \in \{0,1\} \mid \sum C_j$	25.74 [IKM18]	2	2
$Q \mid \text{pmtn} \mid \sum C_j$	25.74 [IKM18]	2	2
$P \mid \text{pmtn} \mid \sum w_j C_j$	2 [Bea+12]	2	2
$P \mid \text{pmtn} \mid \sum C_j$	2 [MPT94]	2	2
$1 \mid \text{pmtn} \mid \sum w_j C_j$	2 [KC03]	2	2
$1 \mid \text{pmtn} \mid \sum C_j$	2 [MPT94]	2 [MPT94]	2

knowledge, this is the first tight analysis of a non-clairvoyant algorithm on heterogeneous machines for this objective. There results are presented in Section 5.5.

We note that PF cannot be 2-competitive for non-uniform release dates. In Section 5.8, we give an example showing that its competitive ratio is at least 2.19, even on a single machine.

Implications for Matching Markets. As a byproduct of our argumentation, we give new insights on one-sided matching markets with dichotomous utilities. For an overview of Fisher markets, matching markets, and related concepts of economics and algorithmic game theory, we refer to [Nis+07; VY21]. Recently, it has been shown that market equilibria can be computed in strongly polynomial time [GTV22; VY21] and, in the case of unit budgets, correspond to optimal solutions of the Nash Social Welfare maximization problem [GTV22]. By noting that the market can be formulated as a submodular utility allocation market, we show that both results are also directly implied by the work of Jain and Vazirani [JV10], even for arbitrary budgets (cf. Section 5.4.2).

5.1.2 Techniques and Intuition

The Proportional Fairness (PF) algorithm repeatedly solves a convex program, and we use its Lagrange multipliers and its KKT conditions to characterize the optimal solutions. Further, we use a dual fitting analysis (cf. Section 2.4) to compare PF to an optimal schedule. It is straightforward to incorporate the instantaneous polyhedral constraints of PSP into a standard time-indexed LP relaxation (similar to (LP_R)) that describes the scheduling problem with the weighted mean busy time objective instead of the total weighted completion time [DW90; IKM18; SS02b]. To prove a performance guarantee, we construct a feasible dual solution whose objective value captures a fraction of the algorithm's objective value.

Challenges in the Dual Fitting Approach. The dual of the standard LP of PSP has two sets of variables: $(a_j)_j$ that correspond to the primal constraints that each job has to be completed, and $(b_{dt})_{d,t}$ that correspond to the primal instantaneous polyhedral constraints (we call them dual packing variables in the following). A natural interpretation in the case of a *single machine* can be derived by wiggling the primal constraints [AGK12]: a_j is proportional

to job j's contribution to the total objective value, and the variable b_t corresponding to the only packing constraint is proportional to the total weight of available jobs at time t. By setting duals according to this interpretation based on PF's schedule, one can easily prove that PF is 4-competitive for single-machine scheduling.

Crafting duals for general PSP is substantially more complex, as pointed out by Im et al. [IKM18], because, at each time t, we must distribute the total weight of available jobs across multiple variables corresponding to the D packing constraints. A natural weight distribution is given by the optimal Lagrange multipliers $(\eta_{dt})_d$ corresponding to the packing constraints in the Eisenberg-Gale convex program [EG59], which PF uses to compute the Weighted Nash Social Welfare at time t. This natural dual setup, however, does not work in general because by the dynamics of PF it can happen that a job j needs more weight distributed to its relevant dual packing variables at some time $t' > t$ to compensate a smaller rate $y_j(t') < y_j(t)$; this seems necessary for arguing dual feasibility. Indeed, there are examples where a job's rate decreases when another job completes in PF's schedule (cf. Section 5.4.3). The analysis of Im et al. [IKM18] handles this by carefully redistributing optimal Lagrange multipliers across different times using more complex dual variable assignments; we present a simplified and improved variant thereof in Section 5.3. Our main contribution are the subsequent new methods.

Monotonicity. Our key observation for MonPSP is that these dynamics have much more structure. In particular, no job requires more weight to be assigned to its relevant dual packing variables at a later time. We show this using the KKT conditions of the Eisenberg-Gale convex program and the fact that a job's rate cannot decrease if another job completes. While the latter can happen when a new job j' arrives, j' then already contributes to the objective before its arrival, and we again use monotonicity to show that no job running before j''s arrival suffers from j''s absence. These observations admit the natural dual setup outlined above and prove that PF is 4-competitive for MonPSP (cf. Section 5.4).

Tight Dual Fitting and Structured Instances. Breaching the factor of 4 with either of these dual setups seems difficult; this barrier can also be found in similar dual fitting analyses [AGK12; Gar+19; Gup+21; IKM18; Im+14; Jäg23; Las+23; LM22]. To the best of our knowledge, better guarantees for non-clairvoyant algorithms have only been achieved in settings where other strong lower bounds are known [Bea+12; Den+00; JW24; KC03; MPT94], which appears challenging for MonPSP as well as for the general PSP.

If we knew the structural properties of an optimal LP solution, we could potentially overcome this barrier by tailoring the dual packing variables in a way that a job only contributes a fraction of its weight to them that is proportional to its progress in an optimal LP solution. Indeed, in Section 5.7, we give a tight dual fitting analysis for PF on a single machine, where we understand the optimal LP solution [Goe96] (cf. Theorem 2.6). For PSP, however, it seems hard to get sufficient insights on (near-)optimal LP solutions.

We instead pursue a different approach. Instead of analyzing the whole schedule with one dual fitting argument, we split PF's objective into several pieces and apply dual fitting to each of them. Our first key insight to make this work is that a special case of PSP admits a tight dual fitting. This comprises instances in which PF completes all jobs simultaneously; we call those *structured*. We prove that for structured instances PF yields an optimal solution to the LP,

minimizing the weighted mean busy time, and that this is 2-competitive for the total weighted completion time.

Decomposition and Superadditivity. The second key observation is that we can decompose PF's objective by splitting the PF schedule at release dates and completion times into structured instances. This does not work for any schedule and uses specific properties: (i) the PF schedule after time t corresponds to the PF schedule for the instance where the processing requirements of all jobs are reduced by the processed amount before time t in the original PF schedule, and (ii) PF assigns an available job a *positive* rate, hence its flow time can be expressed as the sum of its completion times in the structured subinstances in that it appears. Finally, we bound the overall optimal objective value by the optimal objective values of the subinstances via superadditivity.

The primary challenge in showing superadditivity for machine scheduling problems lies in handling migrations. At first sight one may think that allowing job migrations makes a problem easier, since an algorithm has more freedom to revert bad decisions, especially in online settings. We even show that any non-migratory non-clairvoyant algorithm has an unbounded competitive ratio (cf. Theorem 5.3). However, at the same time migrations allow creating more complex optimal schedules, which makes analyses way more difficult. Indeed, there are many indications in literature that scheduling unrelated machines with migrations is much harder than without: (i) if migration is allowed, the unweighted offline problem is APX-hard [Sit17], while otherwise it is polynomial-time solvable [BJS74; Hor73], (ii) strong time-indexed LP relaxations no longer lower bound the optimal objective value if migration is allowed (cf. Lemma 2.8), and (iii) best-known offline approximation ratios for the weighted problem are stronger if migration is disallowed [Har24; IL16; IL23; Li25; SS02b; Sit17; Sku01].

Our key trick for unrelated machines is to evade migrations by showing 1-superadditivity for non-preemptive schedules, which are more structured and thus easier to handle. To go back to the optimal migratory objective value, we use the concept of the *power of preemption*, that is, the factor between an optimal non-preemptive objective value and an optimal migratory objective value. For unrelated machines, the currently best-known upper bound by Sitters [Sit17] on this ratio is 1.81 (cf. Corollary 2.11).

The analysis via non-preemptive schedules always loses the power of preemption as a factor, hence rules out tight competitiveness bounds of 2 whenever this factor is larger than 1 because this ratio can already be tight when comparing to the optimal non-preemptive schedule. To still achieve a competitive ratio of 2 for related machines, where the power of preemption is at least 1.39 [Eps+17], we instead directly analyze the optimal migratory solution. This is possible because on related machines an optimal solution has many nice structural properties [Gon77].

5.1.3 Further Related Work

A technique related to our analysis framework has been introduced by Deng et al. [Den+00], and was also used later [Bea+12; JW24]. These works analyze non-clairvoyant algorithms with an inductive argument on the number of jobs by showing that truncating the schedule at the first completion time maintains the bound on the competitive ratio. Compared to our problem, these works can use simpler combinatorial lower bounds on an optimal solution to make this

approach work. Moreover, they assume that all jobs are available in the beginning, and it is not clear how to integrate non-uniform release dates into the inductive argument.

Non-clairvoyant scheduling to minimize the total weighted completion time on parallel identical machines is well-understood [Bea+12; Den+00; MV22b; MPT94]. While we show that even on a single machine, PF cannot be 2-competitive in the presence of release dates, we note that this bound is achieved by some other single-machine scheduling algorithm [Jäg+23]. For heterogeneous machines, besides the analysis by Im et al. [IKM18] for PF (and a slight improvement [LMR23]), there are several analyses for the more general total flow time objective with speed augmentation [Gup+12; GKP10; IKM18; Im+14], whose bounds can be translated to the completion time objective. Furthermore, Gupta et al. [GKS21] presented a non-clairvoyant algorithm for related machine scheduling with a special type of precedence constraints.

5.2 Preliminaries

PSP and Unrelated Machine Scheduling. In the polytope scheduling problem, jobs $j = 1, \dots, n$ arrive online at their release dates $r_j \geq 0$. When a job is released, its weight $w_j > 0$ as well as the column B_j from the matrix $B = (b_{dj})_{d,j} \in \mathbb{Q}_{\geq 0}^{D \times n}$ become known. A schedule assigns at every time $t \geq 0$ a processing rate $y_j(t) \geq 0$ to each job j with $r_j \leq t$, while all other jobs j have $y_j(t) = 0$. The resulting completion time C_j of a job j is the first time t' such that $\int_{r_j}^{t'} y_j(t)\,\mathrm{d}t \geq p_j$, where p_j denotes the processing requirement of job j. The processing rate vector $y(t)$ must satisfy, at any time t, the constraints $B \cdot y(t) \leq 1$, $y(t) \geq 0$ defining the polytope $\mathcal{P}$. The objective is to minimize the sum of weighted completion times $\sum_j w_j C_j$ of all jobs.

As mentioned before, unrelated machine scheduling with preemption and migration is a special case of PSP. To see this, note that we can model any schedule for a fixed time unit t using variables $x_{ijt} \geq 0$ that indicate whether job j runs on machine i at time t and constraints $\sum_{j=1}^{n} x_{ijt} \leq 1$ for every machine i and $\sum_{i=1}^{m} x_{ijt} \leq 1$ for every job j. The rate of job j at time t is thus equal to $y_j(t) = \sum_{i=1}^{m} s_{ij} x_{ijt}$, and we can write the polytope $\mathcal{Q}$ of feasible rate-allocation pairs as

$$\left\{ (y, x) \in \mathbb{R}_{\geq 0}^{n} \times \mathbb{R}_{\geq 0}^{m \times n} \;\middle|\; y_j = \sum_{i=1}^{m} s_{ij} x_{ij} \;\forall j \in [n],\ \sum_{j=1}^{n} x_{ij} \leq 1 \;\forall i \in [m],\ \sum_{i=1}^{m} x_{ij} \leq 1 \;\forall j \in [n] \right\}.$$

Since $\mathcal{Q}$ is downward-closed and full-dimensional in y, we can obtain a matrix $B \in \mathbb{Q}_{\geq 0}^{D \times n}$ such that the projection of $\mathcal{Q}$ to y is equal to $\{y \in \mathbb{R}_{\geq 0}^{n} \mid By \leq 1\}$ by applying Fourier-Motzkin elimination to $\mathcal{Q}$, showing that unrelated machine scheduling is indeed a special case of PSP.

Any fractional assignment x can be efficiently decomposed into integral assignments, which we can feasibly be scheduled with preemptions and migrations during a discrete time interval around t. We emphasize that migration is essential for non-clairvoyant algorithms to achieve a constant competitive ratio for unrelated machine scheduling. We show that this is true even in the special case of related machines.

Theorem 5.3. *Any non-migratory non-clairvoyant algorithm for minimizing the total completion time of n jobs on related machines, $Q \mid pmtn \mid \sum C_j$, has a competitive ratio of at least $\Omega(\sqrt{n})$.*

Proof. Consider n jobs and n machines, one with speed $\sqrt{n}$, and all others with speed 1. If the algorithm puts any job on a slow machine, then this job turns out to be very long, while all other jobs are negligible small in comparison. Therefore, the competitive ratio of such an algorithm would be at least $\sqrt{n}$. If the algorithm puts all jobs on the first machine, then they all have processing time 1, so that, even in an optimal schedule on that machine, the total completion time would be $\frac{n(n+1)}{2\sqrt{n}}$. An alternative schedule for all machines would process each job on a different machine, resulting in total completion time $1/\sqrt{n} + n - 1$. The ratio is thus at least $\frac{n(n+1)}{2(1+\sqrt{n}(n-1))} \in \Omega(\sqrt{n})$. □

Proportional Fairness. The Proportional Fairness strategy (PF), which has been analyzed for PSP by Im et al. [IKM18], computes at every release time and completion time the processing rates $y_j(t)$ of all jobs j that have been released but not yet completed. These jobs are constantly processed at the computed rates until a new job is released or completed. We denote by $J(t) \coloneqq \{j \in [n] \mid r_j \leq t < C_j\}$ the set of *available jobs* at time t in PF's schedule. Although the completion times of the jobs j with $C_j > t$ are still unknown at time t, it is already known at this point whether a job j belongs to $J(t)$, so that this set can be used by PF.

As mentioned in the introduction, at any instant t, PF maximizes the Weighted Nash Social Welfare, which is the weighted geometric mean of the allocated job rates among all feasible rate vectors $y(t) \in \mathcal{P}$ supported on $J(t)$. It is the solution to the following convex program applied to $J \coloneqq J(t)$:

$$\begin{aligned} \max \quad & \sum_{j \in J} w_j \cdot \log(y_j) && && \text{(CP}(J)\text{)} \\ \text{s.t.} \quad & \sum_{j \in J} b_{dj} \cdot y_j \leq 1 && \forall d \in [D] \\ & y_j \geq 0 && \forall j \in J \end{aligned}$$

Note that the PF strategy may not be executed exactly algorithmically because for some rational inputs the convex program may have only irrational solutions [JV10, Example 48]. While this is no issue for many settings, in general (CP(J)) can be solved to arbitrary precision $\delta > 0$ in time polynomial in the encoding length of B and w and polynomial in $\log \frac{1}{\delta}$ using the ellipsoid method [GLS88; Vis21]. As a consequence, we can algorithmically implement PF so that, for every $\varepsilon > 0$, we can compute rates in polynomial time and only lose a factor of $1 + \varepsilon$ in the competitive ratio for PSP. This is possible by approximately computing rates for a slower machine and increase them by δ afterwards.

Since PF computes rates at most $2n$ times, we say that it runs in polynomial time. Note that an actual schedule for these rates requires a pseudo-polynomial number of preemptions. However, with an implementation in geometrically increasing time windows one can reduce the number to a polynomial at the cost of an arbitrarily small constant in the competitive ratio [MPT94].

We can assume without loss of generality that $p_j > 0$, because every reasonable algorithm can complete a job j with $p_j = 0$ at time 0. Similarly, if there is a job j such that $y_j = 0$ for all $y \in \mathcal{P}$, no solution can complete job j. Thus, we assume that this is not the case. Then the definition of (CP(J)) implies the following observation.

Observation 5.4. *Let y be the solution to (CP(J)). Then, $y_j > 0$ for all $j \in J$.*

As the analysis of Im et al. [IKM18], our analysis uses the optimal Lagrange multipliers $(\eta_d)_d$ of the packing constraints, which satisfy with the optimal solution y the following KKT conditions [BV14], assuming that we use the natural logarithm in (CP(J)):

$$\frac{w_j}{y_j} - \sum_{d=1}^{D} b_{dj}\eta_d = 0 \quad \forall j \in J \tag{5.1}$$

$$\eta_d \cdot \left(1 - \sum_{j \in J} b_{dj} y_j\right) = 0 \quad \forall d \in [D] \tag{5.2}$$

$$\eta_d \geq 0 \quad \forall d \in [D] \tag{5.3}$$

By Observation 5.4, the optimal solution of (CP(J)) assigns every job a positive rate, hence we can omit the Lagrangian multipliers of the non-negativity constraints of (CP(J)) in the KKT conditions.

Lemma 5.5 ([IKM18, Lemma 3.3])**.** *Let J be a set of jobs, and let η be optimal Lagrange multipliers for (CP(J)). Then, $\sum_{d=1}^{D} \eta_d = \sum_{j \in J} w_j$.*

Proof. Let y be an optimal solution to (CP(J)). Then,

$$\sum_{d=1}^{D} \eta_d = \sum_{d=1}^{D} \eta_d \sum_{j \in J} b_{dj} y_j = \sum_{j \in J} y_j \sum_{d=1}^{D} b_{dj}\eta_d = \sum_{j \in J} y_j \frac{w_j}{y_j} = \sum_{j \in J} w_j$$

by using the KKT condition (5.2) in the first equality and (5.1) in the third equality. □

LP Relaxations and Dual Fitting. In the dual fitting analysis, PF is compared to the optimal solution that runs at speed $\frac{1}{\kappa}$ for some parameter $\kappa \geq 1$. We assume that the time is scaled in a way that all release dates and completion times in this schedule and in PF's schedule are integral (which is possible assuming rational input). In this case, the following linear program is a relaxation of the speed-scaled PSP, where the variables y_{jt} indicate the total amount of processing that job j receives in the interval $[t, t+1]$. It uses the total weighted mean busy time as an underestimation of the total weighted completion time [DW90; Goe96], and can be seen as the straightforward generalization of (LP$_R$) to PSP.

$$\begin{aligned}
\min \quad & \sum_{j \in J} w_j \sum_{t \geq r_j} \frac{y_{jt}}{p_j}\left(t + \frac{1}{2}\right) && && \text{(LP}(\kappa)\text{)} \\
\text{s.t.} \quad & \sum_{t \geq r_j} y_{jt} \geq p_j && \forall j \in J \\
& \sum_{j \in J} b_{dj} \cdot y_{jt} \leq \frac{1}{\kappa} && \forall d \in [D],\ \forall t \geq 0 \\
& y_{jt} \geq 0 && \forall j \in J,\ \forall t \geq r_j
\end{aligned}$$

The scalability of completion times immediately implies that the optimal objective value of (LP(κ)) lower bounds $\kappa \cdot \mathrm{OPT}$ for every $\kappa \geq 1$. The dual of (LP(κ)) can be written as follows.

$$\begin{aligned} \max \quad & \sum_{j \in J} a_j - \sum_{d=1}^{D} \sum_{t \geq 0} b_{dt} && && \text{(DLP}(\kappa)\text{)} \\ \text{s.t.} \quad & \frac{a_j}{p_j} - \frac{w_j}{p_j}\left(t + \frac{1}{2}\right) \leq \kappa \sum_{d=1}^{D} b_{dj} b_{dt} && \forall j \in J,\ \forall t \geq r_j \\ & a_j, b_{dt} \geq 0 && \forall j \in J,\ \forall t \geq 0,\ \forall d \in [D] \end{aligned}$$

Offline Complexity. The offline PSP is APX-hard, even for uniform release dates. This follows for example from the hardness of preemptive unrelated machine scheduling [Sit17]. On the positive side, constant-factor approximation algorithms can be obtained via standard techniques [IKM18]. Specifically, randomized α-point rounding [QS02; SS97] applied to an optimal solution of a variant of (LP(1)) yields a $(2+\varepsilon)$-approximation for PSP; for details we refer to [JLM25]. The proof also implies that (LP(1)) lower bounds the optimal objective value within a factor of $\frac{1}{2}$.

5.3 General PSP

We give a simplified and improved variant of the analysis of PF for PSP of Im et al. [IKM18].

Theorem 5.6. *PF has a competitive ratio of at most 27 for minimizing the total weighted completion time of PSP.*

The remaining section is devoted to the proof of this theorem. Fix an instance and PF's schedule. Let $\kappa \geq 1$ and $0 < \lambda < 1$ be constants, which we fix later. In the following, we assume by scaling that all weights are integers.

For every time t, let $\zeta(t)$ be the weighted λ-quantile of the values $y_j(t)/p_j$, $j \in U(t)$, with respect to the weights w_j. More formally, if Z_t denotes the sorted (ascending) list of length $W(t)$ composed of w_j copies of $y_j(t)/p_j$ for every $j \in U(t)$, then $\zeta(t)$ is the value at the index $\lceil \lambda W(t) \rceil$ in Z_t. Let $\bar{a}_{jt} \coloneqq w_j \cdot \mathbb{1}\left[y_j(t)/p_j \leq \zeta(t)\right]$ for $t \geq 0$ and $j \in U(t)$, where $\mathbb{1}[\varphi]$ is the indicator variable of the expression φ. Further, for $t \geq 0$ let $\eta_d(t)$ be the optimal Lagrange multiplier corresponding to the constraint $d \in [D]$ of (CP($J(t)$)). We consider the following dual solution:

- $\bar{a}_j \coloneqq \sum_{t=0}^{C_j} \bar{a}_{jt}$ for every job $j \in J$,
- $\bar{b}_{dt} \coloneqq \frac{1}{\kappa} \sum_{t' \geq t} \zeta(t') \cdot \eta_d(t')$ for every $d \in [D]$ and time $t \geq 0$.

We first show in the following lemma that the objective value of (DLP(κ)) for this assignment upper bounds a constant fraction of the algorithm's objective value.

Lemma 5.7. *It holds that* $\left(\lambda - \frac{1}{(1-\lambda)\kappa}\right)\mathrm{ALG} \leq \sum_{j \in J} \bar{a}_j - \sum_{d=1}^{D} \sum_{t \geq 0} \bar{b}_{dt}$.

The lemma follows from the following two statements.

Lemma 5.8. *It holds that* $\sum_{j \in J} \bar{a}_j \geq \lambda \cdot \mathrm{ALG}$.

Proof. Consider a time t. Observe that $\sum_{j\in U(t)} \bar{a}_{jt}$ contains the total weight of jobs j that satisfy $y_j(t)/p_j \le \zeta(t)$. By the definition of $\zeta(t)$, we conclude that this is at least $\lambda \cdot W(t)$, that is, $\sum_{j\in U(t)} \bar{a}_{jt} \ge \lambda \cdot W(t)$. The statement then follows by summation over time. □

Lemma 5.9. *At any time t, it holds that $\sum_{d=1}^{D} \bar{b}_{dt} \le \frac{1}{(1-\lambda)\kappa} W(t)$.*

Proof. Fix a time t. Observe that for every $t' \ge t$, the definition of $\zeta(t')$ implies that $\sum_{j\in U(t')} w_j \cdot \mathbb{1}\left[y_j(t')/p_j \ge \zeta(t')\right] \ge (1-\lambda)W(t')$. Thus,

$$\zeta(t') \cdot (1-\lambda)W(t') \le \sum_{j\in U(t')} w_j \cdot \zeta(t') \cdot \mathbb{1}\left[\frac{y_j(t')}{p_j} \ge \zeta(t')\right] \le \sum_{j\in U(t')} w_j \cdot \frac{y_j(t')}{p_j} \,. \tag{5.4}$$

The definition of $\bar{b}_{it}$ and Lemma 5.5 imply

$$\begin{aligned}\sum_{d=1}^{D} \bar{b}_{dt} = \sum_{d=1}^{D} \frac{1}{\kappa} \sum_{t'\ge t} \eta_d(t') \cdot \zeta(t') = \frac{1}{\kappa} \sum_{t'\ge t} \zeta(t') \sum_{d=1}^{D} \eta_d(t') = \frac{1}{\kappa} \sum_{t'\ge t} \zeta(t') \sum_{j\in J(t')} w_j \\ \le \frac{1}{\kappa} \sum_{t'\ge t} \zeta(t') \cdot W(t') \,.\end{aligned}$$

Using (5.4), we conclude that this is at most

$$\begin{aligned}\frac{1}{\kappa} \sum_{t'\ge t} \zeta(t') W(t') &\le \frac{1}{(1-\lambda)\kappa} \sum_{t'\ge t} \sum_{j\in U(t')} w_j \cdot \frac{y_j(t')}{p_j} \\ &\le \frac{1}{(1-\lambda)\kappa} \sum_{j\in U(t)} w_j \sum_{t'\ge t} \frac{y_j(t')}{p_j} \le \frac{1}{(1-\lambda)\kappa} \sum_{j\in U(t)} w_j \,.\end{aligned}$$

The second inequality follows from $U(t') \subseteq U(t)$ for $t' \ge t$. The third inequality holds because $\sum_{t'\ge t} y_j(t') \le p_j$ for every job j. This concludes the proof of the lemma. □

Next, we show dual feasibility.

Lemma 5.10. *The dual solution $(\bar{a}_j)_j$ and $(\bar{b}_{it})_{i,t}$ is feasible for (DLP(κ)).*

Proof. Fix a job j, a machine i and a time $t \ge r_j$. Then,

$$\begin{aligned}\frac{\bar{a}_j}{p_j} - \frac{w_j}{p_j}\left(t+\frac{1}{2}\right) &\le \sum_{t'=0}^{C_j} \frac{w_j}{p_j} \cdot \mathbb{1}\left[\frac{y_j(t')}{p_j} \le \zeta(t')\right] - \frac{w_j}{p_j} t \le \sum_{t'=t}^{C_j} \frac{w_j}{p_j} \cdot \mathbb{1}\left[\frac{y_j(t')}{p_j} \le \zeta(t')\right] \\ &= \sum_{t'=t}^{C_j} \frac{w_j}{y_j(t')} \cdot \frac{y_j(t')}{p_j} \cdot \mathbb{1}\left[\frac{y_j(t')}{p_j} \le \zeta(t')\right] \\ &= \sum_{t'=t}^{C_j} \sum_{d=1}^{D} b_{dj} \cdot \eta_d(t') \cdot \frac{y_j(t')}{p_j} \cdot \mathbb{1}\left[\frac{y_j(t')}{p_j} \le \zeta(t')\right] \\ &\le \sum_{t'=t}^{C_j} \sum_{d=1}^{D} b_{dj} \cdot \eta_d(t') \cdot \zeta(t') \le \sum_{d=1}^{D} b_{dj} \sum_{t'\ge t} \eta_d(t') \cdot \zeta(t') = \kappa \sum_{d=1}^{D} b_{dj} \bar{b}_{dt} \,.\end{aligned}$$

The second equality uses (5.1). This concludes the proof of the lemma. □

Finally, we prove Theorem 5.6. Weak duality, Lemma 5.7, and Lemma 5.10 imply

$$\kappa \cdot \mathrm{OPT} \geq \sum_{j \in J} \bar{a}_j - \sum_{d=1}^{D} \sum_{t \geq 0} \bar{b}_{dt} \geq \left(\lambda - \frac{1}{(1-\lambda)\kappa}\right) \cdot \mathrm{ALG}\,.$$

Choosing $\kappa = 9$ and $\lambda = \frac{2}{3}$ implies $\mathrm{ALG} \leq 27 \cdot \mathrm{OPT}$.

5.4 PF-Monotone PSP

In this section, we consider the class of PF-monotone PSP (MonPSP, cf. Definition 5.1) and prove the following theorem.

Theorem 5.11. *PF has a competitive ratio of at most 4 for minimizing the total weighted completion time for MonPSP.*

Further, we show implications of this result for specific machine scheduling problems in MonPSP, namely, related machine scheduling and restricted assignment (Theorem 5.15). We also point out the implications of our work for matching markets in this section (Corollaries 5.17 and 5.18). As mentioned before, unrelated machine scheduling is not PF-monotone. We give an example in Section 5.4.3.

Recall that MonPSP consists of instances to PSP whose polytope $\mathcal{P}$ has the property that for all possible sets $J' \subseteq J$ of available jobs with corresponding rate vectors y' and y computed by PF it holds that $y'_j \geq y_j$ for all $j \in J'$ (cf. Definition 5.1).

5.4.1 Competitive Analysis

To show the upper bound of 4 on the competitive ratio (Theorem 5.11), we fix an arbitrary instance of MonPSP and the corresponding PF schedule. Let C_j denote the completion time of job j in that schedule, and let $\mathrm{ALG} := \sum_{j=1}^{n} w_j C_j$ denote the objective function value of PF. For each time t, in addition to the actual rate vector $y(t)$ of PF, we consider the rate vector $\hat{y}(t) \in \mathcal{P}$ that PF would choose if all jobs $j \in U(t)$ were available at time t. In other words, $y(t)$ is the optimal solution to (CP($J(t)$)) and $\hat{y}(t)$ is the optimal solution to (CP($U(t)$)). Note that, while we use these rates for the analysis, during the actual execution, PF cannot use them because it has no access to unreleased jobs. Moreover, the set $U(t)$ always refers to the unfinished jobs in the actual PF schedule and not in the schedule with rates $\hat{y}(t)$. Let $\hat{\eta}_d(t)$, $d \in [D]$, be the optimal Lagrange multipliers corresponding to (CP($U(t)$)).

Using $J(t) \subseteq U(t)$ for any time t and monotonicity, we make the following observation.

Observation 5.12. *At every time t and for every $j \in J(t)$ it holds that $\hat{y}_j(t) \leq y_j(t)$.*

We now perform a dual fitting argument via (DLP(κ)) for arbitrary $\kappa \geq 1$, which we later set to 2. To this end, we consider the following assignment of dual variables:

- $\bar{a}_j := w_j C_j$ for every job $j \in J$,
- $\bar{b}_{dt} := \frac{1}{\kappa} \hat{\eta}_d(t)$ for every $d \in [D]$ and time $t \geq 0$.

Lemma 5.13. *It holds that* $\left(1 - \frac{1}{\kappa}\right)\text{ALG} = \sum_{j \in J} \bar{a}_j - \sum_{d=1}^{D} \sum_{t \geq 0} \bar{b}_{dt}$.

Proof. First, note that $\sum_{j \in J} \bar{a}_j = \text{ALG}$. Moreover, the definition of $\bar{b}_{dt}$ and Lemma 5.5 imply that $\sum_{d=1}^{D} \bar{b}_{dt} = \frac{1}{\kappa} W(t)$ at every time t. Thus, $\sum_{t \geq 0} \sum_{d=1}^{D} \bar{b}_{dt} = \frac{1}{\kappa}\text{ALG}$, which concludes the proof. □

Lemma 5.14. *The solution* $(\bar{a}_j)_j$ *and* $(\bar{b}_{dt})_{d,t}$ *is feasible for* (DLP(κ)).

Proof. The dual variables as defined above are clearly non-negative. We now verify the dual constraint. Fix a job j and a time $t \geq r_j$. Then,

$$\frac{\bar{a}_j}{p_j} - \left(t + \frac{1}{2}\right) \cdot \frac{w_j}{p_j} \leq (C_j - t) \cdot \frac{w_j}{p_j} \leq \sum_{t'=t}^{C_j - 1} \frac{w_j}{p_j} = \sum_{t'=t}^{C_j - 1} \frac{\hat{y}_j(t')}{p_j} \cdot \frac{w_j}{\hat{y}_j(t')} \leq \sum_{t'=t}^{C_j - 1} \frac{y_j(t')}{p_j} \cdot \frac{w_j}{\hat{y}_j(t')} .$$

The last inequality uses Observation 5.12. Now observe that for every time $t' \geq t$ it holds that $U(t') \subseteq U(t)$, which implies $\hat{y}_j(t') \geq \hat{y}_j(t)$ via monotonicity. Thus, the above is at most

$$\frac{w_j}{\hat{y}_j(t)} \sum_{t'=t}^{C_j - 1} \frac{y_j(t')}{p_j} = \left(\sum_{d=1}^{D} b_{dj} \hat{\eta}_d(t) \right) \sum_{t'=t}^{C_j - 1} \frac{y_j(t')}{p_j} \leq \kappa \sum_{d=1}^{D} b_{dj} \bar{b}_{dt} ,$$

where the equality uses the KKT condition (5.1) for (CP($U(t)$)), and the inequality holds because j receives at most as much processing rate as it requires. This concludes the proof. □

Proof of Theorem 5.11. Weak duality, Lemma 5.13, and Lemma 5.14 imply

$$\kappa \cdot \text{OPT} \geq \sum_{j \in J} \bar{a}_j - \sum_{d=1}^{D} \sum_{t \geq 0} \bar{b}_{dt} \geq \left(1 - \frac{1}{\kappa}\right) \cdot \text{ALG} .$$

Setting $\kappa = 2$ implies $\text{ALG} \leq 4 \cdot \text{OPT}$. □

5.4.2 Implications for Machine Scheduling and Matching Markets

In this section, we prove the following theorem and thereby present new insights for one-sided matching markets. For an introduction into Fisher markets and matching markets, we refer to [Nis+07; VY21].

Theorem 5.15. *There is a strongly polynomial-time non-clairvoyant online algorithm for minimizing the total weighted completion time on related machines,* $Q \mid r_j, pmtn \mid \sum w_j C_j$*, and for restricted assignment,* $R \mid r_j, pmtn, s_{ij} \in \{0, 1\} \mid \sum w_j C_j$*, with a competitive ratio of at most* 4.

We first note that both scheduling problems, on related machines and with restricted assignment, fall under MonPSP. Hence, PF is 4-competitive by Theorem 5.11. For scheduling on related machines, this has been shown by Im et al. [IKM18] via a connection to abstract markets, which we present in the following. Then, we prove that scheduling with restricted assignment falls also under MonPSP. Finally, we argue that PF can be implemented in strongly polynomial time for both problems.

Monotonicity via Submodular Utility Allocation Markets. In *uniform utility allocation* (UUA) *markets*, introduced by Jain and Vazirani [JV10], n buyers j with budgets w_j want to maximize their own utility y_j, but there are constraints of the form $\sum_{j\in S} y_j \leq v(S)\ \forall S \subseteq [n]$ for some set function $v\colon 2^{[n]} \to \mathbb{R}_{\geq 0}$, coupling the utilities of different buyers. Every constraint can post a price η_S that each buyer $j \in S$ has to pay per unit of utility. That means, the total amount to be paid by j is the sum of prices of the constraints affecting j. A *market equilibrium* consists of a feasible utility allocation y together with prices $\eta \geq 0$ such that only tight constraints have a positive price and each buyer exactly uses up their budget, thus maximizing their own utility.

Jain and Vazirani [JV10] showed for UUA markets that optimal solutions to the Weighted Nash Social Welfare problem correspond to market equilibria. This problem is exactly the problem (CP($[n]$)) for the polytope $\mathcal{P} = \{y \in \mathbb{R}^n_{\geq 0} \mid \sum_{j\in S} y_j \leq v(S)\ \forall S \subseteq [n]\}$. Moreover, they proved that if v is monotone, submodular and $v(\emptyset) = 0$, that is, if $\mathcal{P}$ is a polymatroid, then the market (termed *submodular utility allocation* (SUA) *market*) is competition-monotone, meaning that if some budgets are reduced, the utilities of other buyers cannot decrease. By taking buyers as jobs and utilities as processing rates, this implies that PSP is PF-monotone on $\mathcal{P}$ because we can model the restriction to a subset of jobs by setting the weight of the other jobs to 0. Consequently, it suffices to show that the polytopes associated to our scheduling problems are polymatroids. Im et al. [IKM18] proved this for scheduling on related machines. We now show this statement for scheduling with restricted assignment. This implies that $R \mid r_j, \text{pmtn}, s_{ij} \in \{0,1\} \mid \sum w_j C_j$ is PF-monotone.

Lemma 5.16. *For any instance of preemptive scheduling with restricted assignment, the associated polytope $\mathcal{P}$ is a polymatroid.*

Proof. Consider the bipartite graph $G \coloneqq (J \dot\cup [m], E)$ with an edge $\{j,i\} \in E$ whenever job j can be processed on machine i. Then, letting δ denote the set of incident edges, we have

$$\mathcal{P} = \left\{ y \in \mathbb{R}^n_{\geq 0} \;\middle|\; \exists x \in \mathbb{R}^E_{\geq 0} : y_j = \sum_{e\in\delta(j)} x_e\ \forall j \in J,\ \sum_{e\in\delta(i)} x_e \leq 1\ \forall i \in [m],\ \sum_{e\in\delta(j)} x_e \leq 1\ \forall j \in J \right\},$$

that is, $\mathcal{P}$ contains for every fractional matching of G the vector y of fractional covering rates of all nodes j on the left-hand side J of G. By the Birkhoff-von Neumann Theorem, every fractional matching is a convex combination of integral matchings. Then the covering rates of the nodes are also the corresponding convex combinations of the incidence vectors of the subsets of nodes from J covered by the integral matchings. Therefore, $\mathcal{P}$ is the convex hull of the incidence vectors of subsets of J covered by a matching in G. This is exactly the independence polytope associated with the transversal matroid of G (cf. [Oxl11]), which is in particular a polymatroid. □

For matching markets with dichotomous utilities, which correspond to scheduling with restricted assignment, Garg et al. [GTV22] gave a direct proof that (in the unit budget case) optimal solutions to the Nash Social Welfare problem (formulated in terms of the allocation x) correspond to market equilibria, known as Hylland-Zeckhauser (HZ) equilibria [HZ79]. Lemma 5.16 implies that such a market can be modeled as a UUA market, and thus, even in the more general case where buyers have different budgets, the work of Jain and Vazirani [JV10] also implies this

equivalence between the HZ equilibria and the Weighted Nash Social Welfare solution. Hence, we obtain the following slightly stronger result using an arguably simpler argumentation.

Corollary 5.17. *For one-sided matching markets with dichotomous utilities and arbitrary budgets, an HZ equilibrium is an optimal solution to* (CP)*, and every optimal solution to* (CP) *can be extended to an HZ equilibrium.*

Strongly Polynomial Algorithms. Since the rate polytopes for both scheduling on related machines and with restricted assignment are polymatroids, we can use the combinatorial algorithm of Jain and Vazirani [JV10] for SUA markets, which is entirely stated in terms of utilities and based on submodular function minimization, to compute a market equilibrium, and thus, a rate allocation y that is optimal for (CP). A corresponding machine allocation x can be computed easily by finding a feasible fractional assignment. This completes the proof of Theorem 5.15.

This algorithm is thus an alternative to the recently proposed strongly polynomial-time algorithm for computing a market equilibrium in matching markets with dichotomous utilities proposed by Vazirani and Yannakakis [VY21], which directly incorporates the allocation x. Since this problem exactly corresponds to PF's allocation problem for scheduling with restricted assignment, an extension of their algorithm to arbitrary budgets [GTV22] can also be used in PF. While their algorithm is conceptually simpler than the algorithm by Jain and Vazirani [JV10] for general SUA markets, the latter implies the following result via an arguably simpler argumentation.

Corollary 5.18. *An HZ equilibrium in one-sided matching markets with dichotomous utilities can be computed in strongly polynomial time.*

For scheduling on related machines, we demonstrate in Section 5.6, analogously to the algorithms for scheduling with restricted assignment and one-sided matching markets with dichotomous utilities [GTV22; VY21], that also for this problem there is a simpler and faster algorithm that directly works with the allocation x. Moreover, our combinatorial algorithm reveals that the known 2-competitive non-clairvoyant algorithm for identical parallel machines [Bea+12] is a special case of PF.

5.4.3 Unrelated Machine Scheduling is not PF-Monotone

We have shown that PF-monotonicity is a powerful property that allows for the derivation of very good bounds on the competitive ratio of PF (see Theorem 5.11). The fact that several machine scheduling problems in this framework are PF-monotone, including scheduling on related machines and restricted assignment (see Theorem 5.15), raises whether the general unrelated machine scheduling problem also has this property. We answer this question to the negative, even in the absence of weights and release times.

Lemma 5.19. *The unrelated machine scheduling problem $R \mid pmtn \mid \sum C_j$ is not PF-monotone.*

Proof. The representation of unrelated machines scheduling as PSP uses the following polytope:

$$\left\{ y \in \mathbb{R}^n_{\geq 0} \;\middle|\; \exists x \in \mathbb{R}^{m \times n}_{\geq 0} : y_j = \sum_{i=1}^{m} s_{ij} x_{ij} \; \forall j \in [n],\ \sum_{j=1}^{n} x_{ij} \leq 1 \; \forall i \in [m],\ \sum_{i=1}^{m} x_{ij} \leq 1 \; \forall j \in [n] \right\}.$$

Consider the following instance with three machines, unit-weight jobs $J = \{1, 2, 3\}$ and the following speeds s_{ij} for jobs j on machines i:

$$\begin{aligned} s_{11} &= 1\,, \quad s_{21} = 0\,, \quad s_{31} = 0\,, \\ s_{12} &= 2\,, \quad s_{22} = 1\,, \quad s_{32} = 0\,, \\ s_{13} &= 0\,, \quad s_{23} = 2\,, \quad s_{33} = 1\,. \end{aligned}$$

Let $J' = \{1, 2\}$. The optimal solution to (CP(J)) is $y = (2/3, 4/3, 4/3)$, and the optimal solution to (CP(J')) is $y' = (1, 1)$. Observe that PF's rate for job 2 decreases in the absence of job 3. □

5.5 Superadditive PSP

In this section, we consider an α-superadditive PSP instance for some $\alpha \geq 1$. That means that for an arbitrary partition of the processing requirements, the sum of the optimal objective values of the subinstances is at most α times the optimal objective value of the whole instance; cf. Definition 5.2. We first present the analysis framework and prove the following theorem.

Theorem 5.20. *PF has a competitive ratio of at most $2\alpha + 1$ for minimizing the total weighted completion time for α-superadditive PSP. For uniform release dates, this bound reduces to 2α.*

Subsequently, we apply this result and analysis framework to several machine scheduling problems by proving bounds on the superadditivity.

5.5.1 Framework

The proof of Theorem 5.20 consists of three steps: first we decompose the instance into structured subinstances according to the PF schedule. Second we show competitiveness bounds for the resulting structured subinstances, and third we combine the obtained bounds using the α-superadditivity.

Decomposing the PF Schedule. Let C_j, $j \in [n]$, denote the completion times of the jobs in the PF schedule, and let $E_1 < \ldots < E_{L+1}$ be the times when jobs arrive or complete in the PF schedule. We assume without loss of generality that $E_1 = 0$. For every $1 \leq \ell \leq L$ and job j we denote by $p_j^{(\ell)}$ the amount of processing that j receives during $[E_\ell, E_{\ell+1}]$. Let $\mathrm{ALG}(p)$ be the objective value of the schedule constructed by PF, and let $\mathrm{ALG}_0(p^{(\ell)})$ be the objective value of the PF schedule applied to jobs with processing requirements $p^{(\ell)} = (p_1^{(\ell)}, \ldots, p_n^{(\ell)})$ available at time 0. Note that $p = \sum_{\ell=1}^{L} p^{(\ell)}$, because PF computes a feasible schedule. We decompose PF's objective value according to the following lemma.

Lemma 5.21. *It holds that*

$$\mathrm{ALG}(p) = \sum_{\ell=1}^{L} \left(\mathrm{ALG}_0(p^{(\ell)}) + (E_{\ell+1} - E_\ell) \cdot \sum_{j \in U(E_\ell) \setminus J(E_\ell)} w_j \right).$$

Proof. Observation 5.4 implies that a job j is available during an interval $(E_\ell, E_{\ell+1}]$ if and only if $p_j^{(\ell)} > 0$. Hence, the jobs available after time 0 in the subinstance with processing

requirements $p^{(\ell)}$ and uniform release dates, denoted by $I^{(\ell)}$, are exactly the jobs available in the original instance during $(E_\ell, E_{\ell+1}]$. Since PF takes an instantaneous resource allocation view, the composed processing rates depend only on the weights of the currently available jobs, that is, the rate that job j receives in the original instance during $(E_\ell, E_{\ell+1}]$ is equal to the rate that j receives in $I^{(\ell)}$. Moreover, the rates are constant during the considered interval, so that for every available job j it only happens at the end of the interval that its total processing rate received within the interval reaches $p_j^{(\ell)}$. For $I^{(\ell)}$ this means that all jobs with a positive processing requirement finish exactly at time $E_{\ell+1} - E_\ell$ and all jobs with a processing requirement equal to zero finish at time 0. Thus, $\mathrm{ALG}_0(p^{(\ell)}) = \sum_{j \in J(E_\ell)} w_j \cdot (E_{\ell+1} - E_\ell)$. Therefore, we conclude with

$$\begin{aligned}\mathrm{ALG}(p) &= \sum_{j=1}^{n} w_j C_j = \sum_{j=1}^{n} w_j \sum_{\ell : E_\ell < C_j} (E_{\ell+1} - E_\ell) = \sum_{\ell=1}^{L} (E_{\ell+1} - E_\ell) \sum_{j \in U(E_\ell)} w_j \\ &= \sum_{\ell=1}^{L} \left(\mathrm{ALG}_0(p^{(\ell)}) + (E_{\ell+1} - E_\ell) \cdot \sum_{j \in U(E_\ell) \setminus J(E_\ell)} w_j \right). \qquad \square\end{aligned}$$

Analyzing Structured Instances. We now show that PF is 2-competitive against (DLP(1)) for PSP instances where all jobs are available at time 0 and PF completes all jobs with positive processing requirements at the same time. That is, the rate $y_j(0)$ allocated to each job j does not change and is proportional to its processing requirement p_j. Any job with a processing requirement equal to 0 completes at time 0, and thus, can without loss of generality be removed from the instance.

We fix a PSP instance of jobs J with these properties and assume that in the optimal schedule all completion times are integral. Let C be the common completion time in the PF schedule and also assume by scaling that C is an integer. Since the rates and Lagrange multipliers do not change, we write $y_j = y_j(t)$, $j \in J$, and $W = W(t)$ and η_d for the Lagrange multipliers corresponding to (CP($J(t)$)) for all $t \in [0, C)$. We perform a dual fitting argument using the following assignment:

- $\bar{a}_j \coloneqq w_j C$ for every job $j \in J$,
- $\bar{b}_{dt} \coloneqq \left(1 - \frac{1}{C}(t + \frac{1}{2})\right) \cdot \eta_d$ for every $d \in [D]$ and time $t \in \{0, \dots, C-1\}$.

Lemma 5.22. *It holds that* $\frac{1}{2} \cdot \mathrm{ALG} = \sum_{j \in J} \bar{a}_j - \sum_{t=0}^{C-1} \sum_{d=1}^{D} \bar{b}_{dt}$.

Proof. Clearly, $\sum_{j \in J} \bar{a}_j = \mathrm{ALG}$. Further, at every time t it holds that $\sum_{d=1}^{D} \bar{b}_{dt} = (1 - \frac{1}{C}(t + \frac{1}{2})) \cdot W$ due to Lemma 5.5. Thus, we conclude the proof with

$$\sum_{t=0}^{C-1} \sum_{d=1}^{D} \bar{b}_{dt} = W \sum_{t=0}^{C-1} \left(1 - \frac{1}{C}\left(t + \frac{1}{2}\right)\right) = \frac{W}{C} \sum_{t=0}^{C-1} \left(t + \frac{1}{2}\right) = \frac{1}{2} \cdot W \cdot C = \frac{1}{2} \cdot \mathrm{ALG}\,. \qquad \square$$

Lemma 5.23. *The solution* $(\bar{a}_j)_j$ *and* $(\bar{b}_{dt})_{d,t}$ *is feasible for* (DLP(1)).

Proof. The solution is clearly non-negative. We now verify the dual constraint. Fix a job j and a time $t \in \{0, \dots, C-1\}$. Then,

$$\begin{aligned}\frac{\bar{a}_j}{p_j} - \left(t+\frac{1}{2}\right)\cdot\frac{w_j}{p_j} &= \left(C-\left(t+\frac{1}{2}\right)\right)\cdot\frac{w_j}{p_j} = \left(C-\left(t+\frac{1}{2}\right)\right)\cdot\frac{w_j}{y_j}\cdot\frac{y_j}{p_j} \\ &= \left(C-\left(t+\frac{1}{2}\right)\right)\cdot\left(\sum_{d=1}^{D} b_{dj}\eta_d\right)\cdot\frac{y_j}{p_j} = \left(1-\frac{1}{C}\left(t+\frac{1}{2}\right)\right)\cdot\left(\sum_{d=1}^{D} b_{dj}\eta_d\right) \\ &= \sum_{d=1}^{D} b_{dj}\cdot\left(1-\frac{1}{C}\left(t+\frac{1}{2}\right)\right)\cdot\eta_d = \sum_{d=1}^{D} b_{dj}\bar{b}_{dt}\,.\end{aligned}$$

The third equality uses (5.1). To see the fourth equality, note that $C \cdot y_j = p_j$. This shows that the dual assignment satisfies the constraint of (DLP(1)) with equality. □

Theorem 5.24. *PF has a competitive ratio equal to 2 for minimizing the total weighted completion time for PSP with uniform release dates whenever it completes all jobs at the same time.*

Proof. Weak duality, Lemma 5.22, and Lemma 5.23 imply

$$\mathrm{OPT} \geq \sum_{j\in J}\bar{a}_j - \sum_{t=0}^{C-1}\sum_{d=1}^{D}\bar{b}_{dt} = \frac{1}{2}\cdot\mathrm{ALG}\,,$$

which proves the claimed upper bound on the competitive ratio. The lower bound follows by noting that in the deterministic lower bound instance all jobs also complete at the same time [MPT94]. □

Lemmas 5.22 and 5.23 provide the additional insight that PF's schedule $(y_j(t))_{j,t}$ for structured instances is an optimal solution to (LP(1)). This follows from strong duality because $(y_j(t))_{j,t}$ is feasible for (LP(1)) and its objective value is also equal to $\frac{1}{2}\mathrm{ALG}$ as $\sum_{t=0}^{C-1}(t+\frac{1}{2})y_j(t)/p_j = \frac{1}{2}C$ for all j.

Combining Optimal Schedules via Superadditivity. We consider the partition of the processing requirements $p = \sum_{\ell=1}^{L} p^{(\ell)}$, which we initially defined within this framework. As argued in the proof of Lemma 5.21, in the instances with processing requirements $p^{(\ell)}$ and uniform release dates, all jobs finish at the same time. Let $\mathrm{OPT}_0(p^{(\ell)})$ be the optimal objective value for these instances. By combining Lemma 5.21 and Theorem 5.24, we obtain

$$\mathrm{ALG}(p) \leq \sum_{\ell=1}^{L}\left(2\cdot\mathrm{OPT}_0(p^{(\ell)}) + (E_{\ell+1}-E_\ell)\cdot\sum_{j\in U(E_\ell)\setminus J(E_\ell)} w_j\right).$$

The definition of α-superadditivity implies that this is at most

$$2\alpha\cdot\mathrm{OPT}_0(p) + \sum_{j=1}^{n} w_j \sum_{\ell:E_\ell<r_j}(E_{\ell+1}-E_\ell) = 2\alpha\cdot\mathrm{OPT}_0(p) + \sum_{j=1}^{n} w_j r_j \leq (2\alpha+1)\cdot\mathrm{OPT}(p)\,,$$

where the final inequality uses the trivial bound $r_j \le C_j^{\mathrm{OPT}}$ for all $j \in [n]$ and that the problem with uniform release dates is a relaxation of the general PSP. This proves the bound for the general case. Note that for uniform release dates, we can assume that $\sum_{j=1}^{n} w_j r_j = 0$, so that we can bound the objective of PF by $2\alpha \cdot \mathrm{OPT}$. This concludes the proof of Theorem 5.20.

5.5.2 Applications in Machine Scheduling

In this section, we apply the analysis framework from Section 5.5.1 to several important machine scheduling environments.

Unrelated Machines. In order to show superadditivity for preemptive unrelated machine scheduling, denoted as $R \,|\, \mathrm{pmtn} \,|\, \sum w_j C_j$, we fall back on the non-preemptive (and hence non-migratory) variant. In this problem, denoted as $R \,||\, \sum w_j C_j$, every job must be processed uninterruptedly on a complete machine, that is, it processes at a rate equal to the speed of the machine for this job. For fixed weights and speeds, and for processing requirements p let $\mathrm{OPT}_0^{\mathrm{np}}(p)$ denote the optimal objective value of a non-preemptive schedule for release dates 0. Note that this non-preemptive problem is not a special case of PSP. However, it is straightforward to extend the definition of α-superadditivity to it.

Lemma 5.25. *The problem $R \,||\, \sum w_j C_j$ is 1-superadditive.*

Proof. We can model any schedule for the non-preemptive problem using binary variables x_{ijk} that indicate whether job j is being scheduled in the kth last position on machine i. The objective value of this schedule for processing requirements $p_1, \dots, p_n$ is then equal to

$$\sum_{i=1}^{m} \sum_{k=1}^{n} \sum_{j=1}^{n} \frac{p_j}{s_{ij}} x_{ijk} \sum_{j'=1}^{n} w_{j'} \sum_{k'=1}^{k} x_{ij'k'} \,,$$

subject to the constraints that $\sum_{i=1}^{m} \sum_{k=1}^{n} x_{ijk} = 1$ for every job $j \in [n]$ and $\sum_{j=1}^{n} x_{ijk} \le 1$ for every position $k \in [n]$ and machine $i \in [m]$.

We now prove that the problem is 1-superadditive. Let $(x_{ijk})_{i,j,k}$ model an optimal schedule for processing requirements p, and let $p = \sum_{\ell=1}^{L} p^{(\ell)}$ be an arbitrary partition of the processing requirements. Thus,

$$\begin{aligned} \mathrm{OPT}_0^{\mathrm{np}}(p) &= \sum_{i=1}^{m} \sum_{k=1}^{n} \sum_{j=1}^{n} \frac{p_j}{s_{ij}} x_{ijk} \sum_{j'=1}^{n} w_{j'} \sum_{k'=1}^{k} x_{ij'k'} \\ &= \sum_{\ell=1}^{L} \left(\sum_{i=1}^{m} \sum_{k=1}^{n} \sum_{j=1}^{n} \frac{p_j^{(\ell)}}{s_{ij}} x_{ijk} \sum_{j'=1}^{n} w_{j'} \sum_{k'=1}^{k} x_{ij'k'} \right) \ge \sum_{\ell=1}^{L} \mathrm{OPT}_0^{\mathrm{np}}(p^{(\ell)}) \,, \end{aligned}$$

where the inequality holds because for every $1 \le \ell \le L$, $(x_{ijk})_{i,j,k}$ is a feasible solution to the instance with processing requirements $p^{(\ell)}$. □

Lemma 5.26. *The problem $R \,|\, pmtn \,|\, \sum w_j C_j$ is 1.81-superadditive.*

Proof. Clearly, the preemptive problem is a relaxation of the non-preemptive problem, that is, $\mathrm{OPT}_0(p^{(\ell)}) \le \mathrm{OPT}_0^{\mathrm{np}}(p^{(\ell)})$ for all $\ell \in [L]$. Further, we apply a known bound on the

power of preemption in unrelated machine scheduling. Sitters [Sit17, Corollary 3] proved that $\mathrm{OPT}_0^{\mathrm{np}}(p) \leq 1.81 \cdot \mathrm{OPT}_0(p)$ for any instance p. By combining these bounds with Lemma 5.25, we obtain

$$\sum_{\ell=1}^{L} \mathrm{OPT}_0(p^{(\ell)}) \leq \sum_{\ell=1}^{L} \mathrm{OPT}_0^{\mathrm{np}}(p^{(\ell)}) \leq \mathrm{OPT}_0^{\mathrm{np}}(p) \leq 1.81 \cdot \mathrm{OPT}_0(p)\,,$$

showing that the preemptive problem is 1.81-superadditive. □

Theorem 5.20 implies our main result for unrelated machines.

Theorem 5.27. *There is a polynomial-time non-clairvoyant online algorithm for minimizing the total weighted completion time on unrelated machines, $R \mid r_j, pmtn \mid \sum w_j C_j$, with a competitive ratio of at most* 4.62. *For uniform release dates, this bound reduces to* 3.62.

As discussed in Section 5.2, we loose a factor of $1+\varepsilon$ in the competitive ratio when requiring polynomial running time for PF. However, the precise constant proven by Sitters [Sit17] is actually the root of $8x^3 - 11x^2 - 4x - 4$, which is at most 1.806. Therefore, the stated bounds still hold.

Restricted Assignment. Sitters [Sit05] showed that for every instance of $R \mid \mathrm{pmtn}, s_{ij} \in \{0,1\} \mid \sum C_j$, there exists an optimal solution that is non-preemptive. Therefore, Lemma 5.25 implies that the problem is a 1-superadditive PSP. By Theorem 5.20, the PF algorithm is thus 2-competitive for uniform release dates and 3-competitive in general. As discussed in Section 5.4.2, PF can be executed in strongly polynomial time in this setting.

Theorem 5.28. *There is a strongly polynomial-time non-clairvoyant algorithm for minimizing the total completion time with a competitive ratio of* 2 *for restricted assignment, $R \mid pmtn, s_{ij} \in \{0,1\} \mid \sum C_j$. For non-uniform release dates, the competitive ratio is at most* 3.

Related Machines. We now consider the special case of PSP where all jobs are available at the beginning and have unit weight, and the speed of a machine i when processing a job j is independent of the job, that is, $s_{ij} = s_i$ for all $j \in [n]$. This is noted as $Q \mid \mathrm{pmtn} \mid \sum C_j$ in the 3-field notation. We assume without loss of generality that $s_1 \geq \ldots \geq s_m$, and $m \geq n$. The latter can be ensured by adding $n - m$ speed-zero machines.

Lemma 5.29. *The problem $Q \mid pmtn \mid \sum C_j$ is* 1*-superadditive.*

Proof. We first characterize the optimal objective value in terms of processing requirements $p_1 \leq \ldots \leq p_n$. We use that the PreemptiveSPT algorithm computes an optimal solution for this problem [Gon77; Lab+84a]. This algorithm runs at any time t the k shortest unfinished jobs on the k fastest machines, for any $1 \leq k \leq |J(t)|$.

It is not hard to see that PreemptiveSPT finishes the jobs in the order of their indices. Moreover, the resulting completion times C_j satisfy for every $1 \leq k \leq n$ that (using $C_0 := 0$)

$$\sum_{j=1}^{k} s_{k-j+1} \cdot (C_j - C_{j-1}) = p_k\,,$$

which yields by summation for every $1 \leq k \leq n$ that

$$\sum_{j=1}^{k} s_{k-j+1} \cdot C_j = \sum_{j=1}^{k} p_j \,.$$

This equation can be written as

$$\begin{pmatrix} s_1 & & & 0 \\ s_2 & s_1 & & \\ \vdots & & \ddots & \\ s_n & s_{n-1} & \dots & s_1 \end{pmatrix} \begin{pmatrix} C_1 \\ C_2 \\ \vdots \\ C_n \end{pmatrix} = \begin{pmatrix} 1 & & & 0 \\ 1 & 1 & & \\ \vdots & & \ddots & \\ 1 & 1 & \dots & 1 \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{pmatrix} . \tag{5.5}$$

Let λ be the solution to

$$1^\top = \lambda^\top \begin{pmatrix} s_1 & & & 0 \\ s_2 & s_1 & & \\ \vdots & & \ddots & \\ s_n & s_{n-1} & \dots & s_1 \end{pmatrix} .$$

Then, for every $i = n, \dots, 1$, we have $s_1\lambda_i = 1 - \sum_{k=i+1}^{n} s_{k-i+1}\lambda_k \geq 1 - \sum_{k=i+1}^{n} s_{k-i}\lambda_k = 0$. Therefore, μ defined by $\mu_k \coloneqq \sum_{i=k}^{n} \lambda_i$ for every $k \in [n]$ is ordered $\mu_1 \geq \dots \geq \mu_n$. Using this notation, by multiplying (5.5) from the left with $\lambda^\top$, we obtain

$$\mathrm{OPT}(p) = \sum_{k=1}^{n} C_k = \sum_{k=1}^{n} \mu_k \cdot p_k \,.$$

We now prove 1-superadditivity. Let $p = \sum_{\ell=1}^{L} p^{(\ell)}$ be an arbitrary partition of the processing requirements. For every $\ell \in [L]$ let $\sigma_\ell \colon [n] \to [n]$ be a permutation such that $p^{(\ell)}_{\sigma_\ell(1)} \leq \dots \leq p^{(\ell)}_{\sigma_\ell(n)}$. Since $\mu_1 \geq \dots \geq \mu_n$, we have that

$$\mathrm{OPT}(p^{(\ell)}) = \sum_{k=1}^{n} \mu_k \cdot p^{(\ell)}_{\sigma_\ell(k)} \leq \sum_{k=1}^{n} \mu_k \cdot p^{(\ell)}_k \,.$$

Therefore, we can conclude that the problem is 1-superadditive as follows:

$$\sum_{\ell=1}^{L} \mathrm{OPT}(p^{(\ell)}) \leq \sum_{\ell=1}^{L} \sum_{k=1}^{n} \mu_k \cdot p^{(\ell)}_k = \sum_{k=1}^{n} \mu_k \sum_{\ell=1}^{L} p^{(\ell)}_k = \sum_{k=1}^{n} \mu_k \cdot p_k = \mathrm{OPT}(p) \,. \qquad \square$$

Moreover, as discussed in Section 5.4.2, PF can be executed in strongly polynomial time for this problem.

Theorem 5.30. *There is a strongly polynomial-time non-clairvoyant algorithm for minimizing the total completion time with a competitive ratio equal to 2 on related machines, $Q \mid pmtn \mid \sum C_j$. For non-uniform release dates, the competitive ratio is at most 3.*

Parallel Identical Machines. For parallel identical machines, denoted by $P \mid \mathrm{pmtn} \mid \sum w_j C_j$, we recover a result by Beaumont et al. [Bea+12], who prove that their algorithm WDEQ, which

we show corresponds to PF in this setting (cf. Section 5.6), is 2-competitive. For completeness, we argue that this also falls out of our analysis. This follows from a nowadays folkloric result by McNaughton [McN59] who showed that for any instance of $P \mid \text{pmtn} \mid \sum w_j C_j$ there exists an optimal non-preemptive solution. Therefore, Lemma 5.25 implies that the problem is 1-superadditive, and thus, Theorem 5.20 gives that PF is 2-competitive.

Theorem 5.31. *There is a strongly polynomial-time non-clairvoyant online algorithm for minimizing the total weighted completion time with a competitive ratio equal to* 2 *on parallel identical machines,* $P \mid pmtn \mid \sum w_j C_j$. *For non-uniform release dates, the competitive ratio is at most* 3.

5.6 Combinatorial Implementation of PF for Related Machines

In this section, we present a combinatorial implementation of PF for the problem of minimizing the total weighted completion time on uniformly related machines, $Q \mid r_j, \text{pmtn} \mid \sum w_j C_j$, which runs in strongly polynomial time. For simplicity, we fix an arbitrary state at time t and assume that $J(t) = \{1, \dots, n\}$ and $m = n$. This can be achieved by adding speed-zero machines or removing the $m - n$ slowest machines, as this does not weaken an optimal solution.

We note that combining the results of Im et al. [IKM18] and Jain and Vazirani [JV10] already imply that PF can be implemented in strongly polynomial time (as it models a submodular utility allocation market). Our contribution is to provide a simpler and faster algorithm. Moreover, the description of our algorithm directly implies that, to the best of our knowledge, all known 2-competitive non-clairvoyant algorithms for uniform release dates are special cases of PF.

A Specialized Convex Program. Using the representation of the rate polytope for unrelated machine scheduling given in Section 5.2, we can see that the convex program (CP($J(t)$)) for this problem is equal to the following convex program (where the y-variables have been eliminated):

$$\begin{aligned}
\max \quad & \sum_{j=1}^{n} w_j \log\left(\sum_{i=1}^{n} s_i x_{ij}\right) && && (\text{CP}_Q)\\
\text{s.t.} \quad & \sum_{j=1}^{n} x_{ij} \leq 1 && \forall i \in [n]\\
& \sum_{i=1}^{n} x_{ij} \leq 1 && \forall j \in [n]\\
& x_{ij} \geq 0 && \forall i \in [n],\ j \in [n]
\end{aligned}$$

Algorithm 1: Solution algorithm for (CP_Q)

Input: Weights $w_1 \geq \ldots \geq w_n$, machine speeds $s_1 \geq \ldots \geq s_n$.

1 $k \leftarrow 1$

2 **while** $k \leq n$ **do**

3 Find the (largest) index h with $k \leq h \leq n$ that maximizes $\frac{\sum_{j=k}^{h} w_j}{\sum_{i=k}^{h} s_i}$.

4 Compute an allocation of jobs $k, \ldots, h$ to machines $k, \ldots, h$ such that every such job j receives a rate $y_j = \frac{w_j}{\sum_{j'=k}^{h} w_{j'}} \cdot \sum_{i=k}^{h} s_i$, as described in Lemma 5.33.

5 $k \leftarrow h + 1$

The KKT conditions [BV14] of an optimal solution $(x_{ij})_{i,j}$ of (CP_Q) with Lagrange multipliers $(\eta_i)_i$ and $(\delta_j)_j$ can be written as follows:

$$-\frac{s_i w_j}{\sum_{i'} s_{i'} x_{i'j}} + \eta_i + \delta_j \geq 0 \quad \forall i, j \in [n] \tag{5.6}$$

$$\eta_i \cdot \left(1 - \sum_{j=1}^{n} x_{ij}\right) = 0 \quad \forall i \in [n] \tag{5.7}$$

$$\delta_j \cdot \left(1 - \sum_{i=1}^{n} x_{ij}\right) = 0 \quad \forall j \in [n] \tag{5.8}$$

$$x_{ij} \cdot \left(-\frac{s_i w_j}{\sum_{i'} s_{i'} x_{i'j}} + \eta_i + \delta_j\right) = 0 \quad \forall i, j \in [n] \tag{5.9}$$

$$\eta_i, \delta_j \geq 0 \quad \forall i, j \in [n] \tag{5.10}$$

A Combinatorial Algorithm, Intuition, and Relation to Previous Results. We now introduce Algorithm 1. We will show below that this algorithm computes an optimal solution of (CP_Q), and thus, determines the instantaneous resource allocation x. We prove this by giving an assignment of Lagrangian multipliers that, together with x, satisfies the KKT conditions. This is sufficient, because (CP_Q) has a concave objective function and convex restrictions [BV14]. Executing this algorithm at every event time in the schedule leads to a combinatorial, strongly polynomial implementation of PF.

The main idea of Algorithm 1 is the following. If there exists an allocation x_{ij} and resulting processing rates y_j, $i, j \in [n]$, such that they fully utilize all machines, and all jobs j have the same ratio $w_j/y_j =: \pi$, then we can easily conclude via (5.6) that this allocation is optimal for (CP_Q) by setting $\eta_i := s_i \pi$ for all $i \in [n]$ and setting all other multipliers to 0. That means, it is optimal to distribute the processing rates proportional to the weights. However, this is not possible if the distribution of weights is very different to the distribution of speeds. In this case, we search for the largest prefix of $[h] \subseteq [n]$ (called *level*) such that a proportional allocation is possible for jobs and machines in $[h]$ (cf. Line 3). Then, we remove these machines and jobs from the instance (cf. Line 5) and repeat.

Consider, for example, identical machines: we would allocate job 1 entirely to machine 1 if $w_1/W > 1/m$, because its fraction of the total weight W is larger than the fraction of the speed

of a single machine compared to the total speed m. Note that is exactly how WDEQ handles high-weight jobs on identical parallel machines [Bea+12]. In fact, the instantaneous resource allocation computed by Algorithm 1 generalizes all known (to the best of our knowledge) non-clairvoyant algorithms for the simpler parallel identical machine environments and uniform release dates: WDEQ [Bea+12], WRR [KC03], and RR [MPT94].

We remark that Gupta et al. [Gup+12] mention another natural generalization of WRR to related machines, which allocates a w_j/W fraction of each of the fastest $\lfloor W/w_j \rfloor$ machines to job j. However, they show that this algorithm is at least $\Omega(\sqrt{\log n})$-competitive for the total weighted completion time objective, whereas we prove that PF is 4-competitive in that case (Theorem 5.15).

Feldman et al. [Fel+08] considered Algorithm 1 as a mechanism to fractionally allocate n ad slots i with different click rates s_i to n advertisers j. Each advertiser j wishes to maximize their received click rate y_j and has a budget w_j they are willing to pay. To this end, they can bid an amount w'_j. The authors showed that when the slots are allocated according to Algorithm 1 applied to the bids w'_j and each advertiser is charged their own bid, then bidding the true budget w_j is a dominant strategy, that is, the mechanism is strategy-proof. Therefore, since all bids are collected by the auctioneer, it maximizes the revenue of the auctioneer under strategic behavior of the advertisers. Our result provides further insight about this setting. Since the works of Im et al. [IKM18] and Jain and Vazirani [JV10] imply that a solution to (CP_Q) is equal to a market equilibrium, our result shows that the mechanism actually computes a market equilibrium. That means that it does not have to enforce the allocation, but only needs to set the prices for ad rate in the different slots. For these prices the allocation maximizes each advertisers' click rate subject to their budget constraint, so that it is expected to arise without central coordination.

Proof of Equivalence. We finally prove the main theorem of this section.

Theorem 5.32. *Algorithm 1 computes an optimal solution of* (CP_Q).

Note that this theorem also implies that the above-mentioned non-clairvoyant algorithms are all special cases of PF.

The remaining section is dedicated to the proof of Theorem 5.32. Let $L_1, \dots, L_r$ be the partition of $1, \dots, n$ produced by the iterations of Algorithm 1. We call each $1 \le \ell \le r$ a *level* and denote by

$$\pi(L_\ell) = \frac{\sum_{j \in L_\ell} w_j}{\sum_{i \in L_\ell} s_i}$$

the *price* of level ℓ. Further, we write $\pi_j = \pi(L_\ell)$ for every $j \in L_\ell$. Note that for every job j it holds that $\pi_j = w_j/y_j$.

Lemma 5.33 ([Fel+08])**.** *The allocation in Line 4 always exists, and can be computed efficiently.*

Proof. Consider a level $L = \{k, \dots, h\}$ with price $\pi := \pi(L)$ computed in Line 3. Thus, for every $k \le h' \le h$ it holds that $\pi \sum_{i=k}^{h'} s_i \ge \sum_{j=k}^{h'} w_j$. Since $\pi = w_j/y_j$ for every $j \in L$, we conclude that for every $k \le h' \le h$ we have

$$\sum_{i=k}^{h'} s_i \ge \sum_{j=k}^{h'} \frac{w_j}{\pi} = \sum_{j=k}^{h'} y_j \ .$$

Horvath et al. [HLS77] show that these conditions suffice to ensure that jobs of length $y_k, \dots, y_h$ can be preemptively scheduling on machines with speeds $s_k, \dots, s_h$ within a makespan of 1. Moreover, such a schedule can be computed efficiently using the *level algorithm*. This concludes the proof of the lemma. □

Gonzalez and Sahni [GS78] give an even faster algorithm to compute the allocations in Line 4. We can derive two immediate observations from the description of Algorithm 1.

Observation 5.34. *If $j \in L_\ell$, then $x_{ij} = 0$ for all $i \in [n] \setminus L_\ell$.*

Observation 5.35. *It holds that $\pi_1 \ge \dots \ge \pi_n$.*

Finally, we prove Theorem 5.32.

Proof. Let x_{ij}, $i, j \in [n]$, be the allocation computed by Algorithm 1. Since (CP_Q) has a concave objective function and convex restrictions, we show via the sufficient KKT condition for convex programs that this allocation is an optimal solution to (CP_Q) [BV14]. To this end, we present Lagrange multipliers such that the KKT conditions are satisfied: for every machine $i \in [n]$ we define

$$\eta_i := \pi_n s_n + \sum_{k=i}^{n-1} \pi_k \left(s_k - s_{k+1}\right) \ ,$$

and for every job $j \in [n]$ we define

$$\delta_j := \pi_j s_j - \eta_j \ .$$

We first verify (5.6). Fix a job $j \in [n]$ and a machine $i \in [n]$. We distinguish two cases.

Case $i \le j$. Using Observation 5.35, we have

$$\begin{aligned} \frac{w_j \cdot s_i}{y_j} = \pi_j s_i = \pi_j \left(s_j + \sum_{k=i}^{j-1} (s_k - s_{k+1}) \right) &\le \pi_j s_j + \sum_{k=i}^{j-1} \pi_k (s_k - s_{k+1}) \\ &= \pi_j s_j + \eta_i - \eta_j = \eta_i + \delta_j \ . \end{aligned} \tag{5.11}$$

Case $i > j$. Using Observation 5.35, we have

$$\begin{aligned} \frac{w_j \cdot s_i}{y_j} = \pi_j s_i = \pi_j \left(s_j + \sum_{k=j}^{i-1} (s_{k+1} - s_k) \right) &\le \pi_j s_j + \sum_{k=j}^{i-1} \pi_k (s_{k+1} - s_k) \\ &= \pi_j s_j - (\eta_j - \eta_i) = \eta_i + \delta_j \ . \end{aligned}$$

Next, we verify (5.7) to (5.9). Since $n = m$ and by the definition of the algorithm, it is clear that every job and every machine is fully allocated, hence (5.7) and (5.8) follow. For (5.9), note that this follows from Observation 5.34 whenever i and j are in different levels. Otherwise, if $i, j \in L_\ell$, the inequality in (5.11) holds with equation because in that case $\pi_j = \pi(L_\ell) = \pi_k$ for $k = i, \dots, j$.

We finally check the non-negativity (5.10). Note that it suffices to show that $0 \leq \eta_i \leq \pi_i s_i$. To see this, observe that for every machine i we have

$$\eta_i = \pi_n s_n + \sum_{k=i}^{n-1} \pi_k \underbrace{(s_k - s_{k+1})}_{\geq 0} \geq 0 \,,$$

and, by using Observation 5.35, we have

$$\eta_i = \pi_n s_n + \sum_{k=i}^{n-1} \pi_k \, (s_k - s_{k+1}) = \pi_i s_i + \sum_{k=i+1}^{n} s_k \underbrace{(\pi_k - \pi_{k-1})}_{\leq 0} \leq \pi_i s_i \,.$$

This completes the proof. □

Algorithm 1 can be executed in time $O(n^2)$. Since we do this at every release and completion time, the total running time is in $O(n^3)$.

5.7 Tight Dual Fitting for Weighted-Round-Robin

In this section we present a tight analysis of the Weighted-Round-Robin algorithm (WRR) for $1 \,|\, \text{pmtn} \,|\, \sum w_j C_j$ via dual fitting. Let $[n]$ be the set of jobs. We assume that $w_1/p_1 \geq \dots \geq w_n/p_n$. Then, WRR processes at any time t every unfinished job $j \in U(t)$ at rate $y_j(t) = w_j / \sum_{j' \in U(t)} w_{j'}$. Observe that in WRR's schedule, job 1 completes at time $C_1 = p_1/y_1(0) = (\sum_{k=1}^{n} w_k) \cdot p_1/w_1$, job 2 completes at time

$$C_2 = C_1 + \frac{p_2 - C_1 \cdot y_2(0)}{y_2(C_1)} = \frac{p_1}{w_1}\left(\sum_{k=1}^{n} w_k\right)\left(1 - \frac{\sum_{k=2}^{n} w_k}{\sum_{k=1}^{n} w_k}\right) + \frac{p_2}{w_2}\sum_{k=2}^{n} w_k = p_1 + \frac{p_2}{w_2}\sum_{k=2}^{n} w_k \,,$$

and so on. In general, the completion time of job j in WRR's schedule is equal to

$$C_j = \sum_{k=1}^{j-1} p_k + \frac{p_j}{w_j}\sum_{k=j}^{n} w_k \,.$$

We prove the following theorem.

Theorem 5.36. *The total weighted completion time of Weighted Round-Robin is equal to twice the optimal objective value of* (LP(1)) *for uniform release dates.*

It is known that the total weighted completion time of a schedule produced by WRR is equal to twice the optimal total weighted mean busy time [Jäg+23; KC03]. Since the objective function

of (LP(1)) corresponds to the total weighted mean busy time, the statement of Theorem 5.36 already follows.

Our contribution is to show how an optimal solution to (DLP(1)) can be expressed in terms of WRR's schedule. This suggests that we probably need a substantially more complex dual setup and sufficient knowledge on optimal LP solutions to improve over the factor of 4, which can be achieved by the simple dual setup (cf. Section 5.4). Specifically, in order to craft an optimal dual solution, it is necessary (due to complementary slackness) to respect that scheduling jobs in WSPT order minimizes the total weighted mean busy time on a single machine [Goe96], and thus, optimally solves (LP(1)).

Let $T = \sum_{j=1}^{n} p_j$ be the makespan of any non-idling schedule. For convenience, we restate (DLP(1)) for the special case of single-machine scheduling:

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} a_j - \sum_{t=0}^{T-1} b_t && && \text{(DLP(1))} \\ \text{s.t.} \quad & \frac{a_j}{p_j} - \frac{w_j}{p_j}\left(t + \frac{1}{2}\right) \leq b_t && \forall j \in [n],\ \forall t \in \{0, \dots, T-1\} \\ & a_j, b_t \geq 0 && \forall j \in [n],\ \forall t \in \{0, \dots, T-1\} \end{aligned}$$

For the analysis we assume by scaling that all processing requirements and weights are integers. Define $q_j := \sum_{k=1}^{j} p_k$ for every job j and set $q_0 := 0$. Note that $q_j - q_{j-1} = p_j$. We define the following dual assignment:

- $\bar{a}_j := w_j C_j$ for every job $j \in [n]$, and
- for every $t \in \{0, \dots, T-1\}$ we set $\bar{b}_t := \frac{w_i}{p_i}(C_i - (t + \frac{1}{2}))$ where $q_{i-1} \leq t < q_i$.

Clearly, $\sum_j \bar{a}_j = \text{ALG}$. Combined with the following lemma, we conclude that the dual objective value of our dual assignment is equal to $\frac{1}{2} \cdot \text{ALG}$.

Lemma 5.37. *It holds that* $\sum_{t=0}^{T-1} \bar{b}_t = \frac{1}{2} \cdot \text{ALG}$.

Proof. Using the above formula for a job's completion time in WRR's schedule, we have

$$\begin{aligned} \sum_{t=0}^{T-1} \bar{b}_t &= \sum_{j=1}^{n} \sum_{t=q_{j-1}}^{q_j-1} \frac{w_i}{p_i}\left(C_i - \left(t + \frac{1}{2}\right)\right) = \sum_{j=1}^{n} \sum_{t=q_{j-1}}^{q_j-1} \sum_{k=j}^{n} w_k - \frac{w_j}{p_j}\left(t + \frac{1}{2} - \sum_{k=1}^{j-1} p_k\right) \\ &= \sum_{j=1}^{n}\left(p_j \sum_{k=j}^{n} w_k\right) - \frac{w_j}{p_j} \sum_{t=0}^{p_j-1}\left(t + \frac{1}{2}\right) = \sum_{j=1}^{n}\left(p_j \sum_{k=j}^{n} w_k\right) - \frac{w_j}{p_j} \cdot \left(\frac{p_j(p_j-1)}{2} + \frac{p_j}{2}\right) \\ &= \sum_{j=1}^{n} p_j\left(\frac{w_j}{2} + \sum_{k=j+1}^{n} w_k\right) = \frac{1}{2}\sum_{j=1}^{n} p_j\left(\sum_{k=j}^{n} w_k + \sum_{k=j+1}^{n} w_k\right) \\ &= \frac{1}{2}\sum_{j=1}^{n} p_j\left(\sum_{k=j}^{n} w_k\right) + w_j \sum_{k=1}^{j-1} p_k = \frac{1}{2}\sum_{j=1}^{n} w_j\left(\sum_{k=1}^{j-1} p_k + \frac{p_j}{w_j}\sum_{k=j}^{n} w_k\right) = \frac{1}{2}\text{ALG}\,. \end{aligned}$$

This completes the proof. □

Lemma 5.38. *The dual solution* $(\bar{a}_j)_j$ *and* $(\bar{b}_t)_t$ *is feasible for* (DLP(1)).

Proof. First observe that we can rewrite

$$\bar{b}_t = \frac{w_i}{p_i}\left(C_i - \left(t + \frac{1}{2}\right)\right) = \sum_{k=i}^{n} w_k - \frac{w_i}{p_i}\left(t + \frac{1}{2} - \sum_{k=1}^{i-1} p_k\right).$$

Clearly, $\bar{a}_j \geq 0$ for every $j \in [n]$. To see that $\bar{b}_t \geq 0$, note that in WRR's schedule the jobs complete in order of their index, and thus, job j cannot complete earlier than $q_j = \sum_{k=1}^{j} p_k$. It remains to prove that for every $j \in [n]$ and $q_{i-1} \leq t \leq q_i - 1$ that the first constraint of (DLP(1)) is satisfied. Let $\tau \in [0, 1)$ such that $t + \frac{1}{2} = q_{i-1} + \tau \cdot p_i$. We distinguish two cases.

If $j \leq i$, the left side of the dual constraint is equal to

$$\begin{aligned}
\bar{a}_j - w_j\left(t + \frac{1}{2}\right) &= w_j \sum_{k=1}^{j-1} p_k + p_j \sum_{k=j}^{n} w_k - w_j\left(t + \frac{1}{2}\right) \\
&= w_j \sum_{k=1}^{j-1} p_k + p_j \sum_{k=j}^{n} w_k - w_j \sum_{k=1}^{i-1} p_k - \tau \cdot w_j \cdot p_i \\
&= -w_j \sum_{k=j}^{i-1} p_k + p_j \sum_{k=j}^{n} w_k - \tau \cdot w_j \cdot p_i \\
&= -w_j \sum_{k=j}^{i-1} p_k + p_j \sum_{k=j}^{i-1} w_k + p_j \sum_{k=i}^{n} w_k - \tau \cdot w_j \cdot p_i \,.
\end{aligned}$$

Using the fact that $w_j p_k \geq w_k p_j$ for every $k \geq j$, we can bound this expression from above by

$$\begin{aligned}
&- w_j \sum_{k=j}^{i-1} p_k + w_j \sum_{k=j}^{i-1} p_k + p_j \sum_{k=i}^{n} w_k - \tau \cdot w_i \cdot p_j = \left(\sum_{k=i}^{n} w_k - \tau \cdot w_i\right) \cdot p_j \\
&= \left(\sum_{k=i}^{n} w_k - \frac{w_i}{p_i}(\tau \cdot p_i)\right) \cdot p_j = \left(\sum_{k=i}^{n} w_k - \frac{w_i}{p_i}\left(t + \frac{1}{2} - \sum_{k=1}^{i-1} p_k\right)\right) \cdot p_j = \bar{b}_t \cdot p_j \,,
\end{aligned}$$

giving the right side of the constraint.

Similarly, we have for the case $j > i$ that

$$\begin{aligned}
\bar{a}_j - w_j\left(t + \frac{1}{2}\right) &= w_j \sum_{k=1}^{j-1} p_k + p_j \sum_{k=j}^{n} w_k - w_j\left(t + \frac{1}{2}\right) \\
&= w_j \sum_{k=1}^{j-1} p_k + p_j \sum_{k=j}^{n} w_k - w_j \sum_{k=1}^{i-1} p_k - \tau \cdot w_j \cdot p_i \\
&= w_j \sum_{k=i}^{j-1} p_k + p_j \sum_{k=j}^{n} w_k - \tau \cdot w_j \cdot p_i \\
&= w_j \sum_{k=i+1}^{j-1} p_k - p_j \sum_{k=i}^{j-1} w_k + p_j \sum_{k=i}^{n} w_k + (1 - \tau) \cdot w_j \cdot p_i \,.
\end{aligned}$$

Using the fact that $w_k p_j \geq w_j p_k$ for every $k < j$, we can bound this expression from above by

$$p_j \sum_{k=i+1}^{j-1} w_k - p_j \sum_{k=i}^{j-1} w_k + p_j \sum_{k=i}^{n} w_k + (1-\tau) \cdot w_i \cdot p_j = p_j \sum_{k=i}^{n} w_k - \tau \cdot w_i \cdot p_j$$
$$= \left(\sum_{k=i}^{n} w_k - \frac{w_i}{p_i}(\tau \cdot p_i) \right) \cdot p_j = \left(\sum_{k=i}^{n} w_k - \frac{w_i}{p_i}\left(t + \frac{1}{2} - \sum_{k=1}^{i-1} p_k\right) \right) \cdot p_j = \bar{b}_t \cdot p_j \,,$$

which concludes that our dual solution is feasible. This finishes the proof. □

Proof of Theorem 5.36. Let LP^* denote the optimal objective value of (LP(1)). Then, weak duality, Lemma 5.37, and Lemma 5.38 imply

$$\mathrm{LP}^* \geq \sum_{j=1}^{n} \bar{a}_j - \sum_{t=0}^{T-1} \bar{b}_t = \frac{1}{2} \cdot \mathrm{ALG} \,.$$

Moreover, the proof of Lemma 5.38 reveals via complementary slackness and strong duality that our dual solution is even optimal for (DLP(1)), and thus, $\mathrm{LP}^* = \sum_{j=1}^{n} \bar{a}_j - \sum_{t=0}^{T-1} \bar{b}_t$. This is because the scheduling the jobs in the order of their index is optimal for (LP(1)) [Goe96]. □

5.8 Lower Bound for PF with Non-Uniform Release Dates

In this section, we present a lower bound on the competitive ratio of PF for PSP with non-uniform release dates that is strictly larger than 2. We prove it for the special case of scheduling on a single machine, where PF reduces to RR (cf. Section 5.6). RR processes at any time t every available job $j \in J(t)$ at rate $1/|J(t)|$. To prove a lower bound on RR's competitive ratio, we compare RR to the optimal solution, which is computed by the Shortest Remaining Processing Time rule (SRPT) [Sch68]. This strategy processes at any time the available job of shortest remaining processing time. We first present a weaker but simpler lower bound, and then give an improved lower bound of at least 2.19.

Theorem 5.39. *The competitive ratio of RR (resp. PF) is at least* 2.074 *for* $1 \mid r_j, pmtn \mid \sum C_j$.

Proof. Consider the following instance. There are n jobs J_1 released at time 0 with processing time 1 and n jobs J_2 released at time $r = (2-\sqrt{3})n$ with processing time $\sqrt{3}-1$. The number of jobs in J_1 and J_2 is equal to n each, and since we assume that $n \to \infty$, we can without loss of generality scale n to 1, that is, the jobs are sand. Note that, in RR, the jobs in J_1 have at time r the same remaining processing time as the jobs in J_2. Thus, all jobs complete at time $\sqrt{3}$, and the total completion time of RR is equal to $2\sqrt{3}$. SRPT first sequentially schedules an r fraction of the jobs in J_1 until time r, then sequentially completes all jobs in J_2 and finally completes the remaining $1-r$ volume of J_1. In this schedule, the total completion time of the jobs completed until time r is equal to $\frac{1}{2}r^2$, the total completion time of jobs in J_2 equal to $r + \frac{1}{2}(\sqrt{3}-1)$, and the total completion time of the remaining jobs of J_1 equal to $(1-r)(r+\sqrt{3}-1) + \frac{1}{2}(1-r)^2$. Hence,

$$\mathrm{OPT} \leq \frac{1}{2}r^2 + r + \frac{1}{2}(\sqrt{3}-1) + (1-r)(r+\sqrt{3}-1) + \frac{1}{2}(1-r)^2 = 6 - \frac{5}{2}\sqrt{3} \,.$$

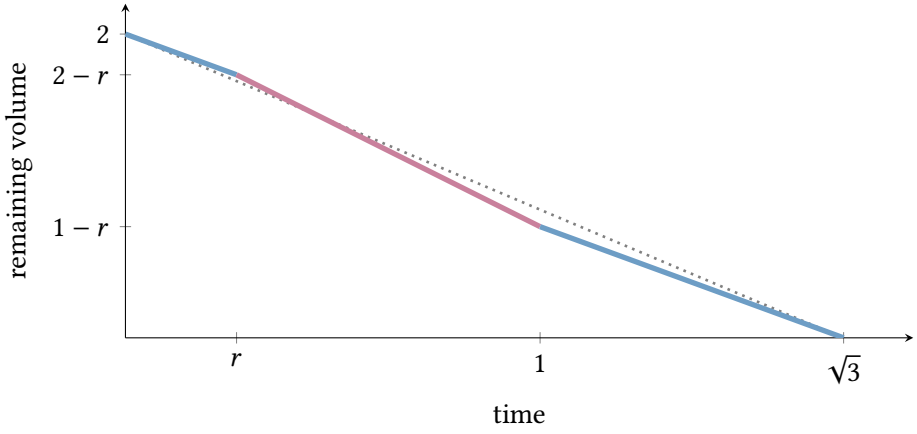


Figure 5.1: The total completion time of both schedules used in Theorem 5.39. The solid lines indicate the completion of jobs in the SRPT schedule; the area below is equal to the total completion time of the SRPT schedule. The colors of the solid line indicate which jobs are processed at which time. The total area below the dotted line is equal to half of the total completion time of RR. Observe that the area under the solid line is (slightly) less than the area under the dotted line.

Therefore, the competitive ratio of RR is at least $\frac{2\sqrt{3}}{6-\frac{5}{2}\sqrt{3}} = \frac{4}{23}(5+4\sqrt{3}) > 2.074$. □

We strengthen this example by adding more release dates, which improves the lower bound on the competitive ratio. We consider an instance with k release dates, and at every release date (including $r_1 = 0$) the same number of jobs are released. We again assume that the jobs are sand, hence scale their number to 1. Let $J_1, \dots, J_k$ be those sets, and let $p_1 > \cdots > p_k$ be their processing times. Our goal is to construct an instance in which all jobs complete at the same time in the schedule of RR with a total completion time equal to $k \sum_{i=1}^{k} p_i$. Thus, given values for $p_1, \dots, p_k$, we can observe that the release dates $r_2, \dots, r_k$ must satisfy that at every time r_i all jobs of $J_1, \dots, J_i$ have the same remaining processing time. This requires that

$$r_i = r_{i-1} + p_{i-1} - p_i$$

for all $i = 2, \dots, k$.

We now consider the solution of SRPT for this instance. Let V_j be the fraction of jobs J_j that is being processed before the release of the jobs J_{j+1} in the SRPT schedule, and let $V_k = 1$. Note that $V_j = (r_{j+1} - r_j)/p_j$ for all $j = 1, \dots, k-1$. Thus, the total completion time of SRPT can be expressed as follows (using $r_{k+1} = r_k + p_k$):

$$\sum_{j=1}^{k} (r_{j+1} - r_j)\left(k - \frac{1}{2}V_j - \sum_{i=1}^{j-1} V_i\right) + \sum_{j=1}^{k-1} (p_j - (r_{j+1} - r_j))\left(\frac{1}{2}(1 - V_j) + \sum_{i=1}^{j-1}(1 - V_i)\right).$$

We numerically maximize the ratio between RR's objective and this expression over all $1 = p_1 > \ldots > p_k$ for $k = 30$ and derive the following improved lower bound. The processing times are presented in Table 5.3.

Theorem 5.40. *The competitive ratio of RR (resp. PF) is at least* 2.1906 *for* $1 \mid r_j, pmtn \mid \sum C_j$.

Table 5.3: The (approximate) processing times used for the proof of Theorem 5.40.

j	p_j	j	p_j	j	p_j	j	p_j	j	p_j
1	1	7	0.75885	13	0.59832	19	0.48282	25	0.39510
2	0.95160	8	0.72782	14	0.57656	20	0.46659	26	0.38245
3	0.90702	9	0.69872	15	0.55590	21	0.45106	27	0.37027
4	0.86581	10	0.67137	16	0.53628	22	0.43619	28	0.35854
5	0.82759	11	0.64561	17	0.51760	23	0.42194	29	0.34722
6	0.79203	12	0.62131	18	0.49980	24	0.40825	30	0.33630

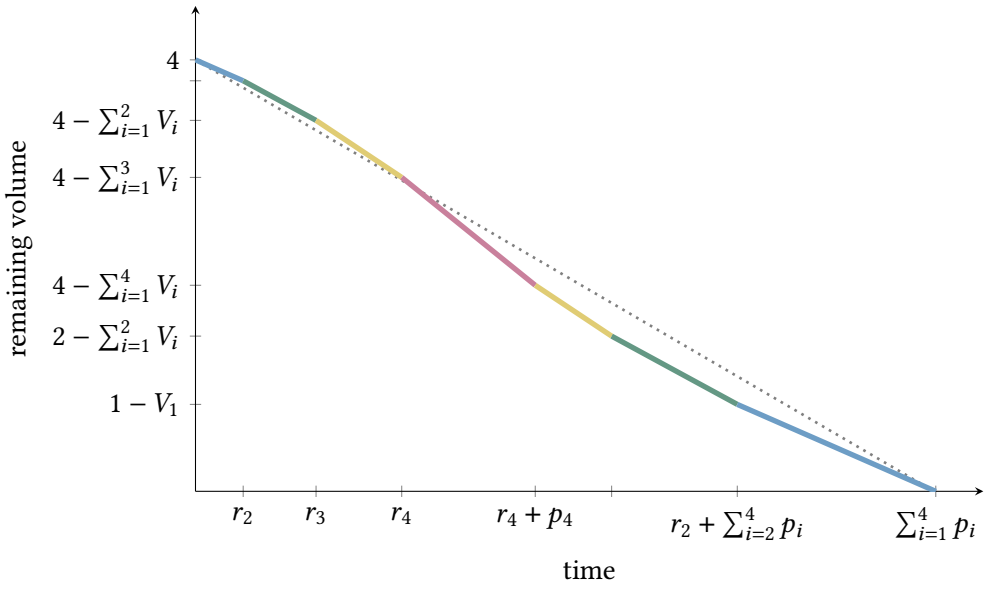


Figure 5.2: The total completion time of both schedules in the improved construction with $k = 4$. Observe that ratio between the area under the solid line and the area under the dotted line is smaller than the corresponding ratio in Figure 5.1. Here, $p_1 = 1$, $p_2 \approx 0.80632$, $p_3 \approx 0.65835$, and $p_4 \approx 0.54231$.

5.9 Concluding Remarks

For weighted jobs and unrelated machines, there remains the gap between 3.62 (Theorem 5.27) and the lower bound of 2. We remark that improving the bound on the power of preemption of 1.81 [Sit17] directly implies an improved bound on the competitive ratio by Lemma 5.26 and Theorem 5.20. This cannot, however, completely close the gap to 2, as there is a known lower bound of 1.39 on the power of preemption [Eps+17]. In contrast, our analysis established a tight competitive ratio for related machines, despite the fact that an optimal solution migrates jobs. We conjecture that the competitive ratio of PF is exactly 2 on unrelated machines.

Another well-studied objective is to minimize the total weighted flow time. Im et al. [IKM18] showed that PF is $O(1/\varepsilon^2)$-competitive for MonPSP if the given resource capacity per time unit is increased by a factor $(e + \varepsilon)$, for any $\varepsilon \in (0, 1/2)$. This is called speed augmentation. For the special case of related machines and equal weights, Gupta et al. [GKP10] showed that PF is $(2 + \varepsilon)$-speed $O(1)$-competitive, which is tight for PF even on a single machine [KP00].

Our improved understanding of PF (cf. Algorithm 1) might help to close this gap also for the weighted setting. We note that a different and arguably less natural generalization of RR to (un-)related machines achieves this tight result [Im+14]. Moreover, it is open whether a combination between PF and a technique for *scalable* algorithms ($(1+\varepsilon)$-speed $O(1)$-competitive algorithms), such as LAPS [EP12] or its smoothed variant [Im+14], yields a scalable algorithm for unrelated machines or MonPSP. While for unrelated machines a tailored and rather complex algorithm achieves this [Im+14], it is known that for the general PSP no algorithm can be constant competitive with constant speed augmentation [IKM18].

An intermediate model between clairvoyant and non-clairvoyant scheduling is stochastic scheduling, where job size distributions become known upon job release. Our bounds for the non-clairvoyant PF achieve the best known performance guarantees for polynomial-time preemptive stochastic scheduling policies. This raises the question of whether policies could benefit from stochastic information, as it is on a single machine $1 \mid \text{pmtn} \mid \sum w_j C_j$ [Sev74].

Bibliographic Note

This chapter is based on joint work with Sven Jäger and Nicole Megow [JLM25]. Thus, some parts of this chapter are identical with [JLM25].

Part III

Learning-Augmented Scheduling

Chapter 6

Introduction to Learning-Augmented Algorithms

6.1 Motivation and Introduction

A cornerstone in the theoretical study of algorithms is the analysis of performance characteristics such as running time, approximation ratios, or competitive ratios in the *worst case*. This approach has been very successful, as it provides a simple and clear basis for evaluating algorithms, as we have seen in the previous two parts of this thesis. Moreover, in many safety-critical applications, a good understanding of worst-case behavior is unavoidable. Worst-case analysis also has obvious issues: if an algorithm performs badly only on a unique *worst-case instance*, but optimally on all others, worst-case guarantees would rank this algorithm below a medium-performing algorithm. In practice, where such a worst-case instance is very unlikely to occur, one clearly wants to favor the algorithm that performs better in practice, although it might have a poorer worst-case guarantee.

To address this issue, various techniques were proposed to analyze algorithms beyond the worst case [Rou20]. Learning-augmentation is such an approach, which is motivated by the recent success of machine learning. The key idea is that a machine-learned model (or any predictor) can predict characteristics of an upcoming instance, which, if somewhat precise, may help an algorithm to improve its worst-case behavior. The challenging aspect of this approach is the "somewhat precise" part, namely the big issue that a machine-learned model cannot guarantee for sure that the prediction is accurate. Indeed, precisely predicting the future seems to be a utopia.

The *learning-augmented* framework (also called *algorithms with predictions*) combines the best parts of worst-case analysis and machine-learned predictions by seeking for the following properties:

Consistency: When the prediction is accurate, we want to achieve the same performance as an algorithm that has access to the ground truth.

Robustness: When the predictions are bad, we want to achieve a good worst-case performance.

Smoothness: The performance of the algorithm should degrade gracefully between the consistent and the robust case depending on the *quality* of the prediction.

The quality of a prediction is measured by some *prediction error*, which is a real-valued function that takes as input the problem instance and the prediction. It describes how well the prediction

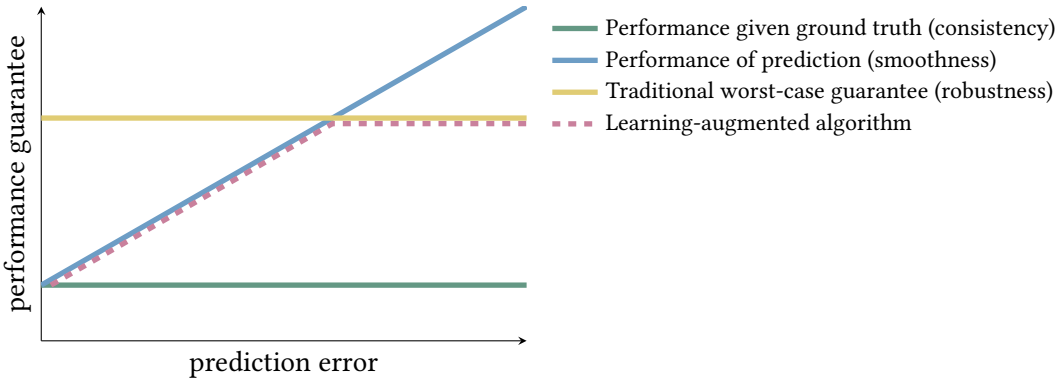


Figure 6.1: Visualization of the main concepts of learning-augmented algorithms. The performance of a learning-augmented algorithm is consistent with the performance of an algorithm given the ground truth when the prediction error is 0, it degrades smoothly with respect to the quality of the prediction, and it is robust against arbitrarily bad predictions.

matches the ground truth. We usually assume that a learning-augmented algorithm does not know the prediction error upfront. Therefore, blindly following the prediction can result in an arbitrarily bad performance from which an algorithm cannot recover after potentially noticing that the prediction error was large. Thus, one difficulty in learning-augmented algorithm design is to find the level of trust into the predictions that guarantees consistency, smoothness, and robustness. An illustration of these ideas is given in Figure 6.1. For introductory examples of this framework, we refer the reader to articles by Mitzenmacher and Vassilvitskii [MV20; MV22a].

In this final part of the thesis, we apply this framework to online scheduling problems. Online problems are a very natural application of the learning-augmented framework, as algorithms have to cope against any eventuality of the uncertain future. Fixing a prediction and a value of the prediction error upfront, intuitively restricts the future to instances that are close to the prediction with respect to the prediction error. Thus, we can potentially avoid arbitrarily bad worst-case instances.

We say that an online algorithm is α-*consistent* if it is α-competitive whenever it receives correct predictions, and α-*robust* if it is α-competitive for any prediction. The consistency ratio (robustness ratio) is the smallest α such that the algorithm is α-consistent (α-robust).

6.2 Classification of Prediction Models for Online Problems

When applying the learning-augmented framework to a problem, an important decision is to choose a specific prediction model, that is, which information an algorithm should additionally receive. Most of the prediction models used so far for online problems can be classified into two categories: *input predictions* and *action predictions*. Input predictions give information about the uncertain online input. An important and commonly used subclass are *full input predictions*, which predict the whole instance. Action predictions are independent of the online input and rather give suggestions on algorithmic actions, that is, which action to execute next.

Depending on the problem, action predictions can also be *adaptive*, that is, they arrive online with the input at the time when the algorithm has to make a decision.

Examples for works using full input predictions for online problems are [AJS22; ALT21; ALT22; Ber+22a; Erl+22; Im+23; PSK18]. Action predictions are used in, for example, [Ana+22; Ant+20; BMS20; Ebe+22; JM22; LM22; LMS22].

6.3 Related Work

General Overview. The concepts of consistency, smoothness, and robustness were first formalized by Lykouris and Vassilvitskii [LV18; LV21] for the online caching problem. The competitive ratio for the online caching problem is $\Theta(\log k)$ for a cache of size k, while it can be solved optimally if the sequence of cache requests is known upfront. Lykouris and Vassilvitskii presented an algorithm that achieves at the same time a competitive ratio of at most $O(\log k)$, giving robustness, and of at most $O(\sqrt{\eta/\mathrm{OPT}})$, giving consistency and smoothness, where η denotes the prediction error. There are some prior works that used predictions to improve the performance of algorithms [Kra+18; MNS12; MV17; RS13]. Their models, however, do not completely fit into the above framework, usually because they miss either robustness or smoothness. We refer to [LV21] to a comprehensive overview of prior related work.

Subsequently, over 200 works using this framework have been published until mid of 2024, establishing learning-augmented algorithms as an important subarea of algorithm design and beyond worst-case analysis. We refer to the online repository for a nearly complete list of papers [LM24] in this area.

Some arguably early important results besides [LV18; LV21] are by Purohit et al. [PSK18] on online ski-rental and non-clairvoyant scheduling, Lattanzi et al. [Lat+20] on online makespan minimization, Antoniadis et al. [Ant+23b] on online metrical task systems, Bamas et al. [BMS20] on online primal-dual algorithms, or Dinitz et al. [Din+21] on warm-starting matching algorithms with predictions.

While the majority of works consider online problems, predictions have also been successful for improving running times (for example, [BC23; Che+22; Dav+23; Din+21]), data structures and dynamic algorithms (for example, [Bra+24; Hen+24; LLW22; McC+23]), approximation algorithms ([Coh+24]), or performance indicators of mechanisms (for example, [Agr+22; BGT23; Gka+22; XL22]).

Learning-Augmented Scheduling. We give more details on related work on scheduling problems in the learning-augmented framework.

Many learning-augmented results on scheduling address non-clairvoyance. Purohit et al. [PSK18] initially studied this problem on a single machine to minimize the total completion time by predicting unknown job lengths. They showed that having access to sufficiently good predictions bypasses the classic lower bound on the competitive ratio. Several works subsequently studied this problem with full input prediction, also in variations [Bam+22; BP23; BP24; Eli+24; Im+23; WZ20]. The author of this thesis and Megow [LM22] proposed a different prediction model using action predictions for this problem (see Chapter 7), which has also been picked up in subsequent works [Din+22; Eli+24; Kho+22]. Moreover, non-

clairvoyant scheduling with input predictions on parallel machines to minimize the makespan was considered [Bam+23c; ZLZ22b]. Finally, non-clairvoyant scheduling to minimize the total (weighted) flow time with input predictions was studied on a single machine [ALT21; ALT22; ZLZ22a] and on multiple machines [APT22b].

Predictions were also used to improve the competitive ratio of online problems with known processing times. Lattanzi et al. [Lat+20] started this line of research by presenting a learning-augmented algorithm for the online makespan minimization problem for restricted assignment. Later, Lavastida et al. [Lav+21] extended their result, and Li and Xian [LX21] improved and generalized it to unrelated machines. Cohen and Panigrahi [CP23] then further improved these results in a more general online allocation framework.

Other scheduling problem that have been studied are online speed scaling [AJS22; Bal+23c; Bam+20], speed-robust scheduling [Bal+23b], or interval scheduling [Boy+23].

Chapter 7

Permutation Predictions for Unknown Processing Requirements

7.1 Introduction

In this chapter, we study non-clairvoyant scheduling in a learning-augmented setting. More specifically, we consider the problem of scheduling jobs on a single or parallel identical machines to minimize the total weighted completion time. As already mentioned in Chapter 5, no non-clairvoyant algorithm can have a competitive ratio better than 2 for this problem [MPT94]. Moreover, this optimal competitive ratio is achieved by Proportional Fairness (cf. Chapter 5) on parallel identical machines with uniform release dates [Bea+12; KC03; MPT94].

Purohit et al. [PSK18] initiated the study of learning-augmented algorithms for this problem on a single machine, and gave the first results of competitive ratios better than 2. They assume that for every job j there is given a prediction $\hat{p}_j$ on its actual unknown length p_j. We call this model in this context *length predictions*, and note that it belongs to the class of full input predictions. This prediction model has been picked up in later works on non-clairvoyant scheduling, also for other objectives [ALT21; ALT22; APT22b; Bam+22; Bam+23c; Im+23; WZ20].

7.1.1 Our Results

We introduce a new prediction model for scheduling to minimize the total weighted completion time. Our prediction model is heavily inspired by the relevance of the WSPT order (cf. Theorem 2.1), the order of non-increasing weight to processing time ratios, for this problem on a single machine. For a scheduling instance with job set $J = [n]$, weights $w = (w_j)_j$ and processing times $p = (p_j)_j$, our prediction is a *permutation* $\hat{\sigma} : [n] \to [n]$ of all jobs. We call a WSPT order σ on the same job set a *perfect prediction*. Thus, permutation predictions extract the essential information required to schedule jobs (near) optimally.

In the permutation prediction model, jobs may arrive online at release dates and, at any time, an algorithm has access only to a permutation on jobs that have been released already. At any release date, the permutation is updated consistently with the previous permutation. That is, the prediction model is not allowed to change the relative order of previously known jobs.

We show the following results for scheduling with permutation predictions:

- We define an error measure η^p for permutation predictions and prove desired properties for it, such as PAC-learnability in the agnostic setting (see Section 7.2).

- We give a non-clairvoyant algorithm for the problem $1 \mid \text{pmtn} \mid \sum w_j C_j$ with a competitive ratio of at most
$$\min\left\{\frac{1}{1-\lambda}\left(1+\frac{\eta^p}{\text{OPT}}\right), \frac{2}{\lambda}\right\}$$
for any $\lambda \in (0,1)$ (see Section 7.4.1).

- We give a non-clairvoyant online algorithm for the problem $P \mid r_j, \text{pmtn} \mid \sum w_j C_j$ with a competitive ratio of at most
$$\min\left\{\frac{1}{1-\lambda}\left(2+\frac{\eta^p}{m \cdot \text{OPT}}\right), \frac{3}{\lambda}\right\}$$
for any $\lambda \in (0,1)$, where m is the number of machines (see Section 7.4.2).

Further, we revisit in Section 7.3 the Preferential Time Sharing framework introduced by Purohit et al. [PSK18], which we slightly modify and improve.

7.2 Prediction Error for Permutation Predictions

7.2.1 Intuition and Definition

Intuitively, an error measure shall quantify the impact that an erroneous prediction has on an (optimal) scheduling algorithm. It is not unnatural to express the error as $|\text{OPT}(\hat{\sigma}) - \text{OPT}(\sigma)|$, as has been done in [BMS20; Ebe+22; LMS22], but for more complex scheduling environments such as parallel identical machines or scheduling with release dates, the optimal solution is hard to compute and, more importantly, this error could be even negligible whereas the impact of running an optimal algorithm with the wrong prediction could be significant. The latter is what we want to quantify.

In more detail, our error measure shall capture the change in the cost that an optimal schedule must face when two jobs j and j' are inverted in a prediction $\hat{\sigma}$ with respect to σ. For example, on a single machine without release dates, if j and its successor j' in $\hat{\sigma}$ are swapped in σ, the schedule that follows $\hat{\sigma}$ pays an additional cost of $w_{j'}p_j$ but saves $w_j p_{j'}$ compared to the schedule that follows σ. However, in presence of release dates and on multiple machines, just knowing the orders may not allow us to express the change in the exact optimal cost. Therefore, we rely on an approximation as a surrogate for the optimal cost, namely, the *change in the sum of weighted completion times when preemptively scheduling jobs in the given priority order,* $\hat{\sigma}$ and σ.

Formally, for a scheduling instance with permutation prediction $\hat{\sigma}$ consisting of a single permutation, and WSPT order σ, let $\mathcal{I}(J,\hat{\sigma}) := \{(j',j) \in J^2 \mid \sigma(j') < \sigma(j) \wedge \hat{\sigma}(j') > \hat{\sigma}(j)\}$ be the set of inverted job pairs. The prediction error of $\hat{\sigma}$ is defined as
$$\eta^p(J,\hat{\sigma}) := \sum_{(j',j)\in\mathcal{I}(J,\hat{\sigma})} w_{j'}p_j - w_j p_{j'} \ .$$

This error measures the *exact* change in the objective value, in the absence of release dates.

7.2.2 Properties and Comparison to Other Error Measures

Our new error measure satisfies several desired properties such as (i) monotonicity, (ii) Lipschitzness, and (iii) theoretical learnability.

Im et al. [Im+23] advocate particularly the first two properties. *Monotonicity* requires, in the length prediction model, that the error grows as more length predictions become incorrect. In our setting, we have $\eta(\hat{\sigma}) = 0$ if $\hat{\sigma} = \sigma$, and for any inversion added to $\hat{\sigma}$, the error grows. This is because an inversion $(j', j) \in \mathcal{I}$ increases the error by $w_{j'}p_j - w_j p_{j'}$, since $\sigma(j') < \sigma(j)$ implies $w_{j'}/p_{j'} \geq w_j/p_j$. Thus, our definition satisfies monotonicity.

Lipschitzness requires the error to bound the absolute difference of the optimal objective values for the actual and predicted instance from above. Our error definition *precisely* measures the cost between a solution that follows $\hat{\sigma}$ and one that follows σ, when scheduling the actual instance preemptively according to the given order. Hence, our error measures immediately satisfy Lipschitzness for our prediction setup.

Our prediction model is *theoretically learnable* in the framework of PAC-learnability [SB14]. We show that permutations are efficiently PAC-learnable in the agnostic sense with respect to our error definition in Section 7.2.3.

In general, it is difficult to compare different prediction and error models. However, we can convert a given length prediction $\hat{p} = (\hat{p}_j)_j$ into a permutation prediction by simply computing the WSPT order based on the predicted processing requirements. Let $\mathrm{OPT}(q)$ denote the optimal objective value for the instance $q = (q_j)_j$ of the problem $1 \,|\, pmtn \,|\, \sum C_j$. This conversion allows us to compare our error to the previously proposed measures $\nu = \mathrm{OPT}(\max\{p, \hat{p}\}) - \mathrm{OPT}(\min\{p, \hat{p}\})$ [Im+23] and $\ell_1 = \sum_{j\in[n]} |p_j - \hat{p}_j|$ [PSK18] for the case of $1 \,|\, pmtn \,|\, \sum C_j$.

Firstly, we note that our error η^p is less vulnerable than ν and ℓ_1 to changes in the predicted instance that do not affect the *structure* of an optimal solution. Indeed, the optimal solution of an instance with $p_j = j$ for all $j \in [n]$ has the same structure as the optimal solution of a predicted instance with $\hat{p}_j = j - 1$ for all $j \in [n]$. One would expect a small error, and indeed $\eta^p = 0$. In contrast, previously defined errors are large: $\nu = \mathrm{OPT}(\max\{p, \hat{p}\}) - \mathrm{OPT}(\min\{p, \hat{p}\}) = \frac{1}{2}n(n+1) - \frac{1}{2}(n-1)n = n$ and $\ell_1 = \sum_{j\in J}|p_j - \hat{p}_j| = n$. This shows that our prediction and error seem to capture well the relevant characteristics of an input-prediction in terms of derived actions, while ν and ℓ_1 also track insignificant numerical differences between the actual and predicted instances.

In contrast to this example, there are other instances where ν and ℓ_1 underestimate the actual difficulty that is caused by the inaccuracy of the prediction given to an (optimal) algorithm. Im et al. [Im+23] give such an example with $p_1 = \hat{p}_1 = \ldots = p_{n-1} = \hat{p}_{n-1} = 1$ and $p_n = n^2$ but $\hat{p}_n = 0$. While the structural difference of the optimal solutions for predicted and true values is large ($\eta^p = \Omega(n^3)$) the other error definitions only measure $\nu = n^2 + n$ and $\ell_1 = n^2$.

Finally, is not difficult to see that our prediction error never exceeds $n\ell_1$.

Proposition 7.1. *For any instance of* $1 \,|\, pmtn \,|\, \sum C_j$ *and length prediction,* $\eta^p \leq n \cdot \ell_1$.

Proof. Consider an instance J and length prediction $\hat{p}$. Let $\hat{\sigma}$ be the corresponding predicted permutation. Since $(j', j) \in \mathcal{I}(J, \hat{\sigma})$ implies $\hat{\sigma}(j') > \hat{\sigma}(j)$, which must be due to $\hat{p}_j \leq \hat{p}_{j'}$, we conclude

$$\eta^p(J, \hat{\sigma}) = \sum_{(j',j)\in\mathcal{I}(J,\hat{\sigma})} p_j - \hat{p}_j + \hat{p}_j - \hat{p}_{j'} + \hat{p}_{j'} - p_{j'} \leq \sum_{(j',j)\in\mathcal{I}(J,\hat{\sigma})} |p_j - \hat{p}_j| + |p_{j'} - \hat{p}_{j'}| \leq n\ell_1 \,.$$

□

Our results for non-uniform job weights on a single and identical machines translate to the length prediction model, as one can similarly show that η^p is bounded by the natural weighted generalization of $n \cdot \ell_1$, that is $\sum_{j'\in J} w_{j'} \sum_{j\in J} |p_j - \hat{p}_j|$.

7.2.3 Learnability

We show that permutation predictions for identical machines are PAC-learnable in the agnostic sense with respect to η^p.

Theorem 7.2. *For any $\varepsilon, \delta \in (0,1)$ and any distribution $\mathcal{D}$ over the instances with n jobs, there exists a learning algorithm that, given an i.i.d. sample of $\mathcal{D}$ of size $z \in O\left(\frac{1}{\varepsilon^2} \cdot (n \log n - \log \delta) n^2\right)$, returns in polynomial time depending on n and z a prediction $\sigma_p \in \mathcal{H}$ from the set of all possible permutations of the set $\{1, \ldots, n\}$, such that with probability of at least $(1-\delta)$ it holds*

$$\mathbb{E}_{J\sim\mathcal{D}}[\eta^p(J, \sigma_p)] \leq \mathbb{E}_{J\sim\mathcal{D}}[\eta^p(J, \sigma)] + \varepsilon \,,$$

where $\eta^p(J, \hat{\sigma})$ denotes the error of $\hat{\sigma}$ for instance J, and $\sigma = \arg\min_{\hat{\sigma}\in\mathcal{H}} \mathbb{E}_{J\sim\mathcal{D}}[\eta^p(J, \hat{\sigma})]$.

Proof. Let $\varepsilon, \delta \in (0,1)$. We prove that we can use the classic Empirical Risk Minimization (ERM) learning method to find such a prediction. Let $\mathcal{S} = \{J_1, \ldots, J_z\}$ be a set of i.i.d. samples from $\mathcal{D}$. The ERM method then determines the prediction that minimizes the empirical error $\eta^p_{\mathcal{S}}(\hat{\sigma}) = \frac{1}{z}\sum_{s=1}^{z} \eta^p(J_s, \hat{\sigma})$. Since there are $n!$ possible permutations of the set $\{1, \ldots, n\}$, we conclude that $\mathcal{H}$ is finite, and we can assume by scaling processing requirements and weights to $[0,1]$ that our error function is bounded by n. Classic results [SB14] imply for this case that $\mathcal{H}$ is agnostically PAC learnable using the ERM method with sample complexity

$$z \leq \left\lceil \frac{2\log(2|\mathcal{H}|/\delta)n^2}{\varepsilon^2} \right\rceil \in O\left(\frac{(n\log n - \log\delta)n^2}{\varepsilon^2}\right) \,,$$

which is polynomial in the number of jobs, n, as $\log n! \in O(n \log n)$.

It remains to prove that the ERM algorithm can be implemented efficiently in our setting, that is, given a sample set of size z, determine in time polynomial in n, a prediction that minimizes the empirical error. Rewriting the empirical error gives

$$\eta^p_{\mathcal{S}}(\hat{\sigma}) = \frac{1}{z}\sum_{s=1}^{z} \eta^p(J_s, \hat{\sigma}) = \frac{1}{z}\sum_{s=1}^{z}\sum_{j=1}^{n} \left(W_j(J_s, \hat{\sigma}) - W_j(J_s, \sigma)\right) \,.$$

Since the values $W_j(J_s, \sigma)$ are independent of $\hat{\sigma}$, it suffices to find a prediction $\hat{\sigma}$ that minimizes $\frac{1}{z}\sum_{s=1}^{z}\sum_{j=1}^{n} W_j(J_s, \hat{\sigma})$. For the special error η^p, by denoting for a job $j \in J_s$ its weight by $w_j^{(s)}$ and its processing requirement by $p_j^{(s)}$, this is equal to

$$\frac{1}{z}\sum_{s=1}^{z}\sum_{j=1}^{n} W_j(J_s, \hat{\sigma}) = \frac{1}{z}\sum_{s=1}^{z}\sum_{j=1}^{n} w_{\hat{\sigma}(j)}^{(s)} \sum_{\ell=1}^{j} p_{\hat{\sigma}(\ell)}^{(s)} = \sum_{j=1}^{n}\left(\frac{1}{z}\sum_{s=1}^{z} w_{\hat{\sigma}(j)}^{(s)}\right)\sum_{\ell=1}^{j}\left(\frac{1}{z}\sum_{s=1}^{z} p_{\hat{\sigma}(\ell)}^{(s)}\right) .$$

By defining the average weight $\bar{w}_{\hat{\sigma}(j)} = \frac{1}{z}\sum_{s=1}^{z} w_{\hat{\sigma}(j)}^{(s)}$ and average processing requirement $\bar{p}_{\hat{\sigma}(j)} = \frac{1}{z}\sum_{s=1}^{z} p_{\hat{\sigma}(j)}^{(s)}$ over $\mathcal{S}$ for all $j \in [n]$, this is equal to minimizing

$$\sum_{j=1}^{n} \bar{w}_{\hat{\sigma}(j)} \sum_{\ell=1}^{j} \bar{p}_{\hat{\sigma}(\ell)} .$$

Consider the *average* instance of $\mathcal{S}$, that is, the scheduling instance of n jobs with weights $(\bar{w}_j)_j$ and processing requirements $(\bar{p}_j)_j$. Since the above expression is equal to the objective value of this instance when scheduling jobs in order $\hat{\sigma}(1), \ldots, \hat{\sigma}(n)$, we can minimize it by ordering the jobs according to WSPT in polynomial time in z and n [Smi56]. □

7.3 Preferential Time Sharing

In this section, we revisit the *Preferential Time Sharing* method introduced by Purohit et al. [PSK18] for combining two scheduling algorithms, $\mathcal{A}$ and $\mathcal{B}$. This method produces a new algorithm with a competitive ratio that is a function of the competitive ratios of the individual algorithms $\mathcal{A}$ and $\mathcal{B}$. We generalize this idea into a powerful framework that is applicable in substantially more complex scheduling settings and remove a previously required monotonicity assumption.

The idea by Purohit et al. [PSK18] is to execute two algorithms somewhat simultaneously by alternating between them very frequently. They crucially exploit the fact that job preemption does not incur any extra cost or delay. A parameter $\lambda \in (0, 1)$ is used to control the balancing of the execution rates. More precisely, we fix a time discretization into infinitesimal time intervals and execute both algorithms in each time interval. Algorithm $\mathcal{A}$ runs in the first $1-\lambda$-fraction of the interval, then the currently processing jobs are preempted, and algorithm $\mathcal{B}$ runs in the remaining λ-fraction of the interval. Purohit et al. [PSK18] prove the following result for the single-machine environment, which can be easily extended to more complex heterogeneous machine settings, but restricted to algorithms $\mathcal{A}$ and $\mathcal{B}$ that are both *monotone*. An algorithm is monotone if increasing the processing requirement of a job in any instance does not decrease the overall objective value.

Theorem 7.3 (Corollary of [PSK18]). *Given any two deterministic monotone algorithms $\mathcal{A}$ and $\mathcal{B}$ for a preemptive scheduling problem with competitive ratios $\rho_{\mathcal{A}}$ and $\rho_{\mathcal{B}}$ for minimizing the total weighted completion time and a constant $\lambda \in (0, 1)$, there exists an algorithm for the same problem with a competitive ratio of at most* $\min\{\frac{\rho_{\mathcal{A}}}{1-\lambda}, \frac{\rho_{\mathcal{B}}}{\lambda}\}$.

This bound follows because the completion times scale with the execution rate, and whenever $\mathcal{B}$ reduces the remaining processing time of a job the completion time in $\mathcal{A}$'s schedule does not decrease due to the assumed monotonicity. Notice that in this algorithm, both subroutines $\mathcal{A}$ and $\mathcal{B}$ can be applied at any time as black-box algorithms to the remaining scheduling instance.

We give an alternative formulation of the method, which does *not* require monotonicity of the involved algorithms. Instead, we require that the algorithms are *progress-aware*, meaning that they can track the exact progress (amount of processing) that jobs have received so far at any point in time. This allows us to simulate both algorithms on the original instance while ignoring the progress made by the other algorithm. While this algorithm formulation may seem less relevant in practice, it is interesting from a theoretical perspective as it removes the monotonicity requirement and the ignored progress does not lead to degraded performance in terms of competitive ratio.

Formally, we consider the algorithm that simulates each of both algorithms $\mathcal{A}$ and $\mathcal{B}$ with a rate equal to $1-\lambda$ and λ on the same set of jobs. That is, in every time instance it executes algorithm $\mathcal{A}$ in the first $1-\lambda$ fraction of the instant, and algorithm $\mathcal{B}$ in the remaining part. Additionally, it keeps track of how much each algorithm advances every job. For every job j, both algorithms ignore the progress in the processing of j made by the other algorithm. In particular, if $\mathcal{A}$ finishes job j, then our algorithm still simulates the processing of j until the total time spend by $\mathcal{B}$ on the (partially simulated) processing of j equals p_j (and vice versa if $\mathcal{B}$ finishes a job before $\mathcal{A}$). This is possible because the algorithms are progress-aware. We remark that this requires that the main algorithm is able to manipulate the input for the sub-algorithms $\mathcal{A}$ and $\mathcal{B}$.

Theorem 7.4. *Given any two deterministic progress-aware algorithms $\mathcal{A}$ and $\mathcal{B}$ for a preemptive scheduling problem with competitive ratios $\rho_{\mathcal{A}}$ and $\rho_{\mathcal{B}}$ for minimizing the total weighted completion time and a constant $\lambda \in (0,1)$, there exists an algorithm for the same problem with a competitive ratio of at most* $\min\{\frac{\rho_{\mathcal{A}}}{1-\lambda}, \frac{\rho_{\mathcal{B}}}{\lambda}\}$.

Proof. We consider the algorithm described above. Let $\tilde{C}_j^{\mathcal{A}}$ and $\tilde{C}_j^{\mathcal{B}}$ be the completion time of j in the simulated combined schedule, and $C_j^{\mathcal{A}}$ and $C_j^{\mathcal{B}}$ be the completion time of j in the independent schedule of the algorithm $\mathcal{A}$ and $\mathcal{B}$, respectively. Since both algorithms are deterministic, $\tilde{C}_j^{\mathcal{A}}$ and $\tilde{C}_j^{\mathcal{B}}$ are exactly scaled by factors $\frac{1}{1-\lambda}$ and $\frac{1}{\lambda}$ compared to $C_j^{\mathcal{A}}$ and $C_j^{\mathcal{B}}$. Thus, job j completes no later than $\min\{\tilde{C}_j^{\mathcal{A}}, \tilde{C}_j^{\mathcal{B}}\} = \min\{\frac{1}{1-\lambda}C_j^{\mathcal{A}}, \frac{1}{\lambda}C_j^{\mathcal{B}}\}$ in the combined schedule, which implies the stated bound on the algorithm's competitive ratio. □

7.4 Learning-Augmented Algorithms

7.4.1 Single Machine

Consider the problem $1 \mid \text{pmtn} \mid \sum w_j C_j$. Given a permutation prediction $\hat{\sigma}$, consider the algorithm that schedules the jobs in the predicted order. Clearly, if the prediction is accurate, this algorithm computes the optimal solution. Otherwise, the following lemma bounds competitive ratio in terms of the prediction error η^p.

Lemma 7.5. *For every instance J and permutation prediction $\hat{\sigma}$,* $\text{ALG} \leq \text{OPT} + \eta^p(J, \hat{\sigma})$.

Proof. Consider an instance J with jobs being indexed by σ, a prediction $\hat{\sigma}$, and the schedule obtained by the algorithm. In this schedule, let $d(j', j)$ denote the amount of job j' that has been processed before job j completed. Thus, $d(j', j) = p_{j'}$ if and only if $\hat{\sigma}(j') < \hat{\sigma}(j)$. This implies

$$\begin{aligned}
\mathrm{ALG} &= \sum_{j=1}^{n} w_j p_j + \sum_{j=1}^{n} \sum_{j'=1}^{j-1} \left(w_j \cdot d(j', j) + w_{j'} \cdot d(j, j') \right) \\
&= \sum_{j=1}^{n} w_j p_j + \sum_{j=1}^{n} \sum_{\substack{j'=1 \\ \hat{\sigma}(j') < \hat{\sigma}(j)}}^{j-1} w_j p_{j'} + \sum_{j=1}^{n} \sum_{\substack{j'=1 \\ \hat{\sigma}(j') > \hat{\sigma}(j)}}^{j-1} w_{j'} p_j \\
&= \sum_{j=1}^{n} w_j \sum_{j'=1}^{j} p_{j'} + \sum_{j=1}^{n} \sum_{\substack{j'=1 \\ \hat{\sigma}(j') > \hat{\sigma}(j)}}^{j-1} (w_{j'} p_j - w_j p_{j'}) = \mathrm{OPT} + \eta^p(J, \hat{\sigma}) \,.
\end{aligned}$$

The last equation holds since the first sum equals the optimal objective value, that is, a schedule according to σ, and the second sum equals $\eta^p(J, \hat{\sigma})$, since we assumed the jobs to be indexed according to σ. □

So far, this algorithm has no robustness guarantee. To this end, we combine it with the Weighted-Round-Robin algorithm (respectively, Proportional Fairness; see Chapter 5), which is a non-clairvoyant 2-competitive for our problem [KC03]. Both algorithms are progress-aware. Thus, Theorem 7.4 gives the following result.

Theorem 7.6. *For every $\lambda \in (0, 1)$, there is a non-clairvoyant algorithm augmented with permutation predictions that has a competitive ratio of at most*

$$\min \left\{ \frac{1}{1-\lambda} \left(1 + \frac{\eta^p}{\mathrm{OPT}} \right), \frac{2}{\lambda} \right\}$$

for minimizing the total weighted completion time on a single machine, $1 \,|\, pmtn \,|\, \sum w_j C_j$.

Finally, we note that the consistency-robustness tradeoff between $\frac{1}{1-\lambda}$ and $\frac{2}{\lambda}$ of our algorithm is asymptotically almost optimal with respect to λ. This follows from a result by Wei and Zhang [WZ20], who showed that for any $\delta \geq 0$, any $(1+\delta)$-consistent algorithm for $1 \,|\, \mathrm{pmtn} \,|\, \sum C_j$ has a robustness ratio of at least $\Omega((n + \delta n^2)/(\delta n^2))$ for n jobs. Since this result was proven for length predictions, it also holds for permutation predictions. Choosing $\delta := \frac{1}{1-\lambda} - 1$ and $n := \sqrt{1/\delta}$ implies that any $\frac{1}{1-\lambda}$-consistent algorithm for $1 \,|\, \mathrm{pmtn} \,|\, \sum C_j$ with permutation predictions has a robustness ratio of at least

$$\Omega\left(\frac{n + \delta n^2}{\delta n^2} \right) = \Omega\left(\sqrt{\frac{1}{\delta}} \right) = \Omega\left(\sqrt{\frac{1-\lambda}{\lambda}} \right) = \Omega\left(\sqrt{\frac{1}{\lambda}} \right) .$$

7.4.2 Parallel Identical Machines

Next, consider the more general problem of scheduling jobs with non-uniform release dates on m parallel identical machines, $P \,|\, r_j, \mathrm{pmtn} \,|\, \sum w_j C_j$.

We assume that we are given a single permutation $\hat{\sigma}$ over all jobs. First consider the algorithm that schedules, at any moment in time, the m available jobs with the highest priority in the predicted order $\hat{\sigma}$. If $\hat{\sigma}$ is accurate, this algorithm corresponds to PreemptiveWSPT on parallel identical machines, which is 2-competitive [MS04].

Lemma 7.7. *For every instance J and permutation prediction $\hat{\sigma}$,* $\mathrm{ALG} \leq 2 \cdot \mathrm{OPT} + \frac{1}{m}\eta^p(J, \hat{\sigma})$.

Proof. Consider an instance J with jobs being indexed by σ, a prediction $\hat{\sigma}$, and the schedule obtained by the algorithm. After job j has been released, it is either being processed on a machine or it is delayed by another job. Let $d(j', j)$ denote the total amount of job j' that delays the completion of j. Note that $d(j', j) \leq p_{j'}$. Such a delay can only occur if there are at least m alive jobs before j in $\hat{\sigma}$, and these jobs will be distributed over all m machines. Since j has received p_j units of processing by its completion time, we conclude

$$\begin{aligned}
\mathrm{ALG} &\leq \sum_{j=1}^{n} w_j(r_j + p_j) + \frac{1}{m}\sum_{j=1}^{n}\sum_{j'=1}^{j-1}\left(w_j \cdot d(j', j) + w_{j'} \cdot d(j, j')\right) \\
&\leq \mathrm{OPT} + \frac{1}{m}\sum_{j=1}^{n}\sum_{\substack{j'=1 \\ \hat{\sigma}(j')<\hat{\sigma}(j)}}^{j-1} w_j p_{j'} + \frac{1}{m}\sum_{j=1}^{n}\sum_{\substack{j'=1 \\ \hat{\sigma}(j')>\hat{\sigma}(j)}}^{j-1} w_{j'} p_j \\
&= \mathrm{OPT} + \frac{1}{m}\sum_{j=1}^{n} w_j \sum_{j'=1}^{j-1} p_{j'} + \frac{1}{m}\sum_{j=1}^{n}\sum_{\substack{j'=1 \\ \hat{\sigma}(j')>\hat{\sigma}(j)}}^{j-1} w_{j'} p_j - w_j p_{j'} \leq 2 \cdot \mathrm{OPT} + \frac{1}{m} \cdot \eta^p(J, \hat{\sigma}) \,.
\end{aligned}$$

The second and third inequality hold due to two classical lower bounds on an optimal solution: Every job has to be processed by at least its p_j after its release in any solution. Further, $\frac{1}{m}\sum_{j=1}^{n} w_j \sum_{j'=1}^{j-1} p_{j'}$ equals the objective value of the WSPT schedule on a single machine with speed m without release dates, which is a known relaxation of our problem and therefore also a lower bound on OPT [EEI64]. Since we assumed that the jobs are indexed according to σ, the sum of inversions is equal to $\eta^p(J, \hat{\sigma})$. □

We combine this algorithm with Proportional Fairness (cf. Chapter 5), which has a competitive ratio of at most 3 for this problem (cf. Theorem 5.31). Both algorithms are progress-aware. Therefore, we conclude the following theorem via Theorem 7.4.

Theorem 7.8. *For every $\lambda \in (0, 1)$, there is a non-clairvoyant algorithm augmented with permutation predictions that has a competitive ratio of at most*

$$\min\left\{\frac{1}{1-\lambda}\left(2 + \frac{\eta^p}{m \cdot \mathrm{OPT}}\right), \frac{3}{\lambda}\right\}$$

for minimizing the total weighted completion time on m parallel identical machines with release dates, $P \mid r_j, pmtn \mid \sum w_j C_j$.

7.5 Concluding Remarks

We proposed a new compact prediction model and error measure that fulfill desired properties in theory and practice. Further, we revisited a learning-augmented time sharing framework and generalized and improved it.

Subsequently, to the conference version on which this chapter is based on [LM22], our new permutation prediction has also been successfully applied to other problems and models in the area of algorithms with predictions [Din+22; Eli+24; Kho+22; Las+23], demonstrating its power and applicability.

Our model is an early example for the theoretical study of parsimonious or succinct prediction models that aim to use less or even as little predictive information as possible. Further works dealing with such considerations include [Ant+23a; BP23; DNS23; Im+23; Las+23; SE24]. It is an interesting question whether one can formalize what constitutes a "minimalistic" information in some way, such as the number of bits in advice complexity [Boy+17], which assumes precise additional information.

Bibliographic Note

This chapter is mainly based on joint work with Nicole Megow [LM22]. Section 7.3 is based on joint work with Nicole Megow and Martin Rapp [LMR23]. Thus, some parts of this chapter are identical with [LM22; LMR23]

Chapter 8

Predictions for Unknown Precedence Constraints

8.1 Introduction

Cloud computing is a popular approach to outsource heavy computational tasks to specialized providers [Hay08]. Concepts like Function-as-a-Service (FaaS) offer users on demand the execution of complex computations in a specific domain [Lyn+17; SBW19]. Such tasks are often decomposed into smaller jobs, which then depend on each other by passing intermediate results. The structure of such tasks heavily relies on the users input and internal dependencies within the user's system. It might require diverse jobs to solve different problems with distinct inputs. From the provider's perspective, the goal is to schedule jobs with different priorities and interdependencies that become known only when certain jobs are completed and their results can be evaluated.

From a more abstract perspective, we face *online precedence constraint scheduling*: new jobs arrive only if certain other jobs have been completed but the set of jobs and their dependencies are unknown to the scheduler in advance. As tasks might have different priorities, it is a natural objective to minimize the total (average) weighted completion time of the jobs. We focus on *non-clairvoyant* schedulers, and we allow *preemptive* schedules.

It is not hard to see that for this online problem, we cannot hope for good worst-case guarantees: consider an instance of $n - 1$ initially visible jobs with zero weight such that exactly one of these jobs triggers at its completion the arrival of a job with positive weight. Since the initial jobs are indistinguishable, in the worst case, any algorithm completes the positive-weight job last. An offline optimal solution can distinguish the initially visible jobs and immediately processes the one that triggers the positive-weight job. This already shows that no deterministic algorithm can have a better competitive ratio than $\Omega(n)$ for n jobs. Notice that this strong impossibility result holds even for (seemingly) simple precedence graphs that consist of a collection of chains. In practice, such a topology is highly relevant as, for example, a sequential computer program executes a path (chain) of instructions that upon execution depends on the evaluation of control flow structures [All70].

To overcome such daunting worst-case lower bounds, we study this problem in the *learning-augmented* setting. The intuition is that in many applications, we can learn certain aspects of the uncertainty by considering historical data such as dependencies between jobs for certain computations and inputs.

Despite the immense research interest in learning-augmented algorithms, the particular choice of prediction models remains often undiscussed. In this chapter, we discuss various models and analyze their strengths and weaknesses. The question driving our research is:

> *Which particular minimal information is required to achieve reasonable performance guarantees for scheduling with online precedence constraints?*

We refer with reasonable performance guarantees to competitive ratios in $o(n)$, that is, bounds that improve upon the pessimistic lower bound of $\Omega(n)$ from the setting without additional information.

Our starting point is the analysis of the two most common models, *full input* predictions and *action predictions*. Our main focus is a hierarchy of refined prediction models based on their entropy. That is, one can compute a prediction for a weaker model using a prediction from a stronger one, but not vice versa. We predict quantities related to the weight of unknown jobs, which is in contrast to previous work that assumes predictions on the jobs' processing times or machine speeds.

For each prediction model, we analyze its power and limitations by providing efficient algorithms and lower bounds on the best-possible performance guarantees with respect to these models and the topological properties of the precedence constraints.

8.1.1 Problem Definition

An instance of our problem is composed of a set J of n jobs and a precedence graph $G = (J, E)$, which is a directed acyclic graph (DAG). Every job $j \in J$ has a processing requirement $p_j > 0$ and a weight $w_j \geq 0$. An edge $(j', j) \in E$ indicates that j can only be started if j' has been completed. If there is a directed path from j' to j in G, then we say that j is a *successor* of j' and that j' is a *predecessor* of j. If that path consists of a single edge, we call j and j' a *direct* successor and predecessor, respectively. For a fixed precedence graph G, we denote by ω the *width* of G, which is the length of the longest anti-chain in G.

We consider rate-based schedules for this problem, as introduced in Section 2.2.1. An algorithm can process a job j at a time $t \geq 0$ with a rate $y_j(t) \geq 0$, which describes the amount of processing the job receives at time t. The completion time C_j of a job j is the first time t that satisfies $\int_0^t y_j(t')\,\mathrm{d}t' \geq p_j$. On a single machine a total rate of 1 can be processed at any time t, and thus, we require $\sum_{j \in J} y_j(t) \leq 1$. At any time t in a schedule, let

$$F(t) := \left\{ j \mid C_j > t \text{ and } \forall j' \text{ s.t. } (j', j) \in E \colon C_{j'} < t \right\}$$

denote the set of unfinished jobs without unfinished predecessors in G. We refer to such jobs as *front jobs*. In the online setting, a job is revealed to the algorithm once all predecessors have been completed. The algorithm is completely oblivious to G and, specifically, it does not know whether a front job has successors. Thus, at any time t, an algorithm only sees jobs $j \in F(t)$ with weights w_j but *not* their processing times p_j. Note that the sets $F(t)$ heavily depend on an algorithm's actions. At the start time $t = 0$, an algorithm sees $F(0)$, and until the completion of the last job, it does not know the total number of jobs. An algorithm can at any time t only process front jobs, hence we further require that $y_j(t) = 0$ for all $j \in J \setminus F(t)$. The objective of

our problem is to minimize $\sum_{j \in J} w_j C_j$. For a fixed instance, we denote the optimal objective value by OPT and for a fixed algorithm, we denote its objective value by ALG.

We study different topologies of precedence graphs. In addition to general DAGs, we consider *in-forests* and *out-forests*, where every node has at most one outgoing and incoming edge, respectively. Further, we study *chains*, which is a precedence graph that is an in-forest and an out-forest simultaneously. If an in- or out-forest has only one connected component, we refer to it as *in-* and *out-tree*, respectively.

8.1.2 Prediction Models

As we have discussed in Chapter 6, two of the most studied prediction models are the following:

Full input predictions: Predictions on the set of jobs with processing times and weights, and the complete precedence graph.

Action predictions: Predictions on a full priority order over all jobs predicted to be part of the instance (*static*) or a prediction on which job to schedule next whenever a machine idles (*adaptive*).

Full input predictions, however, require a significant amount of information on the input. The same holds for provably optimal action predictions that go beyond simple heuristics, which can be computed based on limited information. This might be unrealistic or costly to obtain and/or not necessary. We aim for minimalistic extra information and quantify its power.

The set of front jobs $F(0)$ does not give sufficient information for obtaining a competitive ratio better than $\Omega(n)$, as shown above. For a job $v \in F(0)$, we define the set $S(v)$ consisting of v and its successors, and we let $w(S(v)) := \sum_{u \in S(v)} w_u$. We consider various predictions on the set $S(v)$:

Weight predictions: Predictions $\hat{W}_v$ on the total weight $w(S(v))$ of each front job $v \in F(0)$.

Weight order predictions: The *weight order* $\preceq_0$ over $F(0)$ sorts the jobs $v \in F(0)$ in order of non-increasing $w(S(v))$, that is, $v \preceq_0 u$ implies $w(S(v)) \geq w(S(u))$. We assume access to a prediction $\hat{\preceq}_0$ on $\preceq_0$.

Average predictions: Predictions $\hat{a}_v$ on the average weight $a(S(v)) = \frac{\sum_{u \in S(v)} w_u}{\sum_{u \in S(v)} p_u}$ of each front job $v \in F(0)$.

For each of these three models, we distinguish *static* and *adaptive* predictions. Static predictions refer to predictions only on the initial front jobs $F(0)$, and adaptive predictions refer to a setting where we receive access to a new prediction whenever a job becomes visible.

8.1.3 Our Results

Our results can be separated into two categories. First, we consider the problem of scheduling with online precedence constraints with access to additional *reliable* information. In particular, we consider all the aforementioned prediction models and design upper and lower bounds for the online problem enhanced with access to the respective additional information. We classify the power of the different models when solving the problem on different topologies.

Table 8.1: Summary of bounds in this chapter on the competitive ratio given reliable information. We denote by P the total processing time and by H_k the kth harmonic number.

Prediction Model	Topology	Competitive Ratio
Actions	DAG	$\Theta(1)$
Input	DAG	$\Theta(1)$
Adaptive weights	Out-Forests	$\Theta(1)$
Adaptive weights	In-Trees	$\Omega(\sqrt{n})$
Static weights	Out-Trees	$\Omega(n)$
Static weights	Chains	$\Theta(1)$
Adaptive weight order	Out-Forests	$O(H_\omega)$
Static weight order	Chains	$O(H_\omega^2\sqrt{P})$
Adaptive averages	Chains	$\Omega(\sqrt{n})$
No prediction	Chains	$\Omega(n)$

For the second type of results, we drop the assumption that the additional information is accurate and turn our pure online results into learning-augmented algorithms. We define suitable error measures for the different prediction models to capture the accuracy of the predictions, and give more fine-grained competitive ratios depending on these measures. We also extend our algorithms to achieve robustness.

Next, we give an overview of our results for these categories.

Reliable additional information. Table 8.1 summarizes our results for the pure online setting enhanced with reliable additional information. Our main results are a 4-competitive algorithms for chains and out-forests with weight predictions, and a H_ω-competitive algorithm for out-forests with adaptive weight order predictions, where H_k is the kth harmonic number. The results show that additional information significantly improves the (worst-case) ratio compared to the setting with no predictions.

Our main non-clairvoyant algorithm, given correct weight predictions, has a competitive ratio of at most 4 for online out-forest precedence constraints on a single machine. This improves even for offline precedence constraints upon previous best-known bounds of 8 [Jäg21] and 10 [Gar+19] for this problem, although these bounds also hold in more general settings[1]. To achieve this small constant, we generalize the Weighted-Round-Robin (WRR) algorithm [KC03; MPT94] for non-clairvoyant scheduling *without* precedence constraints, which advances jobs proportional to their weight, to our setting. We handle each out-tree as a super-job and update its remaining weight when a sub-job completes. If the out-tree is a chain, this can be done even if only *static* weight predictions are given. Otherwise, when an out-tree gets divided into multiple remaining out-trees, the distribution of the remaining weight is unknown. Thus, we have to rely on *adaptive* predictions. Due to the increased dynamics of gaining partial weight of

[1] After the conference version of this chapter [Las+23] has been published, Jäger and Warode [JW24] presented a 2-competitive non-clairvoyant algorithm for minimizing the total weighted completion time on a single machine with offline precedence constraints, which is best-possible [MPT94].

these super-jobs, the original analysis of WRR is not applicable. Instead, we use the dual fitting technique, which has been previously used for offline precedence constraints [Gar+19]. While their analysis builds on offline information and is infeasible in our model, we prove necessary conditions on an algorithm to enable the dual fitting, which are fulfilled even in our limited information setting. Surprisingly, we also show that a more compact LP relaxation, which does not consider transitive precedences, is sufficient for our result. In particular, compared to the LP used in [Gar+19], it allows us to craft simple duals that do not involve gradient-type values of the algorithm's rates.

In the more restricted model of weight order predictions, WRR cannot directly be applied, as even the initial rate computation of the algorithm crucially relies on *precise* weight values (cf. Section 8.3.1). We observe, however, that WRR's rates at the start of an instance have the same ordering as the known chain order. We show that guessing rates for chains in a way that respects the ordering compromises only a factor of at most H_ω in the competitive ratio. If the weight order is adaptive, we show a competitive ratio of $4 \cdot H_\omega$. Otherwise, we give a worse upper bound and evidence that this might be best-possible for this algorithm.

Learning-augmentation. We extend our algorithmic results by designing suitable error measures for the different prediction models and proving error-dependent competitive ratios. Finally, we show how existing techniques can be used to give these algorithms a robustness of $\mathcal{O}(\omega)$ at the loss of only a constant factor in the error-dependent guarantee. Note that a robustness $\mathcal{O}(\omega)$ matches the lower bound for the online problem without access to additional information.

8.1.4 Further Related Work

Scheduling jobs with precedence constraints to minimize the sum of (weighted) completion times is a well-studied scheduling problem. The offline problem is known to be NP-hard, even for a single machine [Law78; LK78], and on two machines, even when precedence constraints form chains [DLY91; Tim03]. Several polynomial-time algorithms based on different linear programming formulations achieve an approximation ratio of 2 on a single machine, whereas special cases are even solvable optimally; we refer to [AM09; CS05] for comprehensive overviews. For scheduling on m parallel identical machines, the best known approximation factor is $3 - 1/m$ [Hal+97].

For scheduling with online precedence constraints, strong and immediate lower bounds rule out a competitive ratio better than $\Omega(n)$ for the min-sum objective. Therefore, online scheduling has been mainly studied in a setting where jobs arrive online, and once a job arrives its processing time, weight, and relation to other (already arrived) jobs is revealed [BKL21; Cha+96; Hal+97].

8.2 Robustness via Preferential Time Sharing

We show that any ρ-competitive algorithm for scheduling with online precedence constraints of width ω can be extended to a $O(\min\{\rho, \omega\})$-competitive algorithm. In particular, if ρ depends

on a prediction's quality, this ensures that this algorithm is robust against arbitrarily bad predictions.

To this end, consider the algorithm that at any time t shares the machine equally among all front jobs $F(t)$, that is, gives every job $j \in F(t)$ rate $y_j(t) = \frac{1}{|F(t)|} \geq \frac{1}{\omega}$. For a fixed job j, compared to its completion time in a fixed optimal schedule, the completion time in the algorithm's schedule can be delayed by at most a factor of ω.

Proposition 8.1. *There is an ω-competitive non-clairvoyant single-machine algorithm for minimizing the total weighted completion time of jobs with online precedence constraints.*

We can now use Preferential Time Sharing from Chapter 7 (cf. Theorem 7.4) to combine this ω-competitive algorithm with any other algorithm for scheduling online precedence constraints while maintaining the better competitive ratio of both up to a factor of 2.

8.3 Weight Value Predictions

We begin with problem-specific prediction models, starting with weight value predictions. We first prove strong lower bounds for algorithms with access to static weight predictions on out-trees and adaptive predictions on in-trees. Then, we give 4-competitive algorithms for accurate static predictions on chains, and adaptive weight predictions on out-forest precedence constraints, and finally extend these results to obtain robust algorithms with error dependency.

The lower bound for out-trees adds a dummy root r to the pure online lower bound composed of $\Omega(n)$ zero weight jobs, where exactly one hides a valuable job. In the static prediction setting, we thus only receive a prediction for r, which does not help any algorithm to improve.

Observation 8.2. *Any algorithm that has only access to static weight predictions has a competitive ratio of at least $\Omega(n)$, even if the precedence constraint graph is an out-tree.*

For in-trees and adaptive weight predictions, we prove the following lower bound.

Lemma 8.3. *Any algorithm that has only access to adaptive weight predictions has a competitive ratio of at least $\Omega(\sqrt{n})$, even for in-tree precedence constraints.*

Proof. Consider an in-tree instance with unit-size jobs and root r of weight 0. There are $\sqrt{n}$ chains of length 2 with leaf weights 0 and inner weights 1 that are connected to r. Further, there are $n-2\sqrt{n}-1$ leaves with weight 0, which are connected to a node v with weight 1, which itself is a child of r. Note that the weight prediction for all potential front jobs except r is always 1. Thus, even the adaptive predictions do not help, and we can assume that the algorithm first processes the children of v, giving a total objective of at least $\Omega((n-2\sqrt{n}-1)^2+(n-2\sqrt{n}-1)\sqrt{n}) = \Omega(n\sqrt{n})$, while processing the other leaves first yields a value of at most $O((2\sqrt{n})^2 + (2\sqrt{n}+n-2\sqrt{n})) = O(n)$. □

8.3.1 Algorithms for Reliable Information

We present algorithms assuming access to *correct* static or adaptive weight predictions and prove their competitiveness on online chain and out-forest precedence constraints using a unified analysis framework. This uses a dual fitting argumentation inspired by an analysis

of an algorithm for *known* precedence constraints [Gar+19]. The framework only requires a condition on the rates at which an algorithm processes front jobs, hence it is independent of the considered prediction model. Let $U(t)$ be the set of unfinished jobs at time t, that is, $U(t) = \bigcup_{v \in F(t)} S(v)$. Denote by $w(J')$ the total weight of jobs in a set J'. We write $W(t)$ for $w(U(t))$.

Theorem 8.4. *If an algorithm for online out-forest precedence constraints satisfies at every time t and for each $j \in F(t)$ that $w(S(j)) \leq \rho \cdot y_j(t) \cdot W(t)$, where $y_j(t)$ is the processing rate of j at time t, it is at most 4ρ-competitive for minimizing the total weighted completion time on a single machine.*

We first present algorithms for weight predictions and derive results using Theorem 8.4, and finally prove the theorem.

Static Weight Values for Chains. We give an algorithm for correct static weight predictions. As Observation 8.2 rules out well-performing algorithms for out-tree precedence constraints with static weight predictions, we focus on chains. Correct static weight predictions mean access to the total weight W_c of every chain c in the set of chains $\mathcal{C}$.

Algorithm 2: Weighted-Round-Robin on Chains

Input: Chains $\mathcal{C}$, initial total weight W_c for each $c \in \mathcal{C}$.

$t \leftarrow 0$ and $W_c(t) \leftarrow W_c$ for every $c \in \mathcal{C}$.
while *there are unfinished jobs* **do**
  Process the front job j_c of every $c \in \mathcal{C}$ at rate $y_{j_c}(t) = W_c(t) / \sum_{c \in \mathcal{C}} W_c(t)$.
  If j_c completes, update $W_c(t^+) \leftarrow W_c(t) - w_{j_c}$, else $W_c(t^+) \leftarrow W_c(t)$.
  $t \leftarrow t^+$.

Algorithm 2 executes a classical Weighted-Round-Robin strategy where the rate at which the front job of a chain c is executed at time t is proportional to the total weight of unfinished jobs in that chain, $W_c(t)$. As this definition is infeasible for unfinished chains with $W_c(t) = 0$, we process these in an arbitrary order in the end. As they have no weight, this does not negatively affect the objective. We denote by t^+ the immediate time instant after time t.

Despite initially only having access to the weights $W_c(t)$ for $t = 0$, the algorithm can compute $W_c(t)$ for any $t > 0$ by subtracting the weight of finished jobs of c from the initial W_c. Thus, $W_c(t) = w(S(j))$ holds for any time t and every $j \in F(t)$, where c is the corresponding chain of job j. Further, $W(t) = \sum_c W_c(t)$. We conclude that, for any t and $j \in F(t)$, it holds $y_j(t) = w(S(j))/W(t)$. Using Theorem 8.4 with $\rho = 1$, we can immediately derive the following result:

Theorem 8.5. *Given correct weight predictions, Algorithm 2 is a non-clairvoyant 4-competitive algorithm for minimizing the total weighted completion time of jobs with online chain precedence constraints on a single machine.*

We remark that Algorithm 2 crucially relies on access to the *precise* weight values. Even if all chain weights were just overpredicted by the same constant factor $\alpha > 1$, the algorithm

would not be constant competitive anymore. While the algorithm would compute the same initial rates as it does with access to the correct weights, it cannot precisely recompute the rates in Line 3. Instead, it slightly overestimates the remaining weight of a large number of chains, which can slow down important chains that contribute a lot to the objective value by a large factor. We use this intuition to prove the following lemma.

Lemma 8.6. *Given wrong weight predictions with $\hat{W}_c = \alpha \cdot W_c$ for all chains $c \in \mathcal{C}$ and a constant factor $\alpha > 1$, Algorithm 2 has a competitive ratio of at least $\Omega(\frac{\alpha-1}{\alpha} n^{1/3})$ for minimizing the total weighted completion time of jobs with online out-forest precedence constraints on a single machine.*

Proof. Consider an instance consisting of $k = n^{1/3}$ chains with only unit jobs, that is, all jobs have processing times of 1. Each chain c has a predicted weight of $\hat{W}_c = \alpha$ and an actual weight of $W_c = 1$. The first chain consists of $\ell = n^{2/3}$ jobs and has its total weight of 1 on the last job in the chain and all previous jobs have a weight of zero. The remaining $k - 1$ chains each consist of at least $n^{1/3}$ jobs with the total weight of 1 being on the very first job of the respective chain and all other jobs having a weight of zero.

The optimal solution for this instance is to first process the front jobs with weight 1 in an arbitrary order, then process the ℓ jobs of the first chain and finally process all remaining jobs in an arbitrary order. This leads to an objective value of $\mathrm{OPT} = \frac{k \cdot (k-1)}{2} + k + \ell \in \mathcal{O}(n^{2/3})$.

Algorithm 2 starts to process each chain with a rate of $\frac{1}{k}$ until the first jobs of each chain is completed after k time units. Afterwards, the first chain is processed with rate $\frac{\alpha}{(\alpha-1)\cdot(k-1)+\alpha} \leq \frac{\alpha}{(\alpha-1)\cdot k}$, as the algorithm thinks that the other chains still have a remaining weight of $\alpha - 1$. The algorithm processes the first chain at this rate until it completes. This leads to an objective value of $\mathrm{ALG} \geq k \cdot (k-1) + \frac{(\alpha-1)\cdot k}{\alpha} \cdot \ell \geq k \cdot (k-1) + \frac{\alpha-1}{\alpha} \cdot n \in \Omega(\frac{\alpha-1}{\alpha} \cdot n)$.

Putting the bounds together, we get a competitive ratio of $\frac{\mathrm{ALG}}{\mathrm{OPT}} \in \Omega(\frac{\alpha-1}{\alpha} \cdot n^{1/3})$. □

Adaptive Weight Values for Out-Forests. Observation 8.2 states that static weight predictions are not sufficient to obtain $\mathcal{O}(1)$-competitive algorithms for out-forests. The reason is that we, in contrast to chains, cannot recompute $\hat{W}_j$ whenever a new front job j appears. For adaptive predictions, however, we do not need to recompute $\hat{W}_j$, as we simply receive a new prediction. Thus, we can process every front job $j \in F(t)$ with rate $y_j(t) = \hat{W}_j / \sum_{j' \in F(t)} \hat{W}_{j'}$. For correct predictions, Theorem 8.4 directly implies the following.

Theorem 8.7. *Given correct adaptive weight predictions, there exists a non-clairvoyant algorithm for minimizing the total weighted completion time of jobs with online out-forest precedence constraints on a single machine with a competitive ratio of at most* 4.

Full Proof of Theorem 8.4. Fix an algorithm satisfying the conditions of Theorem 8.4. Let ALG be the objective value of the algorithm's schedule for a fixed instance. We introduce a mean busy time linear programming relaxation similar to the one in [Gar+19; SS97] and (LP_1^M) for our problem on a machine running at lower speed $\frac{1}{\kappa}$, for some $\kappa \geq 1$. As the completion time of every job is linear in the machine speed, the optimal objective value of this LP is at most $\kappa \cdot \mathrm{OPT}$. The variable x_{jt} denotes the fractional assignment of job j at time t. The first constraint ensures that every job receives enough amount of processing to complete, the second constraint restricts the available rate per time to $\frac{1}{\kappa}$, and the final constraint asserts that no

job can be completed before its predecessors. It is not hard to see that every integral feasible schedule satisfies these constraints, hence ($\mathrm{LP}_1^c(\kappa)$) is a relaxation.

$$\begin{aligned}
\min \quad & \sum_{j\in J}\sum_{t\geq 0} w_j \cdot t \cdot \frac{x_{jt}}{p_j} && && (\mathrm{LP}_1^c(\kappa))\\
\text{s.t.} \quad & \sum_{t\geq 0} \frac{x_{jt}}{p_j} \geq 1 && \forall j \in J\\
& \sum_{j\in J} \kappa \cdot x_{jt} \leq 1 && \forall t \geq 0\\
& \sum_{s=0}^{t} \frac{x_{js}}{p_j} \geq \sum_{s=0}^{t} \frac{x_{j's}}{p_{j'}} && \forall t \geq 0, \forall (j,j') \in E\\
& x_{jt} \geq 0 && \forall j \in J, \forall t \geq 0
\end{aligned}$$

The dual of ($\mathrm{LP}_1^c(\kappa)$) can be written as follows.

$$\max \quad \sum_{j\in J} a_j - \sum_{t\geq 0} b_t \qquad (\mathrm{DLP}_1^c(\kappa))$$

$$\text{s.t.} \quad \sum_{s\geq t}\left(\sum_{j'\in J:(j,j')\in E} c_{s,j\to j'} - \sum_{j'\in J:(j',j)\in E} c_{s,j'\to j}\right) \leq \kappa b_t p_j - a_j + w_j t \quad \forall j \in J, \forall t \geq 0 \qquad (8.1)$$

$$a_j, b_t, c_{t,j\to j'} \geq 0 \qquad \forall t \geq 0, \forall (j,j') \in E$$

We prove Theorem 8.4 via dual fitting ($\mathrm{DLP}_1^c(\kappa)$). We define an assignment for ($\mathrm{DLP}_1^c(\kappa)$) as follows:

- $\bar{a}_j := \sum_{s\geq 0} \bar{a}_{j,s}$ for every job j, where $\bar{a}_{j,s} := w_j$ if $s \leq C_j$ and $\bar{a}_{j,s} = 0$ otherwise.
- $\bar{b}_t := \frac{1}{2} \cdot W(t)$ for every time t.
- $\bar{c}_{t,j'\to j} := w(S(j))$ if $j, j' \in U(t)$, and $\bar{c}_{t,j'\to j} := 0$ otherwise, for every time t and $(j', j) \in E$.

The following two lemmas show that this assignment is feasible subject to the conditions stated in Theorem 8.4, and that its dual objective value captures a fraction of the algorithm's objective value.

Lemma 8.8. *It holds that* $\sum_{j\in J} \bar{a}_j - \sum_{t\geq 0} \bar{b}_t = \frac{1}{2}\mathrm{ALG}$.

Proof. Note that $\bar{a}_j = w_j C_j$, and thus, $\sum_{j\in J} \bar{a}_j = \mathrm{ALG}$. Also, since the weight w_j of a job j is contained in $W(t)$ if $t \leq C_j$, we conclude $\sum_{t\geq 0} \bar{b}_t = \frac{1}{2}\mathrm{ALG}$. □

Lemma 8.9. *The assignment* $(\bar{a}_j)_j$, $(\bar{b}_t)_t$ *and* $(\bar{c}_{t,j\to j'})_{t,j\to j'}$ *is feasible for* ($\mathrm{DLP}_1^c(\kappa)$) *if* $\kappa = 2\rho$ *and* $w(S(j))/W(t) \leq \rho \cdot y_j(t)$ *for all times* t *and for all* $j \in F(t)$.

Proof. Since the assignment is non-negative by definition, it suffices to show that it satisfies (8.1). Fix a job j and a time $t \geq 0$. By observing that $\bar{a}_j - t \cdot w_j \leq \sum_{s\geq t} \bar{a}_{j,s}$, it suffices to verify

$$\sum_{s\geq t}\left(\bar{a}_{j,s} + \sum_{(j,j')\in E} \bar{c}_{s,j\to j'} - \sum_{(j',j)\in E} \bar{c}_{s,j'\to j}\right) \leq \kappa \bar{b}_t p_j \,. \qquad (8.2)$$

To this end, we consider the terms on the left side for all times $s \geq t$ separately. For any s with $s > C_j$, the left side of (8.2) is zero, because $\bar{a}_{j,s} = 0$ and $j \notin U(s)$.

Otherwise, if $s \leq C_j$, let t_j^* be the first point in time after t when j is available, and let $s \in [0, t_j^*)$. Then, $j \in U(s)$, and since each vertex in an out-forest has at most one direct predecessor, there must be a unique job $j_1 \in U(s)$ with $(j_1, j) \in E$. Thus, $\bar{c}_{s,j_1 \to j} = w(S(j))$ and $\bar{c}_{s,j \to j'} = w(S(j'))$ for all $(j, j') \in E$. Observe that in out-forests, we have $S(j') \cap S(j'') = \emptyset$ for all $j' \neq j''$ with $(j, j'), (j, j'') \in E$. This implies $\sum_{(j,j') \in E} \bar{c}_{s,j \to j'} = w(S(j)) - w_j$ and $\sum_{(j,j') \in E} \bar{c}_{s,j \to j'} - \sum_{(j_1,j) \in E} \bar{c}_{s,j_1 \to j} = -w_j$. Hence,

$$\bar{a}_{j,s} + \sum_{(j,j') \in E} \bar{c}_{s,j \to j'} - \sum_{(j',j) \in E} \bar{c}_{s,j' \to j} \leq w_j - w_j = 0 \,.$$

Therefore, proving (8.2) reduces to proving

$$\sum_{s=t_j^*}^{C_j} \left(w_j + \sum_{(j,j') \in E} \bar{c}_{s,j \to j'} - \sum_{(j',j) \in E} \bar{c}_{s,j' \to j} \right) \leq \kappa \bar{b}_t p_j \,. \tag{8.3}$$

Let $s \in [t_j^*, C_j)$. There cannot be an unfinished job preceding j, hence $\sum_{(j',j) \in E} \bar{c}_{s,j' \to j} = 0$. Observe that if there is a job $j' \in U(s)$ with $(j, j') \in E$, then $j \in U(s)$ implies $j' \in U(s)$ and, thus $\bar{c}_{s,j \to j'} = w(S(j'))$ by definition. Using again that $S(j')$ are pairwise disjoint for all direct successors j' of j, that is, for all $(j, j') \in E$, this yields $\sum_{(j,j') \in E} \bar{c}_{s,j \to j'} = w(S(j)) - w_j$, and further gives

$$w_j + \sum_{(j,j') \in E} \bar{c}_{s,j \to j'} - \sum_{(j',j) \in E} \bar{c}_{s,j' \to j} = w(S(j)) \,.$$

Thus, the left side of (8.3) is equal to $\sum_{s=t_j^*}^{C_j} w(S(j))$. Furthermore, $W(t_1) \geq W(t_2)$ for all $t_1 \leq t_2$ and that j is processed by $y_j(t')$ units at any time $t' \in [t_j^*, C_j]$ combined with the assumption $w(S(j))/W(t) \leq \rho \cdot y_j(t)$ imply the following:

$$\sum_{s=t_j^*}^{C_j} \frac{w(S(j))}{W(t)} \leq \sum_{s=t_j^*}^{C_j} \frac{w(S(j))}{W(s)} \leq \sum_{s=t_j^*}^{C_j} \rho \cdot y_j(s) \leq \rho \cdot p_j \,.$$

Rearranging it, using the definition of $\bar{b}_t$ and $\kappa = 2\rho$ gives

$$\sum_{s=t_j^*}^{C_j} w(S(j)) \leq \rho \cdot p_j \cdot W(t) = 2\rho \cdot p_j \cdot \bar{b}_t = \kappa \cdot p_j \cdot \bar{b}_t \,,$$

which implies (8.3), and thus proves the statement. □

Proof of Theorem 8.4. We set $\kappa = 2\rho$. Weak duality, Lemma 8.9, and Lemma 8.8 imply

$$2\rho \cdot \mathrm{OPT} \geq \sum_{j \in J} \bar{a}_j - \sum_{t \geq 0} \bar{b}_t = \frac{1}{2} \cdot \mathrm{ALG} \,,$$

which concludes that $\mathrm{ALG} \leq 4\rho \cdot \mathrm{OPT}$. □

8.3.2 Learning-Augmented Algorithms

In this section, we extend the algorithms presented in Section 8.3.1 to achieve a smooth error-dependency in the case of inaccurate predictions, while preserving constant consistency. Further, we use Preferential Time Sharing (cf. Section 8.2) to ensure a robustness of $O(\omega)$.

Static Weight Predictions for Chains. Here, the main challenges are as follows: we only have access to the potentially wrong predictions $\hat{W}_c$ on the total chain weight for all $c \in \mathcal{C}$ and, therefore, we execute Algorithm 2 using $\hat{W}_c$ instead of W_c. In particular, the weight of a chain c might be *underpredicted*, $\hat{W}_c < W_c$, or *overpredicted*, $\hat{W}_c > W_c$. This means that $\sum_c \hat{W}_c$ may not be the accurate total weight of the instance and that the recomputation of $W_c(t)$ in Line 4 may be inaccurate. We show how to encode the error due to underpredicted chains in an instance $\mathcal{C}_u$ and the error due to overpredicted chains in an instance $\mathcal{C}_o$, similar to an error-dependency proposed for online set cover [BMS20]. In particular, we prove the following result.

Theorem 8.10. *For minimizing the total weighted completion time of jobs with online chain precedence constraints on a single machine, there is a non-clairvoyant algorithm with predicted chain weights with a competitive ratio of at most*

$$\mathcal{O}(1) \cdot \min\left\{1 + \frac{\mathrm{OPT}(\mathcal{C}_o) + \omega \cdot \mathrm{OPT}(\mathcal{C}_u)}{\mathrm{OPT}}, \omega\right\}.$$

In order to give the proof, we formally define the *predicted instance* (including $\mathcal{C}_o$ and $\mathcal{C}_u$).

Definition 8.11 (predicted instances)**.** The *predicted instance* $\hat{\mathcal{C}}$, *underpredicted subinstance* $\mathcal{C}_u$, and *overpredicted subinstance* $\mathcal{C}_o$ are constructed by considering for every $c = [j_1, \dots, j_\ell] \in \mathcal{C}$ the following cases:

(i) If $\hat{W}_c = W_c$, then the chain $\hat{c} = c$ with job weights $\hat{w}_j = w_j$ for all $j \in c$ is added to $\hat{\mathcal{C}}$.

(ii) If $\hat{W}_c < W_c$, then the chain $\hat{c} = [j_1, \dots, j_k]$, where k is the smallest index s.t. $\hat{W}_c \leq \sum_{i=1}^{k} w_i$, with weights $\hat{w}_{j_i} = w_{j_i}$ for all $1 \leq i \leq k-1$ and $\hat{w}_{j_k} = \hat{W}_c - \sum_{i=1}^{k-1} w_i$ is added to $\hat{\mathcal{C}}$. Additionally, a chain $c_u = [\perp, j_{k+1}, \dots, j_\ell]$ with weights $\hat{w}_{j_i} = w_{j_i}$ for all $k+1 \leq i \leq \ell$ and $\hat{w}_\perp = \sum_{i=1}^{k} w_i - \hat{W}_c$ is added to $\mathcal{C}_u$, where the processing requirement of $\perp$ is equal to the total processing requirement of $\hat{c}$.

(iii) If $\hat{W}_c > W_c$, then chain $\hat{c} = [j_1, \dots, j_\ell]$ with weights $\hat{w}_{j_i} = w_{j_i}$ for all $1 \leq i \leq \ell - 1$ and $\hat{w}_{j_\ell} = \hat{W}_c - \sum_{i=1}^{\ell-1} w_i$ is added to $\hat{\mathcal{C}}$. Additionally, a chain $c_o = [\top]$ is added to $\mathcal{C}_o$, where the weight of $\top$ is equal to $\hat{W}_c - \sum_{i=1}^{\ell} w_i$ and its processing requirement is equal to the total processing requirement of $\hat{c}$.

Finally, $\mathcal{C}_p$ is a copy of $\hat{\mathcal{C}}$ where for every overpredicted chain $\hat{c} = [j_1, \dots, j_\ell] \in \hat{\mathcal{C}}$ the weight of its last job j_ℓ is set to w_{j_ℓ}, the weight of the job in the actual instance. This weight is strictly smaller than the weight $\hat{w}_{j_\ell} = \hat{W}_c - \sum_{i=1}^{\ell-1} w_i$ of the job in instance $\hat{\mathcal{C}}$.

Note that for every $\hat{c} \in \hat{\mathcal{C}}$, we have $\sum_{j \in \hat{c}} \hat{w}_j = \hat{W}_c$, that is, the predicted weights are correct for $\hat{\mathcal{C}}$. In the following, we nevertheless call a chain $\hat{c} \in \hat{\mathcal{C}}$ overpredicted and underpredicted if that is true for its corresponding chain in $\mathcal{C}$, respectively. Since every job j of $\mathcal{C}_p$ is also part

of $\mathcal{C}$ with the same processing requirement and a weight of at most w_j and the chains in $\mathcal{C}_p$ are prefixes of the chains in $\mathcal{C}$, we conclude the following bound.

Proposition 8.12. $\mathrm{OPT}(\mathcal{C}_p) \leq \mathrm{OPT}(\mathcal{C})$.

We use the algorithm of Proposition 8.1 in combination with Preferential Time Sharing of Theorem 7.4 to define Algorithm 3.

Algorithm 3: Preferential Time Sharing on Chains

1 Execute Algorithm 2 with rate $\frac{1}{2}$ using the predicted chain weights.
2 Execute the algorithm of Proposition 8.1 with rate $\frac{1}{2}$.

Lemma 8.13. *Algorithm 3 with predicted chain weights achieves an objective value of at most*

$$\mathcal{O}(1) \cdot \mathrm{OPT}(\mathcal{C}_p) + \mathcal{O}(1) \cdot (\mathrm{OPT}(\mathcal{C}_o) + \omega \cdot \mathrm{OPT}(\mathcal{C}_u)) \,.$$

Proof. We first argue that $\mathrm{OPT}(\hat{\mathcal{C}}) \leq \mathcal{O}(1) \cdot \mathrm{OPT}(\mathcal{C}_p) + \mathcal{O}(1) \cdot \mathrm{OPT}(\mathcal{C}_o)$. To this end, consider the instance $\mathcal{C}_p \cup \mathcal{C}_o$. Every correctly predicted or underpredicted chain in $\hat{\mathcal{C}}$ is contained as an identical copy in $\mathcal{C}_p$. For every overpredicted chain $\hat{c} \in \hat{\mathcal{C}}$ with weight $\hat{W}_{\hat{c}}$ in $\hat{\mathcal{C}}$, all jobs of $\hat{c}$ are contained with a total weight of $W_{\hat{c}}$ in $\mathcal{C}_p$ and the remaining weight of $\hat{W}_{\hat{c}} - W_{\hat{c}}$ is contained in $\mathcal{C}_o$. Additionally, it is ensured by the processing requirement of the jobs in $\mathcal{C}_o$ that their weight can only be gained when processing at least the total processing requirement of $\hat{c}$. This implies that the time to gain weight $\hat{W}_{\hat{c}} - W_{\hat{c}}$ of every overpredicted chain $\hat{c} \in \hat{\mathcal{C}}$ in $\mathcal{C}_p \cup \mathcal{C}_o$ takes at least as long as in $\hat{\mathcal{C}}$, and thus, $\mathrm{OPT}(\hat{\mathcal{C}}) \leq \mathrm{OPT}(\mathcal{C}_p \cup \mathcal{C}_o)$. Finally, it is not hard to see that $\mathrm{OPT}(\mathcal{C}_p \cup \mathcal{C}_o) \leq 2 \cdot \mathrm{OPT}(\mathcal{C}_p) + 2 \cdot \mathrm{OPT}(\mathcal{C}_o)$ as $\mathrm{OPT}(\mathcal{C}_p)$ and $\mathrm{OPT}(\mathcal{C}_o)$ can be executed in parallel by preemptively sharing the machine, yielding the claimed bound.

We now show that $\mathrm{ALG} \leq \mathcal{O}(1) \cdot (\mathrm{OPT}(\hat{\mathcal{C}}) + \omega \cdot \mathrm{OPT}(\mathcal{C}_u))$ for Algorithm 3, which implies the statement. First consider the execution of Algorithm 2 in the first line. We may assume that the algorithm processes the artificial job added to each overpredicted chain in $\hat{\mathcal{C}}$, as it only increases its objective. Further, Algorithm 2 stops processing an underpredicted chain $c \in \mathcal{C}$ when a total weight of $\hat{W}_c$ has been completed on c and finishes them at the very end of the schedule. This concludes that the total objective of Algorithm 2 *without* the weighted completion times of the jobs that are processed at the very end in arbitrary order is at most $\mathcal{O}(1) \cdot \mathrm{OPT}(\hat{\mathcal{C}})$. But, due to Proposition 8.1 and line two of Algorithm 3, we conclude that Algorithm 3 always processes such chains that are completed at the very end of Algorithm 2 with a rate of at least $\frac{1}{2\omega}$, and thus delays the completion the jobs in these chains by a factor of at most 2ω compared to an optimal solution. By observing that the total weight of such jobs is exactly equal to the total weight of chains in $\mathcal{C}_u$ and that the jobs in a chain $c \in \mathcal{C}_u$ can only be processed after time equal to the total processing requirement of the corresponding chain in $\hat{\mathcal{C}}$, we conclude the stated bound. □

Adaptive Weight Predictions for Out-Forests. To capture the quality of an adaptive prediction, we intuitively need to measure its quality over the whole execution. To this end, we use the maximal distortion factor of the weight predictions of every possible front job, which, in fact, can be any job in J.

Algorithm 4: Weighted-Round-Robin on out-forests

Input: Out-forest and adaptive weight predictions.

1 **while** *there are unfinished jobs* **do**

2 Process every $j \in F(t)$ with rate $y_j(t) = \hat{W}_j / \sum_{i \in F(t)} \hat{W}_i$.

3 $t \leftarrow t^+$.

Theorem 8.14. *For minimizing the total weighted completion time on a single machine with online out-forest precedence constraint and adaptive weight predictions, there is a non-clairvoyant algorithm with a competitive ratio of at most*

$$\mathcal{O}(1) \cdot \min\left\{ \max_{v \in J} \frac{\hat{W}_v}{w(S(v))} \cdot \max_{v \in J} \frac{w(S(v))}{\hat{W}_v}, \omega \right\} .$$

Proof. To prove the theorem, we show that Algorithm 4 is $\mathcal{O}(\eta)$-competitive for

$$\eta := \max_{v \in J} \frac{\hat{W}_v}{W_v} \cdot \max_{v \in J} \frac{W_v}{\hat{W}_v} .$$

Then, Proposition 8.1 and Theorem 7.4 imply the theorem.

By Theorem 8.4, it suffices to show that $w(S(j))/W(t) \leq \eta \cdot y_j(t)$ holds for any point in time t during the execution of Algorithm 4 and any $j \in F(t)$, where $y_j(t)$ is the rate with which the algorithm processes j at point in time t. Consider a fixed point in time t and an arbitrary $j \in F(t)$. Then, the algorithm processes j with rate $y_j(t) = \hat{W}_j / \sum_{i \in F(t)} \hat{W}_i$ by definition. We can conclude

$$\begin{aligned} \frac{w(S(j))}{W(t)} &= \frac{w(S(j))}{\sum_{i \in F(t)} w(S(i))} = \frac{\hat{W}_j \cdot \frac{w(S(j))}{\hat{W}_j}}{\sum_{i \in F(t)} \hat{W}_i \cdot \frac{w(S(i))}{\hat{W}_i}} \leq \frac{\left(\max_{v \in J} \frac{w(S(v))}{\hat{W}_v}\right) \cdot \hat{W}_j}{\left(\min_{v \in J} \frac{w(S(v))}{\hat{W}_v}\right) \cdot \sum_{i \in F(t)} \hat{W}_i} \\ &= \max_{v \in J} \frac{\hat{W}_v}{w(S(v))} \cdot \max_{v \in J} \frac{w(S(v))}{\hat{W}_v} \cdot \frac{\hat{W}_j}{\sum_{i \in F(t)} \hat{W}_i} = \eta \cdot y_j(t) . \end{aligned}$$

This concludes the proof of the theorem. □

8.4 Weight Order Predictions

Next we consider static and adaptive weight order predictions. As strong lower bounds hold for in-trees, even for the more powerful adaptive weight predictions (cf. Lemma 8.3), we focus on chains and out-forest precedence constraints.

Further, we introduce an error measure for wrongly predicted orders. A natural function on orders is the *largest inversion*, that is, the maximum distance between the position of a front job in an order prediction $\hat{\preceq}_t$ and the true order $\preceq_t$. However, if all out-trees have almost the same weight, just perturbed by some small constant, this function indicates a large error for the reverse order, although it will arguably perform nearly as good as the true order. To mitigate

this overestimation, we first introduce ε-approximate inversions. Formally, for every precision constant $\varepsilon > 0$, we define

$$\mathcal{L}(\varepsilon) \coloneqq \max_{t, j \in F(t)} \left| \left\{ i \in F(t) \;\middle|\; \frac{w(S(j))}{1+\varepsilon} \geq w(S(i)) \wedge i \hat{\preceq}_t j \right\} \right| .$$

Note that $\mathcal{L}(\varepsilon) \geq 1$ for every $\varepsilon > 0$, because $\hat{\preceq}_t$ is reflexive. We define the *ε-approximate largest inversion* error as $\max\{1+\varepsilon, \mathcal{L}(\varepsilon)\}$. We show performance guarantees depending on this error, which hold for any $\varepsilon > 0$. Therefore, we intuitively get a Pareto frontier between the precision $1+\varepsilon$ and $\mathcal{L}(\varepsilon)$, the largest distance of inversions that are worse than the precision. A configurable error with such properties has been applied to other learning-augmented algorithms, for example, in [APT22a] or in Chapter 10.

8.4.1 Adaptive Weight Order

We introduce Algorithm 5, which exploits access to the adaptive order $\hat{\preceq}_t$. In a sense, the idea of the algorithm is to emulate Algorithm 2 for weight predictions. Instead of having access to the total remaining weight of every out-tree to computing rates, Algorithm 5 uses $\hat{\preceq}_t$ to approximate the rates. For every front job $j \in F(t)$, let i_j be the position of j in $\hat{\preceq}_t$. Recall that H_k denotes the kth harmonic number.

Algorithm 5: Adaptive weight order algorithm

Input: Adaptive weight order $\hat{\preceq}$.

```
t ← 0
while there are unfinished jobs do
    Process every j ∈ F(t) with rate y_j(t) = 1/(H_{|F(t)|} · i_j), where i_j is the position of j
      in ≾̂_t.
    t ← t^+
```

Theorem 8.15. *For any $\varepsilon > 0$, Algorithm 5 has a competitive ratio of at most $4H_\omega \cdot \max\{1+\varepsilon, \mathcal{L}(\varepsilon)\}$ for minimizing the total weighted completion time on a single machine with online out-forest precedence constraints.*

Proof. First note that the rates are feasible, because $\sum_{j \in F(t)} \frac{1}{H_{|F(t)|} \cdot i_j} = \frac{H_{|F(t)|}}{H_{|F(t)|}} = 1$.

Fix a time t and an $\varepsilon > 0$. Assume that $j_1 \hat{\preceq}_t \ldots \hat{\preceq}_t j_{|F(t)|}$, and fix a front job $j_i \in F(t)$. The algorithm processes j_i at time t with rate $q_{j_i t} = 1/(H_{|F(t)|} \cdot i) \geq 1/(H_\omega \cdot i)$. Note that showing $w(S(j_i))/W(t) \leq H_\omega \cdot \max\{1+\varepsilon, \mathcal{L}(\varepsilon)\} \cdot y_{j_i}(t)$ implies the theorem via Theorem 8.4. Assume otherwise, that is, $w(S(j_i))/W(t) > \frac{1}{i} \cdot \max\{1+\varepsilon, \mathcal{L}(\varepsilon)\}$. For the sake of readability, we define $K_> = \{k \in [i-1] \mid w(S(i_k)) > \frac{w(S(j_i))}{1+\varepsilon}\}$ and $K_\leq = \{k \in [i] \mid w(S(i_k)) \leq \frac{w(S(j_i))}{1+\varepsilon}\}$. Since in an out-forest the sets $S(j)$ are pairwise disjoint for all front jobs $j \in F(t)$,

$$1 \geq \sum_{k \in [i]} \frac{w(S(i_k))}{W(t)} \geq \sum_{k \in K_>} \frac{w(S(i_k))}{W(t)} + \sum_{k \in K_\leq} \frac{w(S(i_k))}{W(t)} .$$

Consider the second sum. First, observe that this sum has at most $\mathcal{L}(\varepsilon)$ many terms, including the one for j_i, and that each such term is at most $w(S(j_i))/W(t)$. Then, observe that every

term in the first sum is at least $w(S(j_i))/((1+\varepsilon)W(t))$. Thus, we can further lower bound the sum of the two sums by

$$\frac{1}{1+\varepsilon}\sum_{k\in K_>}\frac{w(S(j_i))}{W(t)}+\frac{1}{\mathcal{L}(\varepsilon)}\sum_{k\in K_\leq}\frac{w(S(j_i))}{W(t)}$$
$$\geq\frac{1}{\max\{1+\varepsilon,\mathcal{L}(\varepsilon)\}}\sum_{k\in[i]}\frac{w(S(j_i))}{W(t)}>\sum_{k=1}^{i}\frac{1}{i}=1\,.$$

This is a contradiction. □

Since $\omega \leq n$, we conclude the following corollary.

Corollary 8.16. *There exists a non-clairvoyant weight-oblivious algorithm for the problem of minimizing the total weighted completion time of n jobs on a single machine with a competitive ratio of at most $O(\log n)$ when given access to the order of the jobs weights.*

8.4.2 Static Weight Order

If we only have access to $\hat{\preceq}_0$, a natural approach would be to compute the initial rates as used in Algorithm 5 and just not update them. As Observation 8.2 rules out well-performing algorithms for out-trees, we focus on chains. Even for chains, we show that this algorithm has a competitive ratio of at least $\Omega(\omega \cdot H_\omega)$.

Lemma 8.17. *The variant of Algorithm 5 that computes the rates using $\hat{\preceq}_0$ instead of $\hat{\preceq}_t$ is at least $\Omega(\omega \cdot H_\omega)$-competitive, even if $\hat{\preceq}_0$ equals $\preceq_0$.*

Proof. Consider an instance with ω chains, each with a total weight of one. Then, $\preceq_0$ is just an arbitrary order of the chains. Recall that the algorithm starts processing the chains c with rate $1/(H_\omega \cdot i_c)$, where i_c is the position of c in the order. We define the first $\omega - 1$ chains to have their total weight of one at the very first job and afterwards only jobs of weight zero. Chain ω, the slowest chain, has its total weight on the last job. We define the chains c to contain a total of $d \cdot H_\omega \cdot i_c$ jobs with unit processing times, for some common integer d. This means that the algorithm finishes all chains at the same time. The optimal solution value for this instance is $\omega \cdot (\omega+1) + \omega - 1 + d \cdot H_\omega \cdot \omega$, where $\omega \cdot (\omega+1)$ is the optimal sum of completion times for the first $\omega - 1$ chains, $d \cdot H_\omega \cdot \omega$ is the cost for processing the last chain, and $\omega - 1$ is the cost for delaying the last chains by the $\omega - 1$ time units needed to process the first jobs of the first $\omega - 1$ chains. The solution value of the algorithm is at least $d \cdot H_\omega^2 \cdot \omega^2$ as this is the cost for just processing the last chain. Thus, for large d, the competitive ratio tends to $H_\omega \cdot \omega$. □

However, the lower bound instance of the lemma requires ω to be "small" compared to the number of jobs, in case of unit jobs, or to $P := \sum_{j\in J} p_j$, otherwise. We exploit this to prove the following theorem.

Theorem 8.18. *For any $\varepsilon > 0$, Algorithm 5 has a competitive ratio of at most $\mathcal{O}(H_\omega^2\sqrt{P} \cdot \max\{1+\varepsilon, \mathcal{L}(\varepsilon)\})$ when computing rates with $\hat{\preceq}_0$ instead of $\hat{\preceq}_t$ at any time t. For unit size jobs, it is $\mathcal{O}(H_\omega^2\sqrt{n} \cdot \max\{1+\varepsilon, \mathcal{L}(\varepsilon)\})$-competitive.*

Proof. Recall that P denotes the sum over all job processing times in the instance. For a subset of chains S, let $\mathrm{OPT}(S)$ denote the optimal objective value for the subinstance induced by S. For a single chain c, $\mathrm{OPT}(c)$ is just the cost for processing chain c with rate 1 on a single machine. Clearly, $\mathrm{OPT}(S) \geq \sum_{c \in S} \mathrm{OPT}(c)$. Let $\mathrm{ALG}(S)$ denote the sum of weighted completion times of the jobs that belong to chains in S in the schedule computed by the algorithm.

In the first part of the proof, we assume that all chains c_i have a total processing time of at most $\sqrt{P}$. This only decreases the objective value of OPT. For ALG, we will analyze the additional cost caused by longer chains afterwards. In a sense, we assume that ALG, for each chain c, has to pay all weight that appears after $\sqrt{P}$ processing times units of the chain two times: Once artificially after exactly $\sqrt{P}$ time units of the chain have been processed and once at the point during the processing where the weight actually appears. This assumption clearly only increases ALG. In the first part of the proof, we analyze only the artificial cost for such weights and ignore the actual cost. In the context of our algorithm, this is equivalent to assuming the chains have total processing times of at most $\sqrt{P}$. In the second part of the proof, we will analyze the actual cost for the jobs that appear after $\sqrt{P}$ time units in their chain.

First Part. Assume $c_1 \hat{\preceq}_0 c_2 \hat{\preceq}_0 \dots \hat{\preceq}_0 c_\omega$. Therefore, the algorithm processes chain c_i with rate $1/(H_\omega \cdot i)$. This directly implies $\mathrm{ALG}(c_i) = H_\omega \cdot i \cdot \mathrm{OPT}(c_i)$, and thus,

$$\mathrm{ALG} = \sum_{i=1}^{\omega} H_\omega \cdot i \cdot \mathrm{OPT}(c_i) \ .$$

Let $\mathcal{C}_k = \{c_1, \dots, c_k\}$ for every $k \in [\omega]$. We first analyze $\mathrm{ALG}(\mathcal{C}_{3 \cdot \mathcal{L}(\varepsilon)})$. For the chains in $\mathcal{C}_{3 \cdot \mathcal{L}(\varepsilon)}$, we get

$$\mathrm{ALG}(\mathcal{C}_{3 \cdot \mathcal{L}(\varepsilon)}) = \sum_{i=1}^{3 \cdot \mathcal{L}(\varepsilon)} H_\omega \cdot i \cdot \mathrm{OPT}(c_i) \leq H_\omega \cdot 3 \cdot \mathcal{L}(\varepsilon) \sum_{i=1}^{3 \cdot \mathcal{L}(\varepsilon)} \mathrm{OPT}(c_i) \leq 3 H_\omega \mathcal{L}(\varepsilon) \mathrm{OPT} \, ,$$

meaning that, for $\mathcal{C}_{3 \cdot \mathcal{L}(\varepsilon)}$, we achieve the desired competitive ratio.

Next, consider the chains in $\mathcal{C} \setminus \mathcal{C}_{3 \cdot \mathcal{L}(\varepsilon)}$, that is, the chains c_i with $i > 3 \cdot \mathcal{L}(\varepsilon)$. To analyze the cost for these chains C_i, we continue by lower bounding $\mathrm{OPT}(c_i)$. To that end, consider $\mathrm{OPT}(\mathcal{C}_i)$. The definition of $\mathcal{L}(\varepsilon)$ implies that there are at most $\mathcal{L}(\varepsilon)$ chains $c_j \in \mathcal{C}_i$ with $W_{c_i} \geq (1+\varepsilon) W_{c_j}$. For all other chains c_j in $\mathcal{C}_i$, we have $\frac{W_{c_i}}{1+\varepsilon} < W_{c_j}$. Thus, there are $i - \mathcal{L}(\varepsilon)$ chains in $\mathcal{C}_i$ with a weight of at least $\frac{W_{c_i}}{1+\varepsilon}$. Since we consider chains with $i > 3 \cdot \mathcal{L}(\varepsilon)$, it holds $i - \mathcal{L}(\varepsilon) \geq 1$. We can lower bound $\mathrm{OPT}(\mathcal{C}_i)$ by assuming that all such chains consist only of a single job with weight $\frac{W_{c_i}}{1+\varepsilon}$ and ignoring the up-to $\mathcal{L}(\varepsilon)$ other chains. These assumptions only decrease $\mathrm{OPT}(\mathcal{C}_i)$. Since in this relaxation all jobs have an equal weight and length, an optimal solution for it processes the jobs in an arbitrary order, giving

$$\begin{aligned} \mathrm{OPT}(\mathcal{C}_i) &\geq \sum_{j=1}^{i-\mathcal{L}(\varepsilon)} j \cdot \frac{W_{c_i}}{1+\varepsilon} = \frac{(i - \mathcal{L}(\varepsilon) + 1) \cdot (i - \mathcal{L}(\varepsilon)) \cdot W_{c_i}}{2 \cdot (1+\varepsilon)} \\ &= \frac{((i+1) \cdot i + \mathcal{L}(\varepsilon)^2 - 2 \cdot i \cdot \mathcal{L}(\varepsilon) - \mathcal{L}(\varepsilon)) \cdot W_{c_i}}{2 \cdot (1+\varepsilon)} \geq \frac{(i+1) \cdot i - 3 \cdot i \cdot \mathcal{L}(\varepsilon)}{2 \cdot (1+\varepsilon)} \cdot W_{c_i} \ . \end{aligned}$$

Since we still assume that each chain has a total processing time of at most $\sqrt{P}$, we can observe $\mathrm{OPT}(c_i) \leq \sqrt{P} \cdot W_{c_i}$. This yields:

$$\frac{2 \cdot (1+\varepsilon)}{(i+1) \cdot i - 3 \cdot i \cdot \mathcal{L}(\varepsilon)} \cdot \sqrt{P} \cdot \mathrm{OPT}(\mathcal{C}_i) \geq \mathrm{OPT}(c_i) \; .$$

We can therefore conclude that

$$\begin{aligned}
\mathrm{ALG}(\mathcal{C} \setminus \mathcal{C}_{3 \cdot \mathcal{L}(\varepsilon)}) &= \sum_{i=3 \cdot \mathcal{L}(\varepsilon)+1}^{\omega} H_\omega \cdot i \cdot \mathrm{OPT}(c_i) \\
&\leq \sum_{i=3 \cdot \mathcal{L}(\varepsilon)+1}^{\omega} H_\omega \cdot \frac{2 \cdot (1+\varepsilon)}{(i+1) - 3 \cdot \mathcal{L}(\varepsilon)} \cdot \sqrt{P} \cdot \mathrm{OPT}(\mathcal{C}_i) \\
&\leq 2 \cdot (1+\varepsilon) \cdot H_\omega \cdot \sqrt{P} \cdot \mathrm{OPT} \sum_{i=3 \cdot \mathcal{L}(\varepsilon)+1}^{\omega} \frac{1}{(i+1) - 3 \cdot \mathcal{L}(\varepsilon)} \\
&\leq 2 \cdot (1+\varepsilon) \cdot H_\omega \cdot H_{\omega - 3\mathcal{L}(\varepsilon)+1} \cdot \sqrt{P} \cdot \mathrm{OPT} \leq 2 \cdot (1+\varepsilon) \cdot H_\omega^2 \cdot \sqrt{P} \cdot \mathrm{OPT} \; .
\end{aligned}$$

We finish the first part by combining the bounds on $\mathrm{ALG}(\mathcal{C} \setminus \mathcal{C}_{3 \cdot \mathcal{L}(\varepsilon)})$ and $\mathrm{ALG}(\mathcal{C}_{3 \cdot \mathcal{L}(\varepsilon)})$:

$$\begin{aligned}
\mathrm{ALG}(\mathcal{C}) &= \mathrm{ALG}(\mathcal{C} \setminus \mathcal{C}_{3 \cdot \mathcal{L}(\varepsilon)}) + \mathrm{ALG}(\mathcal{C}_{3 \cdot \mathcal{L}(\varepsilon)}) \\
&\leq 5 \cdot H_\omega^2 \cdot \sqrt{P} \cdot \max\{1+\varepsilon, \mathcal{L}(\varepsilon)\} \cdot \mathrm{OPT} \; .
\end{aligned}$$

Second Part. It remains to analyze the additional cost incurred by chains with a total processing time of more than $\sqrt{P}$. To that end, consider the set J_L of jobs that, in any schedule, cannot be started before $\sqrt{P}$ time units have past. For a job $j \in J_L$, the predecessors of j in the chain of j must have a total processing time of at least $\sqrt{P}$.

Let $\mathrm{ALG}(J_L)$ and $\mathrm{OPT}(J_L)$ denote the weighted completion times of the jobs in J_L in the optimal solution and the schedule computed by ALG, respectively. Then,

$$\frac{\mathrm{ALG}(J_L)}{\mathrm{OPT}(J_L)} \leq \frac{\sum_{j \in J_L} P \cdot w_j}{\sum_{j \in J_L} \sqrt{P} \cdot w_j} = \sqrt{P} \; .$$

Thus, the additional cost of the jobs in J_L asymptotically does not worsen the competitive ratio. This concludes the proof of the theorem. □

8.5 Average Predictions

Recall that average predictions give access to predicted values $\hat{a}_v$ on

$$a(S(v)) = \frac{\sum_{u \in S(v)} w_u}{\sum_{u \in S(v)} p_u}$$

for each $v \in F(t)$. We prove the following lower bound for chains with unit jobs, where average predictions coincide with the average weight of the jobs in the respective chain. The lower

bound exploits that we can append jobs of weight zero to a chain in order to manipulate the average weight of the chain until all chains have the same average.

Lemma 8.19. *Any algorithm that has only access to correct adaptive average predictions is at least $\Omega(\sqrt{n})$-competitive, even for chain precedence constraints with unit-sized jobs.*

Proof. Consider an instance composed of $\sqrt{n} \in \mathbb{N}$ chains of unit jobs, where the first two jobs of the first chain have weights 1 and $n - \sqrt{n}$, respectively, followed by $n - \sqrt{n} - 1$ zero weight jobs. The other $\sqrt{n} - 1$ chains are single jobs with weight 1. For an algorithm, all chains look identical since the first jobs have weight 1 and the average of every chain is equal to 1. Therefore, an adversary can ensure that the algorithm processes the first chain last, giving an objective value of $\sum_{i=1}^{\sqrt{n}} i + (\sqrt{n}+1)(n-\sqrt{n}) = \Omega(n\sqrt{n})$, while a solution that schedules the heavy weight job initially achieves an objective value of at most $1 + 2(n-\sqrt{n}) + \sum_{i=1}^{\sqrt{n}-1}(3+i) = O(n)$. The adaptivity of the predictions does not help for this lower bound as the algorithm would only receive meaningful updates once it finishes the first job of the first chain, which is too late. □

8.6 Action Predictions

We now turn our focus to *general* prediction models, which are not specifically tailored for our concrete problem. Action predictions induce an optimal algorithm. Hence, following accurate predictions clearly results in an optimal solution. To define an error measure for erroneous static and adaptive action predictions, let $\hat{\sigma} : J \to [n]$ be the order in which a fixed static or adaptive action prediction suggests to process jobs. In case of static action predictions, we receive the predicted order initially, meaning that it might predict a set of jobs $\hat{J}$ different to the actual J. During the analysis, we can simply remove the jobs $\hat{J} \setminus J$ from $\hat{\sigma}$ as they do not have any effect on the schedule for the actual instance. For the jobs in $J \setminus \hat{J}$, we define the static action prediction algorithm to just append them to the end of the order $\hat{\sigma}$ once they are revealed. Thus, we can still treat $\hat{\sigma}$ as a function from J to $[n]$. We analyze an algorithm that follows a static or adaptive action prediction using the permutation error from Chapter 7. To this end, let $\sigma : J \to [n]$ be the order of a fixed optimal solution for instance J, and $\mathfrak{I}(J, \hat{\sigma}) = \{(j', j) \in J^2 \mid \sigma(j') < \sigma(j) \wedge \hat{\sigma}(j') > \hat{\sigma}(j)\}$ be the set of inversions between the permutations σ and $\hat{\sigma}$. Now using the same arguments as in the proof of Lemma 7.5 and applying Theorem 7.4 implies the following theorem.

Theorem 8.20. *Given static or adaptive action predictions, there exists an $O(\min\left\{1 + \frac{\eta}{\mathrm{OPT}}, \omega\right\})$-competitive non-clairvoyant algorithm for minimizing the total weighted completion time on a single machine with online precedence constraints, where $\eta = \sum_{(j',j)\in\mathfrak{I}(J,\hat{\sigma})}(w_{j'}p_j - w_j p_{j'})$.*

8.7 Full Input Predictions

We can use full input predictions to compute static action predictions $\hat{\sigma}$. In general, computing $\hat{\sigma}$ requires exponential running time as the problem is NP-hard. For special cases, for example, chains, there are efficient algorithms [Law78].

While following $\hat{\sigma}$ allows us to achieve the guarantee of Theorem 8.20, the error η does not directly depend on the predicted input, but on an algorithm that computes actions for

that input. Thus, we aim at designing error measures depending directly on the "similarity" between the predicted and actual instance. As describing the similarity between two graphs is a notoriously difficult problem on its own, we leave open whether there is a meaningful error for general topologies. However, we give an error measure for chains. The key idea of this error is to capture additional cost that any algorithm pays due to both, *absent predicted* weights and *unexpected actual* weights. Assuming that the predicted and actual instance only differ in the weights, our error $\Lambda = \Gamma_u + \Gamma_a$ considers the optimal objective values Γ_u and Γ_a for the problem instances that use $((w_j - \hat{w}_j)_+)_j$ and $((\hat{w}_j - w_j)_+)_j$ as weights, respectively. Then, Γ_u and Γ_a measure the cost for *unexpected* and *absent* weights. We generalize this idea to also capture other differences of the predicted and actual chains and have the following theorem.

Theorem 8.21. *Given access to an input prediction, there exists an efficient algorithm for minimizing the total weighted completion time of unit-size jobs on a single machine with online chain precedence constraints with a competitive ratio of at most* $O(\min\{1 + \Lambda, \omega\})$*, where* $\Lambda = \Gamma_u + \Gamma_a$*.*

The remaining section is devoted to the proof of this theorem. We first formally define the error measure.

Let $\hat{\mathcal{C}}$ denote the set of predicted chains, and let $\hat{w}_j$ denote the predicted weight of a job j of the instance. All processing requirements are equal to 1, and the algorithm is aware of this. We assume without loss of generality that $|\hat{\mathcal{C}}| = |\mathcal{C}|$ by adding chains with zero (predicted) weight, and that predicted and actual chains have the same identities. That is, there exists exactly one predicted chain $c_i \in \hat{\mathcal{C}}$ for each actual chain $\hat{c}_i \in \mathcal{C}$, which an algorithm can match to each other. Our error measure further requires that there exists for every actual job a predicted counterpart, and vice versa. For a chain c let $|c|$ denote the number of jobs of chain c. We define augmentations of $\mathcal{C}$ and $\hat{\mathcal{C}}$ as follows. Let $\mathcal{C}'$ be composed of all jobs of $\mathcal{C}$, and additionally, for every paired chains $c_i \in \mathcal{C}$ and $\hat{c}_i \in \hat{\mathcal{C}}$:

- if $|\hat{c}_i| > |c_i|$, we add $|\hat{c}_i| - |c_i|$ jobs J_u with weight 0 at the end of c_i in $\mathcal{C}'$. Note that $\mathrm{OPT}(\mathcal{C}) = \mathrm{OPT}(\mathcal{C}')$.
- if $|c_i| > |\hat{c}_i|$, we add $|c_i| - |\hat{c}_i|$ jobs J_a with predicted weight 0 at the end of $\hat{c}_i$ in $\hat{\mathcal{C}}'$.

Note that this construction ensures $\mathrm{OPT}(\hat{\mathcal{C}}) = \mathrm{OPT}(\hat{\mathcal{C}}')$.

For the sake of analysis, assume without loss of generality that both $\mathcal{C}'$ and $\hat{\mathcal{C}}'$ share the same set of jobs J'. Let $n' = |J'|$. We define $\mathrm{OPT}((w'_j)_j)$ as the objective of an optimal solution for J' where a job j has weight w'_j. We further define

$$\mathrm{OPT}((w'_j)_j, (w_j)_j) \coloneqq \max\left\{ \sum_{j \in J'} w'_j C^*_j \;\middle|\; (C^*_j)_j \text{ is an optimal schedule for } (w_j)_j \right\}.$$

Given two fixed augmented instances $\mathcal{C}'$ and $\hat{\mathcal{C}}'$, we define the input prediction error $\Lambda = \Gamma_u + \Gamma_a$:

- a job $j \in J'$ has *unexpected actual* weight if $w_j > \hat{w}_j$. The prediction error due to all unexpected weights can be expressed as $\Gamma_u = \mathrm{OPT}((\max\{\hat{w}_j, w_j\} - w_j)_j, (w_j)_j)$.
- a job $j \in J'$ has *absent predicted* weight if $\hat{w}_j > w_j$. The prediction error due to all absent weights can be expressed as $\Gamma_a = \mathrm{OPT}((\max\{w_j, \hat{w}_j\} - \hat{w}_j)_j, (\hat{w}_j)_j)$.

We are now ready to prove the main theorem of this section.

Proof of Theorem 8.21. Recall that $O(\omega)$-robustness is achieved via Proposition 8.1 and Theorem 7.4. For the other stated bound, we analyze the following algorithm:

1) Efficiently compute an optimal solution based on $\hat{\mathcal{C}}$ [Law78]. This is a non-preemptive schedule for the predicted instance, that is, an order of the jobs.

2) Follow the computed solution. The following situations might occur:

   a) A chain finishes earlier than expected. In this case, discard the remaining predicted jobs of this chain in the precomputed schedule.

   b) A chain continues, although there are no more jobs in this chain in the algorithms schedule. In this case, schedule the remaining jobs in an arbitrary order at the end of the precomputed schedule.

Let ALG denote the objective value of this algorithm. We first show that

$$\mathrm{ALG} \leq \mathrm{OPT}((w_j)_j, (\hat{w}_j)_j).$$

To see this, recall that the algorithm first follows an optimal schedule for jobs $J' \setminus J_u$ and then schedules all unexpected jobs J_u at the end due to case b). Since jobs J_u have predicted weight 0 in $\hat{\mathcal{C}}'$, we can assume that an optimal solution for $\hat{\mathcal{C}}'$ first schedules jobs $J' \setminus J_u$ as our algorithm with the same objective value and makespan as our algorithm, and then schedules jobs J_u in any order. Since $\mathrm{OPT}((w_j)_j, (\hat{w}_j)_j)$ is an upper bound on the actual objective for any such order, the inequality follows. It further holds that (we drop all indices for readability)

$$\begin{aligned}
\mathrm{OPT}(w, \hat{w}) &\leq \mathrm{OPT}(\max\{w, \hat{w}\}, \hat{w}) \\
&= \mathrm{OPT}(\hat{w}) + \mathrm{OPT}(\max\{w, \hat{w}\} - \hat{w}, \hat{w}) \\
&\leq \mathrm{OPT}(\hat{w}, w) + \mathrm{OPT}(\max\{w, \hat{w}\} - \hat{w}, \hat{w}) \\
&\leq \mathrm{OPT}(\max\{\hat{w}, w\}, w) + \mathrm{OPT}(\max\{w, \hat{w}\} - \hat{w}, \hat{w}) \\
&\leq \mathrm{OPT}(w) + \mathrm{OPT}(\max\{\hat{w}, w\} - w, w) + \mathrm{OPT}(\max\{w, \hat{w}\} - \hat{w}, \hat{w}) \\
&= \mathrm{OPT}(w) + \Lambda\,.
\end{aligned}$$

We finally observe that $\mathrm{OPT}((w_j)_j) = \mathrm{OPT}(\mathcal{C})$, as jobs J_a do not influence the objective value of an optimal solution. This completes the proof of the theorem. □

8.8 Concluding Remarks

We initiated the study of learning-augmented algorithms for scheduling with online precedence constraints by considering a hierarchy of prediction models based on their entropy. For several models of the hierarchy, we were able to show that the predicted information is sufficient to break lower bounds for algorithms without predictions. We hope that our approach leads to more discussions on the identification of the "right" prediction model in learning-augmented algorithm design. As a next research step, we suggest investigating the missing bounds for our

prediction models, for example, an upper bound for average predictions, and exploring error measures for full input predictions based on more fine-grained graph distance metrics.

Bibliographic Note

This chapter is based on joint work with Alexandra Lassota, Nicole Megow, and Jens Schlöter [Las+23]. Thus, some parts of this chapter are identical with [Las+23].

Chapter 9

Predictions for Unknown Processing Speeds

9.1 Introduction

In this chapter, we study online scheduling problems on unrelated machines that more properly align with assumptions made in practice. More specifically, we focus on the job-dependent machine speeds s_{ij}. Despite their relevance for high-performance scheduling, there is a big discrepancy between how theory and practice handle them: while scheduling theory most commonly assumes that speeds are known to an algorithm (cf. [AGK12; Gup+21; IKM18; Im+14]), this is typically not the case in practice (cf. [FR98; Khd+15; The19]). Hence, algorithms that perform well in theory are often not applicable in practice.

We propose new models and algorithms to bridge this gap. In particular, we introduce *speed-oblivious* algorithms, which do not rely on knowing (precise) processing speeds. Thereby, we focus on (non-)clairvoyant scheduling subject to minimizing the total weighted completion time.

State-of-the-Art in Theory. To the best of our knowledge, unrelated machine scheduling has been studied only in a *speed-aware* setting, where an algorithm knows the speeds s_{ij} for every available jobs j on every machine i [AGK12; Gup+21; Hal+97; IKM18; Im+14]. It is not difficult to see that there are prohibitive lower bounds for speed-oblivious scheduling on unrelated machines: consider an instance with a single job j that makes substantial progress only on the machine that a speed-oblivious algorithm uses last to process j; this immediately implies a competitive ratio of at least $\Omega(m)$ for m machines.

State-of-the-Art in Practice. Practical scheduling algorithms commonly operate in open systems [FR98], where jobs arrive online, are non-clairvoyant, and, in contrast to the assumption in theory, their exact processing speeds on every core are *unknown* upfront. Therefore, state-of-the-practice schedulers usually ignore heterogeneity between jobs, for example, Linux Energy-Aware Scheduling [The19]. State-of-the-art schedulers rely on prior knowledge about jobs [Khd+15], which is not always available, or rely on predictions of job characteristics to leverage this information gap. Such predictions could be based on prior executions of repeating jobs or on machine-learning-based techniques [Gup+18; Rap+21]. They are often quite precise, but can be highly inaccurate due to varying and unpredictable input data. Figure 9.1 shows job-dependent speed varieties in common benchmark suites (*PARSEC-3.0*, *SPLASH-3*, *Polybench*) running on *big* and *LITTLE* cores of a Kirin 970 smartphone SoC with Arm big.LITTLE architecture. To the best of our knowledge, all these approaches are evaluated only empirically. In particular,

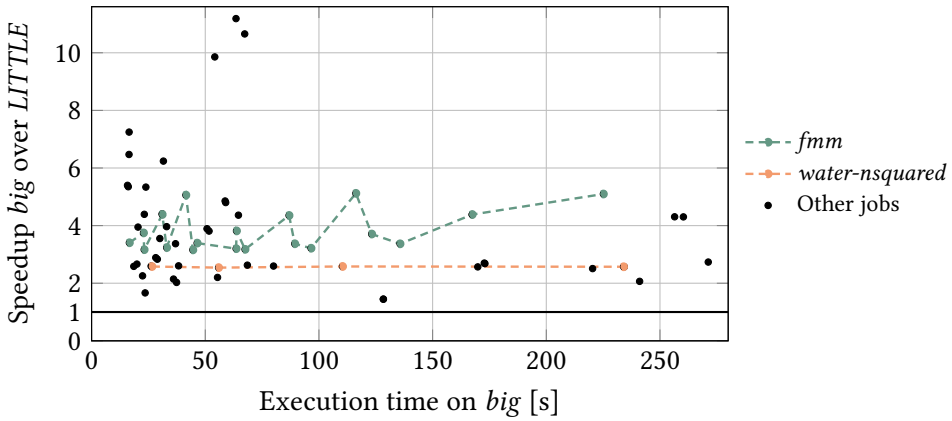


Figure 9.1: The execution time and speedup of the *big* over *LITTLE* cores on an Arm big. Each point represents the execution of a job with a distinct input. LITTLE heterogeneous processor varies strongly between jobs and different input data. Variations of the speedup with respect to the input data are large for some jobs (for example, *water-nsquared*) but small for others (for example, *fmm*).

there are no theoretical guarantees on the performance in worst-case scenarios or with respect to a prediction's quality.

9.1.1 Our Results

We initiate the theoretical study of speed-oblivious algorithms. Since strong lower bounds rule out good worst-case guarantees for unrelated machines without further assumptions, we propose two (new) models, which are motivated by data-driven machine-learned models and modern heterogeneous hardware architectures:

- **Speed predictions** give algorithms access to values $\hat{s}_{ij}$ for every machine i at the release date of every job j. We measure the accuracy of such a prediction by the *distortion error* μ, where $\mu = \mu_1 \cdot \mu_2$ with

$$\mu_1 = \max_{i \in I, j \in J} \frac{\hat{s}_{ij}}{s_{ij}} \text{ and } \mu_2 = \max_{i \in I, j \in J} \frac{s_{ij}}{\hat{s}_{ij}} .$$

- **Speed-ordered machines** assume an order $\leq$ on the machines such that for all machines i, i and jobs $j \in J$ it holds $s_{ij} \geq s_{i'j}$ if and only if $i \leq i'$. Algorithms are aware of this order.

For both models, we present competitive algorithms, which also can handle both, online job arrival and non-clairvoyance. We start with our results on speed predictions in Section 9.2 and then present our results on speed-ordered machines in Section 9.3.

9.1.2 Further Related Work

Uncertainty about machine speeds or, generally, the machine environment, have hardly been studied in scheduling theory. Some works consider scheduling with unknown non-availability

periods, that is, periods with speed 0 [AS01; Die+09], permanent break-downs of a subset of machines [SZ20], or more generally arbitrarily changing machine speed for a single machine [MM16], but not on heterogeneous machines. In scheduling with testing, unknown processing requirements of a job (and thus its machine-dependent speed) can be explored by making queries, for example, [AE20; Ara+18; Dür+20], but also here heterogeneous processors are not considered.

In the area of learning-augmented algorithms, Balkanski et al. [Bal+23b] consider a robust scheduling problem where machine speeds are only predicted and jobs have to be grouped to be scheduled together before knowing the true machine speeds; such problems without predictions were introduced in [Ebe+21; SZ20]. In contrast, in our model an algorithm will never learn about a job's true speed(s) before its completion and, further, the speeds might be job-dependent.

9.2 Algorithms for Speed Predictions

We start with a lower bound for speed-oblivious algorithms with speed predictions, which is linear in the distortion error.

Theorem 9.1. *Any speed-oblivious algorithm with speed predictions has a competitive ratio of at least* $\Omega(\min\{\mu, m\})$ *for* $Q \mid pmtn \mid \sum C_j$.

Proof. Let $\mu_1, \mu_2 \geq 1$ and $\mu = \mu_1 \cdot \mu_2$. Consider an instance $J = \{j\}$ with $p_j = 2\mu$ and $m \geq 2\mu$ machines such that $\hat{s}_i = \mu_1$ for all $1 \leq i \leq m$. The algorithm cannot distinguish the machines. For the first $2\mu - 1$ machines i on which the algorithm processes j, the adversary fixes $s_i = 1$. Thus, at time $2\mu - 1$, the remaining processing requirement of j is at least $2\mu - (2\mu - 1) = 1$, and there exists a machine i' on which j has not been processed yet. Thus, the adversary can set $s_{i'} = \mu$ and complete j on i' within two time units, implying a competitive ratio of at least $\Omega(\min\{\mu, m\})$. □

Observe that this construction already works for two machines when migration is forbidden.

For the upper bound, we can achieve a tight competitive ratio of $\Theta(\mu)$ when executing Proportional Fairness (cf. Chapter 5) with speed predictions. That is, at any time instant, we compute the rates for available jobs based on the vector of predicted speeds $\hat{s}$. Then, it is straightforward to adopt the analysis given in Section 5.3 to obtain a proof for the following theorem.

Theorem 9.2. *There exists a non-clairvoyant speed-oblivious algorithm with speed predictions with a competitive ratio in* $O(\mu)$ *for* $R \mid r_j, pmtn \mid \sum w_j C_j$.

9.3 Algorithms for Speed-Ordered Machines

This section contains our results on speed-ordered machines. In the first subsection, we present a clairvoyant algorithm, and in the second subsection a non-clairvoyant algorithm. But first, we observe that in this model migration is necessary for speed-oblivious algorithms.

Algorithm 6: PreemptiveWSPT on speed-ordered related machines

Input: speed-ordered machines $s_1 \geq \ldots \geq s_m$, time t.

1 $\sigma_t \leftarrow$ order of $J(t)$ with non-increasing w_j/p_j ;

2 $A_t = \{(k, \sigma_t(k))\}_{k \in [\ell]}$ where $\ell = \min\{m, |J(t)|\}$;

3 Schedule jobs to machines according to A_t at time t ;

Algorithm 7: Equal sharing on speed-ordered unrelated machines

Input: speed-ordered machines $1, \ldots, m$, time t.

1 Allocate every job $j \in J(t)$ to every machine $i \in [\min\{|J(t)|, m\}]$ by $x_{ijt} = \frac{1}{|J(t)|}$.

Theorem 9.3. *Any non-migratory speed-oblivious algorithm has a competitive ratio of at least $\Omega(m)$ for minimizing the total completion time on m speed-ordered machines, even if it is clairvoyant and the machines are related.*

Proof. Consider the execution of some algorithm on an instance of n jobs with unit-weights and with processing requirements equal to n^2m and $s_1 = n^2m$. If at some point in time, the algorithm starts a job on machines $2, \ldots, m$, the adversary sets $s_2 = \ldots = s_m = 1$ to enforce an objective value of at least $\Omega(n^2m)$, while scheduling all jobs on the first machine gives an objective value of at most $O(n^2)$. If this does not happen, the algorithm must have scheduled all jobs on the first machine. But then the adversary sets $s_2 = \ldots = s_m = n^2m$ and achieves an objective value of $O(\frac{n^2}{m})$ by distributing the jobs evenly to all machines, while the algorithm has an objective value of $\Omega(n^2)$. □

9.3.1 Clairvoyant Algorithm

We first present a clairvoyant algorithm for speed-ordered related machines with a constant competitive ratio. In fact, we show that we can emulate the PreemptiveWSPT algorithm from Section 4.4 in this setting. To see this, note that if the machines are related and speed-ordered, PreemptiveWSPT assigns jobs by non-increasing order of w_j/p_j to machines in speed order, because this clearly maximizes the total scheduled density, that is, sum of assigned $w_j s_i/p_j$. Therefore, we can compute this assignment at any time t without computing a maximum weighted matching, and thus, do not require precise speed values. A simplified description of PreemptiveWSPT in the speed-ordered setting is given in Algorithm 6.

Therefore, the following theorem is a corollary of Theorem 4.5.

Theorem 9.4. *Algorithm 6 has a competitive ratio of at most 7.24 for minimizing the total weighted completion time on speed-ordered related machines with release dates, preemption, and migration.*

9.3.2 Non-Clairvoyant Algorithm

In this section, we present a non-clairvoyant algorithm for speed-ordered machines. Intuitively, this algorithm (Algorithm 7) behaves like PF for unit weight jobs on related machines. This can be seen via the combinatorial implementation of PF in Section 5.6.

Thus, the following theorem is a direct implication of Theorem 5.30.

Theorem 9.5. *Algorithm 7 has a competitive ratio of at most* 3 *for minimizing the total completion time on speed-ordered related machines with non-uniform release dates. For uniform release dates, the competitive ratio is equal to* 2.

For speed-ordered unrelated machines, Algorithm 7 is not necessarily equal to PF. However, we can still analyze Algorithm 7 for this more general setting and prove a non-trivial bound on its competitive ratio.

Theorem 9.6. *Algorithm 7 has a competitive ratio of at most* $O(\log(\min\{n, m\}))$ *for minimizing the total completion time on speed-ordered unrelated machines.*

For every time t, we write $m(t) \coloneqq \min\{m, |J(t)|\}$. The definition of Algorithm 7 implies that, at every time t, a job $j \in J(t)$ receives a processing rate equal to $y_j(t) = \sum_{i=1}^{m(t)} s_{ij}/|J(t)|$.

We prove Theorem 9.6 via dual-fitting. To this end, we again state the dual of the speed scaled variant of (LP_R) below.

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} a_j - \sum_{i=1}^{m} \sum_{t \geq 0} b_{it} - \sum_{j=1}^{n} \sum_{t \geq r_j} c_{jt} && && (\mathrm{DLP}_R(\kappa)) \\ \text{s.t.} \quad & \frac{a_j s_{ij}}{p_j} - \frac{s_{ij}}{p_j} t \leq \kappa \cdot b_{it} + \kappa \cdot c_{jt} && \forall i \in [m], \forall j \in [n], \forall t \geq r_j \\ & a_j, b_{it}, c_{jt} \geq 0 && \forall i \in [m], \forall j \in [n], \forall t \geq 0 \end{aligned}$$

Fix an instance and the algorithm's schedule. We define

- $\beta_{it} \coloneqq \frac{1}{i} \cdot |J(t)| \cdot \mathbb{1}\left[i \leq |J(t)|\right]$ for every machine i and every time t, and
- $\gamma_{jt} \coloneqq \mathbb{1}\left[j \in J(t)\right]$ for every job j and every time t.

We set $\kappa \coloneqq 2 \cdot \log(\min\{n, m\})$. Note that κ is an upper bound of $\sum_{i=1}^{m(t)} \frac{1}{i}$, which we use in the following observations.

Lemma 9.7. *At any time* t, $\sum_{i=1}^{m} \beta_{it} \leq \kappa \cdot |U(t)|$.

Proof. At any time t,

$$\sum_{i=1}^{m} \beta_{it} = \sum_{i=1}^{m(t)} \frac{1}{i} \cdot |J(t)| \leq |U(t)| \sum_{i=1}^{m(t)} \frac{1}{i} \leq \kappa \cdot |U(t)|\,,$$

where in the last inequality we use that $m(t) = \min\{m, |J(t)|\} \leq \min\{m, n\}$. □

Similarly, we can show the following bound.

Proposition 9.8. *At any time* t, $\sum_{j \in J : r_j \geq t} \gamma_{jt} \leq |U(t)|$.

For every time t, let $\zeta(t)$ be the median of the values $y_j(t)/p_j$, $j \in U(t)$. More formally, if Z_t denotes the sorted (ascending) list of length $|U(t)|$ composed of $y_j(t)/p_j$ for every $j \in U(t)$, then $\zeta(t)$ is the value at the index $\lceil \frac{1}{2}|U(t)| \rceil$ in Z_t.

We define the following dual assignment:

- $\bar{a}_j \coloneqq \sum_{t'=0}^{C_j} \bar{a}_{jt'}$ for every job j, where $\bar{a}_{jt'} \coloneqq \mathbb{1}\left[\frac{y_j(t')}{p_j} \leq \zeta(t')\right]$,
- $\bar{b}_{it} \coloneqq \frac{1}{16\kappa} \sum_{t' \geq t} \beta_{it'} \zeta(t')$ for every machine i and every time t, and
- $\bar{c}_{jt} \coloneqq \frac{1}{16} \sum_{t' \geq t} \gamma_{jt'} \zeta(t')$ for every job j and every time $t \geq r_j$.

The following two lemmas show that this assignment is feasible and that its dual objective value captures a fraction of the algorithm's objective value.

Lemma 9.9. *It holds that* $\frac{1}{4}\mathrm{ALG} \leq \sum_{j=1}^{n} \bar{a}_j - \sum_{i=1}^{m} \sum_{t \geq 0} \bar{b}_{it} - \sum_{j=1}^{n} \sum_{t \geq r_j} \bar{c}_{jt}$.

Proof. We first prove that $\sum_{j=1}^{n} \bar{a}_j \geq \frac{1}{2}\mathrm{ALG}$. Consider a time t. Observe that $\sum_{j \in U(t)} \bar{a}_{jt}$ contains the total number of jobs j that satisfy $y_j(t)/p_j \leq \zeta(t)$. By the definition of $\zeta(t)$, we conclude that this is at least $\frac{1}{2} \cdot |U(t)|$, that is, $\sum_{j \in U(t)} \bar{a}_{jt} \geq \frac{1}{2} \cdot |U(t)|$. The bound then follows by summation over time.

Second, we show that $\sum_{t \geq 0} \sum_{i=1}^{m} \bar{b}_{it} \leq \frac{1}{8} \cdot \mathrm{ALG}$. Fix a time t. Observe that, for every $t' \geq t$, the definition of $\zeta(t')$ implies that $\sum_{j \in U(t')} \mathbb{1}\left[y_j(t')/p_j \geq \zeta(t')\right] \geq \frac{1}{2}|U(t')|$. Thus,

$$\zeta(t') \cdot \frac{1}{2}|U(t')| \leq \sum_{j \in U(t')} \zeta(t') \cdot \mathbb{1}\left[\frac{y_j(t')}{p_j} \geq \zeta(t')\right] \leq \sum_{j \in U(t')} \frac{y_j(t')}{p_j} \,. \tag{9.1}$$

The definition of $\bar{b}_{it}$ and Lemma 9.7 imply

$$\sum_{i=1}^{m} \bar{b}_{it} = \sum_{i=1}^{m} \frac{1}{16\kappa} \sum_{t' \geq t} \beta_{it'} \cdot \zeta(t') = \frac{1}{16\kappa} \sum_{t' \geq t} \zeta(t') \sum_{i=1}^{m} \beta_{it'} \leq \frac{1}{16} \sum_{t' \geq t} \zeta(t') \cdot |U(t')| \,.$$

Using (9.1), we conclude that this is at most

$$\frac{1}{8} \sum_{t' \geq t} \sum_{j \in U(t')} \frac{y_j(t')}{p_j} \leq \frac{1}{8} \sum_{j \in U(t)} \sum_{t' \geq t} \frac{y_j(t')}{p_j} \leq \frac{1}{8}|U(t)| \,.$$

The second inequality follows from $U(t') \subseteq U(t)$ for $t' \geq t$. The third inequality holds because $\sum_{t' \geq t} y_j(t') \leq p_j$ for every job j. Summing over time implies the stated bound.

Analogously, we can show via Proposition 9.8 that $\sum_{j=1}^{n} \sum_{t \geq r_j} \bar{c}_{jt} \leq \frac{1}{8}\mathrm{ALG}$. This concludes the proof of the lemma. □

Lemma 9.10. *The dual solution* $(\bar{a}_j)_j$, $(\bar{b}_{it})_{i,t}$, *and* $(\bar{c}_{it})_{j,t}$ *is feasible for* $(\mathrm{DLP}_R(\kappa))$.

Proof. First observe that the assignment is non-negative. Let $i \in M$, $j \in J$ and $t \geq r_j$. We have

$$\begin{aligned} \frac{\bar{a}_j s_{ij}}{p_j} - \frac{s_{ij} \cdot t}{p_j} &\leq \sum_{t'=t}^{C_j} \frac{s_{ij}}{p_j} \cdot \mathbb{1}\left[\frac{y_j(t)}{p_j} \leq \zeta(t')\right] \\ &= \sum_{t'=t}^{C_j} \frac{s_{ij}}{y_j(t)} \cdot \frac{y_j(t)}{p_j} \cdot \mathbb{1}\left[\frac{y_j(t)}{p_j} \leq \zeta(t')\right] \leq \sum_{t'=t}^{C_j} \frac{s_{ij}}{\sum_{\ell=1}^{m(t')} s_{\ell j}/|J(t')|} \cdot \zeta(t') \,. \end{aligned} \tag{9.2}$$

Consider any time t' with $t \le t' \le C_j$. If $i \le |J(t')|$, by the speed order, $\sum_{\ell=1}^{m(t')} s_{\ell j} \ge \sum_{\ell=1}^{i} s_{\ell j} \ge i \cdot s_{ij}$, and thus,

$$\frac{s_{ij}}{\sum_{\ell=1}^{m(t')} s_{\ell j}} \cdot |J(t')| \cdot \zeta(t') \le \frac{1}{i} \cdot |J(t')| \cdot \zeta(t') = \beta_{it'} \cdot \zeta(t') \,.$$

Otherwise, that is, $i > |J(t')|$, it holds that $s_{ij} \le s_{\ell j}$ for all $1 \le \ell \le |J(t')|$, and thus, $\sum_{\ell=1}^{m(t')} s_{\ell j} \ge \sum_{\ell=1}^{|J(t')|} s_{\ell j} \ge |J(t')| \cdot s_{ij}$. Therefore,

$$\frac{s_{ij}}{\sum_{\ell=1}^{m(t')} s_{\ell j}} \cdot |J(t')| \cdot \zeta(t') \le \frac{|J(t')|}{|J(t')|} \cdot \zeta(t') = \gamma_{jt'} \cdot \zeta(t') \,,$$

because $t' \le C_j$. Put together, (9.2) is at most

$$\sum_{t'=t}^{C_j} \beta_{it'} \zeta(t') + \sum_{t'=t}^{C_j} \gamma_{jt'} \zeta(t') \le \kappa(\bar{b}_{it} + \bar{c}_{jt}) \,,$$

which verifies the dual constraint. □

We now prove Theorem 9.6. Lemmas 9.9 and 9.10 and weak LP duality imply

$$\kappa \cdot \mathrm{OPT} \ge \sum_{j \in J} \bar{a}_j - \sum_{i=1}^{m} \sum_{t \ge 0} \bar{b}_{it} - \sum_{j=1}^{n} \sum_{t \ge r_j} \bar{c}_{jt} \ge \frac{1}{4} \cdot \mathrm{ALG} \,,$$

which concludes that Algorithm 7 has a competitive ratio of at most $O(\kappa)$.

Finally, we show that the analysis is asymptotically tight.

Lemma 9.11. *Algorithm 7 has a competitive ratio of at least $\Omega(\log(\min\{n, m\}))$ for minimizing the total completion time on speed-ordered unrelated machines, even if all $s_{ij} \in \{0, 1\}$.*

Proof. Consider an instance of m unit-sized jobs $[m]$ and m machines $[m]$. Every job $j \in [m]$ has on machine $i \in [m]$ a processing speed equal to $s_{ij} = \mathbb{1}[i \le m - j + 1]$. First, observe that $\mathrm{OPT} \le m$, because we can process and complete every job $j \in [m]$ exclusively on machine $m - j + 1$ at time 1. We now calculate the algorithm's objective value. To this end, we argue that in the algorithm's schedule holds $C_j = 1 + \sum_{i=1}^{j-1} \frac{1}{m-i+1}$ for every job j. Then, $\mathrm{ALG} = \sum_{j=1}^{m} C_j = \Omega(m \log m)$ concludes the statement.

We first observe that $C_1 = 1$, because job 1 receives in interval $I_1 = [0, C_1)$ on every machine a rate equal to $\frac{1}{m}$. We now argue iteratively for $j = 2, \ldots, m$ that $C_j = 1 + \sum_{i=1}^{j-1} \frac{1}{m-i+1}$. Consequently, in interval $I_j = [C_{j-1}, C_j)$ must be exactly jobs $j, \ldots, m$ alive. Fix a job j with $2 \le j \le m$ and let $2 \le i \le j$. Since j receives progress on exactly $m - j + 1$ machines, there are $m - i + 1$ alive jobs in I_i, and I_i has length $\frac{1}{m-i+2}$, its total progress in I_i is equal to $\frac{m-j+1}{(m-i+1)(m-i+2)}$. Further, j's progress is equal to $\frac{m-j+1}{m}$ in I_1. Summing over all intervals I_i with $1 \le i \le j$ concludes that j's progress until the end of I_j is equal to

$$\frac{m - j + 1}{m} + \sum_{i=2}^{j} \frac{m - j + 1}{(m - i + 1)(m - i + 2)} = 1 \,,$$

asserting that $1 + \sum_{i=1}^{j-1} \frac{1}{m-i+1}$ is indeed j's completion time in the algorithm's schedule. □

9.4 Concluding Remarks

We initiated the study of speed-oblivious algorithm, and presented two models that mitigate the pessimistic $\Omega(m)$ lower bound using certain additional information. As a future direction, it would be interesting whether a constant competitive algorithm for speed-ordered unrelated machines is possible for the clairvoyant or even non-clairvoyant setting. Moreover, one could consider the more general total weighted flow time objective for speed-ordered machines, with or without speed augmentation.

Bibliographic Note

This chapter is based on joint work with Nicole Megow and Martin Rapp [LMR23]. Specifically, a detailed proof of Theorem 9.2 can be found there. Some parts of this chapter are identical with [LMR23].

Chapter 10

Predictions for Uncertain Jobs in Online TSP

10.1 Introduction

Imagine that you are a self-employed chimney sweep in a large city and perform annual inspections of heating systems. Every day customers call you with service requests. Since you want good reviews and a happy customer base, your goal is to serve all requests on the same day. You also want to be as early as possible at home in the evening. To achieve these goals, you often have to decide which customer to serve next and which route to take. This clearly affects your evening plans: you may drive from one end of the city to the other end several times because you cannot anticipate upcoming service requests. One day you realize that many of your regular customers call you on the exact same date every year because their validations run out. How can you integrate such estimates in planning your tours?

Now, imagine that you are an independent city courier and have to transport packages from one place to another. Your customers call you during the day with transportation requests, and again, you want to fulfill all requests by the end of the day so you can get home as early as possible. Your main decision is the order in which you serve these requests. After building a customer base, you find that some customers call you in the morning, and others rather in the evening. How can you integrate such estimates in planning your tours?

In this chapter, we introduce and analyze strategies that will help both the chimney sweep and the city courier to optimize their tours and reduce their overtime given that their estimations roughly match the reality. To this end, we first study the *online Traveling Salesperson problem* (OlTSP), which is an abstraction of the chimney sweep's daily situation. Later, we consider the *online Dial-a-Ride problem* (OlDARP), which corresponds to the day in a life of our city courier, and we will see that the courier can adopt the strategies developed by the chimney sweep. We start by formally defining both problems.

Online TSP and Online Dial-a-Ride. In the online Traveling Salesperson Problem (OlTSP), we are given a metric space $M = (X, d)$ with an origin $o \in X$, and a set of requests R that are released online over time. A request $(x, r) \in R$ is composed of a point $x \in X$ and a release date $r \in \mathbb{R}_{\geq 0}$, at which time it becomes known and can be served. The task of an algorithm is to route a server, which is initially located at the origin and moves at unit speed, to serve all requests. A request $(x, r) \in R$ is served if the server is located at x at any time $t \geq r$. Moreover, after serving all requests, the server has to be moved back to the origin. The objective is to minimize the makespan, that is, the total time required to serve the instance and return the server to the origin. For general metric spaces, there are deterministic 2-competitive algorithms for

OLTSP [AKR00; Aus+01], which is best-possible [Aus+01]. If $X = \mathbb{R}_{\geq 0}$, there is a best-possible $\frac{3}{2}$-competitive algorithm [Blo+01].

The online Dial-a-Ride problem (OLDARP) is a generalization of OLTSP where each request (x^s, x^d, r) consists of a starting location x^s, a destination x^d and a release time r. To serve a request (x^s, x^d, r), the server must first visit x^s at some time not earlier than r, and then x^d. We assume here that the server can carry at most one request at a time and a move from x^s to x^d cannot be interrupted, that is, there is no storage possible. There exist best-possible 2-competitive algorithms for OLDARP [AKR00; FS01].

Online Routing with Predictions. In OLTSP with predictions, we are additionally given an input-prediction $\hat{R}$ on the set of requests R. That is, $\hat{R}$ is composed of predicted requests $(\hat{x}, \hat{r})$ with a predicted location $\hat{x}$ and a predicted release date $\hat{r}$. Similarly, in OLDARP with predictions, we are given additionally a set $\hat{R}$ of predicted requests $(\hat{x}^s, \hat{x}^d, \hat{r})$. We emphasize that $\hat{R}$ can be substantially larger or smaller than R. Additionally, in the learning-augmented setting, algorithms receive a signal indicating the no more requests will be revealed if the server is at the origin and there are no unserved requests.

Before presenting our contribution, we emphasize that in the design of online algorithms for OLTSP, we usually assume that computing a shortest tour through a set of points can be done optimally. This task is APX-hard [KLS15], but there exists a classic $\frac{3}{2}$-approximation algorithm [Chr76; Chr22; Ser78] as well as recent slightly-better-than-$\frac{3}{2}$ approximations [KKG21; KKG23]. All relevant online algorithms, such as the already mentioned ones, can also be adopted to use approximation algorithms for this task. These variants have slightly worse competitive ratios, which depend on the approximation ratio of the used subroutine.

10.1.1 Our Results

We develop and analyze learning-augmented algorithms for OLTSP and OLDARP that are consistent, smooth, and robust, and prove impossibility results on the consistency-robustness tradeoff. Furthermore, we design a novel error framework that is applicable to OLTSP but also to other graph and metric problems. We now give a detailed overview of our results and the structure of this chapter.

In Section 10.2, we prove a lower bound on the optimal consistency-robustness tradeoff for OLTSP. We show that for any $\alpha \in (0, \frac{1}{2})$, any deterministic $(1+\alpha)$-consistent algorithm has a robustness ratio of at least $\frac{1-\alpha}{\alpha}$, which already holds when $X = \mathbb{R}_{\geq 0}$. We present for this special metric space, the half-line, an $(1+\alpha)$-consistent, smooth and $\frac{3}{2\alpha}$-robust algorithm (see Section 10.3). Here, $\alpha \in (0, \frac{1}{2}]$ is a tunable hyperparameter with which we can control the level of trust in the given prediction. To achieve this result, we integrate the best-possible online algorithm Move-Right-If-Necessary (MRIN) by Blom et al. [Blo+01] into a three-stage robustness framework.

Then, in Section 10.4, we extend this framework to general metrics and integrate the best-possible SMARTSTART algorithm by Ascheuer et al. [AKR00]. By further adopting the characteristics of SMARTSTART, we give an $(1+\alpha)$-consistent, smooth, and $(2+2/\alpha)$-robust algorithm, for any $\alpha \in (0, 1]$. While these bounds hold for an implementation that does not run in polynomial

time unless P = NP, we present in Section 10.5 a polynomial-time variant with slightly worse guarantees. To complete the results on online routing problems, we show that our approach can be generalized to the more general OLDARP (see Section 10.6).

To achieve these results, we introduce a novel error framework, which we call the *cover error* Λ (see Section 10.1.2). On a high level, it constructs, for any input R and input-prediction $\hat{R}$, a bipartite hypergraph and computes two hyperedge covers, one for each side, which minimum total costs yield the error value Λ. We highlight two useful properties of this framework, and defer more discussions of our error to later sections. First, the only requirement to apply the cover error to a problem is to specify the cost of a hyperedge. Thus, it is very general and can easily be applied to different types of online problems, such as online-list and online-time problems. Second, by further restricting the hyperedge used in a minimum-cost cover by some parameter k, we obtain a family of errors $\{\Lambda_k\}_{k=1}^{\infty}$, where large values of k correspond to a more precise and sensitive error measure. Specifically, we show that algorithms can be smooth for all errors of this family at the same time.

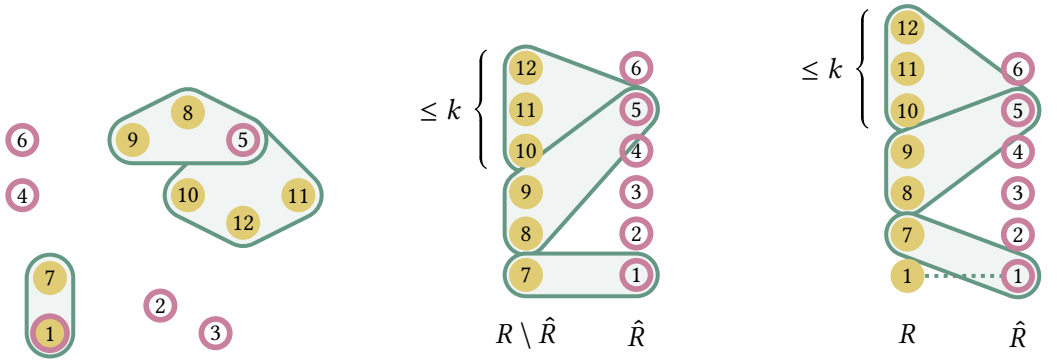
We demonstrate these and more advantages of the cover error by applying it to two classic online-list network design problems: *online Steiner tree* and *online Steiner forest*. For both problems, previous learning-augmented algorithms with specific error measures are known [APT22a; XM22]. In Section 10.7, we analyze the algorithm by Azar et al. [APT22a] with respect to the cover error, and thereby overcome shortcomings in which the previous error measures greatly underestimate the quality of a prediction.

10.1.2 The Cover Error

Next, we motivate and introduce the *cover error*. It is an error framework and can be applied to any problem where a certain (part of the) input set R is predicted by some set $\hat{R}$. Throughout this section, we use OLTSP as sample application, and R as the set of requests and $\hat{R}$ as the set of predicted requests.

The framework distinguishes between two types of errors: *unexpected actual requests* $R \setminus \hat{R}$ and *absent predicted requests* $\hat{R} \setminus R$. Both potentially endanger an algorithm that trust $\hat{R}$ to a certain degree, that is, when an unexpected request appears far up from the predicted route, or when routing the server unnecessarily to a single isolated absent prediction. We *cover* potential extra costs that an algorithm incurs by computing a certain minimum-cost hyperedge cover for each type. To make this idea more precise, think of the situation where an algorithm follows a predicted tour for $\hat{R}$. After serving some predicted request $(\hat{x}, \hat{r})$, it may also serve some unexpected actual requests $R' \subseteq R \setminus \hat{R}$ that have already been released and are *relatively close* to $(\hat{x}, \hat{r})$ (both time- and location-wise). In our terminology, think of $(\hat{x}, \hat{r})$ covering R', and observe that the cost for this cover shall naturally be equal to the optimal cost for serving R' when starting in $\hat{x}$ at time $\hat{r}$. Conversely, the absent predicted requests, $\hat{R} \setminus R$, that are nevertheless visited by an algorithm that follows a predicted tour should be covered by *close* actual requests, to make up for the extra cost incurred by these superfluous visits.

We now embed this intuition into a precise definition. Let A and B be two sets of possibly different size and let $k \geq 1$. We define a bipartite hypergraph $G_k = (A \cup B, \mathcal{H})$ where $\mathcal{H}$ is the set of all hyperedges that have exactly one endpoint in B and at most k endpoints in A. A *k-hyperedge cover of A by B* is a set of hyperedges $\mathcal{H}' \subseteq \mathcal{H}$ in G_k such that every vertex in A is



(a) Instance in metric space (b) Min-cost k-hyperedge cover (c) $\Gamma_k(R, \hat{R}) = \Gamma_k(R \setminus \hat{R}, \hat{R})$

Figure 10.1: Example for a metric instance and input prediction with a min-cost k-hyperedge cover of the set of unexpected requests $R \setminus \hat{R}$. The actual requests are filled green and the predicted requests are encircled red. The labels show which points in the metric space correspond to which nodes in the bipartite graphs.

incident to at least one hyperedge in $\mathcal{H}'$. If every hyperedge $h \in \mathcal{H}$ of G_k has an associated cost $\gamma(h)$, a *minimum-cost k-hyperedge cover* $\mathcal{H}'$ is a k-hyperedge cover that minimizes $\sum_{h' \in \mathcal{H}'} \gamma(h')$. We denote the cost of a min-cost k-hyperedge cover of A by B by $\Gamma_k(A, B)$. The cover error, denoted by $\Lambda_k(R, \hat{R})$, is defined as

$$\Lambda_k(R, \hat{R}) := \Gamma_\infty(\hat{R}, R) + \Gamma_k(R, \hat{R}) \ .$$

Notice that we allow arbitrary large hyperedges ($k = \infty$) to cover predicted requests $\hat{R}$ in the above definition. We emphasize that all results also hold for a symmetric error definition $\Gamma_k(\hat{R}, R) + \Gamma_k(R, \hat{R})$, because $\Gamma_i(A, B) \geq \Gamma_{i+1}(A, B)$, for any i. Nevertheless we use this asymmetric definition to obtain a stronger bound when covering $\hat{R}$. Intuitively, this is possible because all predicted requests are known in advance (as opposed to the actual requests, which arrive online). We simply write Λ_k if $\hat{R}$ and R are clear from the context. Moreover, observe that the framework yields a hierarchical family $\{\Lambda_k\}_{k=1}^{\infty}$ of errors, where errors with a larger k allow more flexible covers, and thus reflect more precisely the cost due to trusting wrong predictions.

To apply the cover error framework to a specific problem, it suffices to specify the cost $\gamma(A', b)$ of a hyperedge $A' \cup \{b\}$. Since our error measure shall give value zero if $\hat{R} = R$, we require that the cost of every hyperedge $\{a\} \cup \{b\}$ for some $a \in A$ and $b \in B$ is equal to zero if $a = b$. Then, all vertices in $A \cap B$ can be covered trivially by B, and we conclude that $\Gamma_k(A \setminus B, B) = \Gamma_k(A, B)$. Although we give precise definitions of γ separately for every concrete problem, all definitions follow a certain paradigm. That is, the cost $\gamma(A', b)$ shall be equal to *the value of an optimal solution for the subinstance induced by A' with respect to b*. This anchoring requirement is the *single* detail that has to be specified for a concrete problem. Note that this matches our intuition discussed above for OLTSP. Figure 10.1 depicts an example of a k-hyperedge cover.

10.1.3 Preliminaries and Classical Algorithms for OlTSP

For a fixed instance, we denote the makespan of an optimal solution by OPT, and for a fixed algorithm, we denote the server's location at time t by $p(t)$.

We now present known algorithmic and impossibility results for OlTSP, which are relevant in later sections. We start with results on the half-line metric, where the following lower bound on the competitive ratio of deterministic algorithms holds.

Theorem 10.1 (Blom et al. [Blo+01]). *The competitive ratio of any deterministic online algorithm for OlTSP is at least $\frac{3}{2}$, even on the half-line metric.*

Proof. Fix any deterministic online algorithm and consider an instance with the initial request $(1, 0)$. Let $T \geq 1$ be the time when the algorithm served it and returned to the origin. If $T \geq 3$, an optimal solution could have completed the instance at time 2, asserting the statement. Otherwise, that is $T < 3$, there is a second request (T, T), for which the algorithm requires at least $2T$ additional time. Since an optimal solution can serve both requests in time $2T$, we also conclude in this case that the algorithm's competitive ratio is at least $\frac{3}{2}$. □

We can precisely characterize an optimal solution for the half-line metric. This is because no request (x, r) can be served before time $\max\{x, r\}$; and once it has been served, at least x more time units are required for returning to the origin. Hence, for a given set of requests R, any algorithm has a makespan of at least $\max_{(x,r)\in R}\{r+x, 2x\}$. Further, an optimal offline algorithm can find a tour of this makespan by initially moving to the point $\frac{1}{2} \cdot \max_{(x,r)\in R}\{r + x, 2x\}$ and then back to the origin. It is not hard to verify that this tour serves all requests of R.

Proposition 10.2. *On the half-line, it holds that* $\mathrm{OPT} = \max_{(x,r)\in R}\{2x, x + r\}$.

Blom et al. also presented an algorithm that matches their lower bound (Theorem 10.1). Since we will use this algorithm in Section 10.3, we introduce it now, and present the analysis of Blom et al. In summary, their algorithm simply moves towards the right, that is, the positive part of the real line, of its current position if there is some unserved request, and otherwise returns to the origin. The formal description is given in Algorithm 8.

Algorithm 8: Move-Right-If-Necessary (Mrin) [Blo+01]

1 At any time t:
2 **if** *there is an unserved requests to the right of* $p(t)$ **then** move towards the right,
3 **else** move towards the origin.

Theorem 10.3 (Blom et al. [Blo+01]). *Move-Right-If-Necessary (Mrin) has a competitive ratio of at most $\frac{3}{2}$ for the OlTSP problem on the half-line.*

Proof. We prove the theorem by induction on the number of requests of the instance. If there is only a single request, the algorithm is clearly optimal. Now let (x, r) be the last request, and f be the largest position of an unserved request at time t, excluding (x, r) (otherwise $f = 0$).

If $x \leq f$ or $p(r) \geq x$, the algorithm's makespan for serving this instance is equal to the makespan of serving all requests excluding (x, r). Since this additional request also cannot lower the optimal makespan, we are done by the induction hypothesis.

Otherwise, that is, $x > f$ and $p(r) < x$, the algorithm's makespan is at most $r + 2x$. By Proposition 10.2, we conclude that the competitive ratio is at most $\frac{2x+r}{\mathrm{OPT}} \leq \frac{x+r}{x+r} + \frac{x}{2x} = \frac{3}{2}$. □

Ausiello et al. [Aus+01] presented an improved lower bound of 2 on the competitive ratio of deterministic algorithms. Their proof uses the boundary of the unit square as metric space.

Theorem 10.4 (Ausiello et al. [Aus+01]). *There exists a metric space for which the competitive ratio of any deterministic online algorithm for* OlTSP *is at least* 2.

There are algorithms for general metric spaces that match this lower bound. In the following, we introduce the SmartStart algorithm by Ascheuer et al. [AKR00], which is 2-competitive and which we will use in Section 10.4. We note that Ausiello et al. [Aus+01] also presented a different 2-competitive online algorithm.

Algorithm 9: SmartStart [AKR00]

1 Whenever the server is at the origin at some time t, compute an optimal tour S of length $\ell(S)$ serving all released but unserved requests at time t.
2 **if** $\ell(S) \leq t$ **then** follow S
3 **else** restart the algorithm at time $\ell(S)$.

The SmartStart algorithm (cf. Algorithm 9) is a tuned variant of the so-called Ignore strategy: whenever the server is at the origin, compute and serve a shortest tour through all currently known request. Its name comes from ignoring all incoming request while it is serving a tour. Ignore has a competitive ratio equal to $\frac{5}{2}$ [AKR00]. The SmartStart algorithm additionally makes sure that the next tour is not too long with respect to the current time, and otherwise waits. Intuitively, this fixes the worst-case instance of Ignore where, immediately after the server started a tour, requests arrive that could have been easily integrated into the current tour.

Theorem 10.5 (Ascheuer et al. [AKR00]). *SmartStart has a competitive ratio of at most* 2 *for the* OlDARP *problem.*

10.1.4 Further Related Work

Variants of OlTSP. For OlTSP as defined here, there are the mentioned best-possible 2-competitive algorithms [AKR00; Aus+01]. In our variant, the server has to return to the origin, which is in the literature also called *closed* OlTSP. In the *open* variant, the server does not have to return to the origin after serving the final request. For this variant, Bjelde et al. [Bje+21] prove a lower bound of ≈ 2.04 on the competitive ratio of deterministic algorithms, which already holds on the real line, and thereby improve an earlier lower bound of 2 [Aus+01]. For open OlTSP on general metrics, there is a $\frac{5}{2}$-competitive algorithm [Aus+01] and a recent 2.457-competitive algorithm by Baligács et al. [Bal+23a]. Jaillet and Wagner [JW08] presented 2-competitive algorithms for the closed OlTSP in a setting with multiple servers and precedence constraints between the requests. On the real line, there is a lower bound of ≈ 1.64 for open OlTSP [Aus+01]. Bjelde et al. [Bje+21] closed the gap for both variants on the line. They give a tight competitive ratio of ≈ 2.04 for the open variant and of ≈ 1.64 for the closed variant.

For closed OlDARP, the lower bound of 2 [Aus+01] translates to OlDARP [Aus+01; FS01]. On the positive side, SmartStart is 2-competitive also for closed OlDARP [AKR00] for any finite server capacity, and Feuerstein and Stougie [FS01] present an adaptation of the 2-competitive algorithm when the server has infinite capacity. For open OlDARP, there is a stronger lower bound of ≈ 2.05 [BDS23], which separates open OlDARP from open OlTSP. The best-known upper bound for open OlDARP on general metric is ≈ 2.457 [Bal+23a]. On special metric spaces, there are tighter results known, also for different variants [AKR00; Aus+01; Bal+23a; BD20; BDS23; Bje+21; Lip+04].

For both OlTSP and OlDARP, one can also minimize the average completion time of the requests instead of the makespan. This variant is also known as the *traveling repairperson problem*, and has been well-studied in different variants and environments [BKL21; BL19; BS09; FS01; HJ18; Kru+03].

Bampis et al. [Bam+23b] study OlTSP in the setting where the set of exact request locations are known in advance, but the requests still arrive online. They show that the competitive ratio for both open and closed OlTSP is equal to $\frac{3}{2}$.

Learning-augmented algorithms for OlTSP. After announcing our results [Ber+22b], independently of our work, two other papers [GLS22; Hu+22] were released that also study OlTSP in the learning-augmented setting. Hu et al. [Hu+22] consider OlTSP with different prediction models in general metric spaces. For arbitrary input predictions, their result has no error-dependency and a weaker consistency-robustness tradeoff compared to our main result for metric OlTSP. Gouleakis et al. [GLS22; GLS23] exclusively study OlTSP on the real line. Assuming that the correct number of requests is known in advance, they study the power of predictions *only* on the locations; their results are incomparable to ours.

Later, Chawla and Christou [CC24] consider OlTSP with time windows, where the goal is to serve as many requests as possible before their deadline. For this variant, no constant competitive algorithm is possible in the traditional online setting [AV16]. They use input predictions and present consistent, smooth and robust learning-augmented algorithms. Although we also use input predictions, their results are incomparable to ours because the problem itself is fundamentally different to ours.

Finally, Bampis et al. [Bam+23a] study both closed and open OlTSP in the same prediction model as Gouleakis et al. [GLS23], where only the locations of the requests are predicted. They present results for both general metric as well as various special cases. We achieve stronger bounds for general metrics as our prediction model is stronger than theirs.

10.2 A Tradeoff Lower Bound

Our first result is a lower bound of the consistency-robustness tradeoff of deterministic algorithms, which even holds on the half-line.

Theorem 10.6. *Let $\alpha \in (0, \frac{1}{2})$ and let $\mathcal{A}$ be a $(1+\alpha)$-consistent deterministic learning-augmented algorithm for OlTSP. Then, $\mathcal{A}$ can be β-robust only for $\beta \geq \frac{1}{\alpha} - 1$. This holds even on the half-line.*

Proof. Fix the half-line as metric space. Let $\varepsilon > 0$ be a small constant such that $\varepsilon \leq 1 - 2\alpha$. In the following, we consider two instances. The first instance consists of the two requests $\sigma_1 =$

$(0, 2\alpha + \varepsilon)$ and $\sigma_2 = (1, 1)$. Since $\alpha \leq \frac{1}{2}(1 - \varepsilon)$, in the optimal solution the server immediately moves to 1, serving σ_2 at time 1, and is back at the origin at time 2, serving σ_1. Suppose that algorithm $\mathcal{A}$ has access to a perfect prediction. Thus, it has to finish the instance within time $2(1 + \alpha)$ due to its consistency. We can make two observations on the behavior of algorithm $\mathcal{A}$. Firstly, $\mathcal{A}$ must serve σ_2 before σ_1, as otherwise, its server must be at the origin at time $2\alpha + \varepsilon$ and can finish the instance at the earliest at time $2(1 + \alpha) + \varepsilon$, a contradiction. Secondly, at time 1, $\mathcal{A}$'s server cannot be strictly to the left of the point $1 - 2\alpha$, otherwise again its consistency would be contradicted.

Now consider a second instance, consisting of the single request $\sigma = (0, 2\alpha + \varepsilon)$. Clearly, an optimal solution finishes at time $2\alpha + \varepsilon$. Suppose that algorithm $\mathcal{A}$ gets the same prediction as in the first instance. Since $\mathcal{A}$ is deterministic and the two instances are the same until time 1, it will behave the same as in the first instance until time 1. By our observations from the first instance, we conclude that at time 1, $\mathcal{A}$'s server is at least at distance $1 - 2\alpha$ from the origin, and it has not yet served σ_1. Thus, $\mathcal{A}$ can finish the second instance at the earliest at time $1 + 1 - 2\alpha$, which yields a robustness factor of at least $\frac{2-2\alpha}{2\alpha+\varepsilon}$. This concludes the proof. □

10.3 Online TSP on the Half-Line

We start with presenting our algorithmic results for the half-line metric space $X = \mathbb{R}_{\geq 0}$.

We first introduce a more compact prediction model than a prediction $\hat{R}$ on the set of requests R. That is, a single value PRD that predicts the makespan of an optimal tour OPT. The reason for this simplification is that we can compute for a given value PRD a optimal tour (assuming that the prediction is correct) by moving at time 0 to $\frac{1}{2}$PRD and then returning to the origin, as we have seen for Proposition 10.2. In particular, we could reduce any predicted input $\hat{R}$ via Proposition 10.2 to the single predicted request $\{(\frac{1}{2} \max_{(\hat{x},\hat{r})\in\hat{R}}\{2\hat{x}, \hat{x} + \hat{r}\}, 0)\}$ without changing an predicted optimal solution. Conversely, we can turn PRD into the input prediction $\{(\frac{1}{2}\text{PRD}, 0)\}$.

This insight also shows us that we can still apply the cover error in this model. The bipartite graph that we consider for the cover error has only a single vertex on each side, where the predicted instance is represented by $\{(\frac{1}{2}\text{PRD}, 0)\}$, and the actual instance is represented by $\{(\frac{1}{2}\text{OPT}, 0)\}$. Since the cost of a hyperedge $R' \cup \{x'\}$ should intuitively capture the optimal solution for instance R' with respect to x', we define the cost of the hyperedge $\{(x_1, r_1)\} \cup \{(x_2, r_2)\}$ to be equal to the additional cost of an optimal solution to serve (x_1, r_1) if it can optimally serve (x_2, r_2) for free. This concludes that we can express the value of a min-cost 1-hyperedge cover in this graph as follows.

Proposition 10.7. $\Lambda_1 = |\text{PRD} - \text{OPT}|$.

Proof.

$$\begin{aligned}\Lambda_1 &= \Gamma_1(R, \hat{R}) + \Gamma_1(\hat{R}, R)\\ &= 2 \cdot \max\left\{0, \frac{\text{OPT}}{2} - \frac{\text{PRD}}{2}\right\} + 2 \cdot \max\left\{0, \frac{\text{PRD}}{2} - \frac{\text{OPT}}{2}\right\} = |\text{PRD} - \text{OPT}|\,.\end{aligned}$$

□

We now introduce our algorithm, which combines the best-possible online algorithm Mrin (cf. Algorithm 8) with predictions in a three-stage framework. We call this algorithm MrinTrust and parameterize it with $\alpha \in (0, \frac{1}{2}]$, which controls our trust into the prediction. In the first phase, we execute for a time depending on PRD the classic Mrin strategy to ensure robustness in case that the prediction greatly overestimates the optimal cost. Then, we partially, depending on α, follow the optimal predicted strategy in Phase 2, and finish the instance again using Mrin in Phase 3. We note that a similar waiting strategy is also used in [GLS23] and [Bam+23b].

Algorithm 10: MrinTrust

1 Execute Mrin until time $\alpha \cdot \mathrm{PRD}$. Let $p_2 := p(\alpha \cdot \mathrm{PRD})$ be the final position of the server.
2 Move the server to the point $p_3 := \frac{1}{2}((1-\alpha) \cdot \mathrm{PRD} + p_2)$.
3 Execute Mrin again (starting from p_3).

We now prove the following main result of this section.

Theorem 10.8. *For any $\alpha \in (0, \frac{1}{2}]$, MrinTrust has a competitive ratio of at most*

$$\min\left\{(1+\alpha)\left(1+\frac{\Lambda_1}{\mathrm{OPT}}\right), \frac{3}{2\alpha}\right\}$$

for the OlTSP problem on the half-line.

Before proving both bounds separately in Lemmas 10.10 and 10.11, we observe that $p_2 \leq \alpha \cdot \mathrm{PRD}$ and $\alpha \in (0, \frac{1}{2}]$ imply

$$p_3 = \frac{1}{2}\left((1-\alpha) \cdot \mathrm{PRD} + p_2\right) \geq \frac{1}{2}\alpha \cdot \mathrm{PRD} + \frac{1}{2}p_2 \geq p_2\,,$$

which gives us the following observation.

Observation 10.9. *The servers position p_3 at the start of Phase 3 is not to the left of the server's position p_2 at the start of Phase 2, that is, $p_3 \geq p_2$.*

We next prove a bound on the robustness ratio of MrinTrust.

Lemma 10.10. *For any $\alpha \in (0, \frac{1}{2}]$, MrinTrust is $\frac{3}{2\alpha}$-competitive.*

Proof. Let ALG be the makespan of a tour determined by MrinTrust. If the algorithm terminates in Phase 1, then $\mathrm{ALG} \leq \frac{3}{2} \cdot \mathrm{OPT}$, because Mrin is $\frac{3}{2}$-competitive (Theorem 10.3). Otherwise, Phase 1 requires $\alpha \cdot \mathrm{PRD}$ time and Phase 2 requires $p_3 - p_2$ time. By denoting the time used in Phase 3 by C_3, it follows

$$\mathrm{ALG} = \alpha \cdot \mathrm{PRD} + (p_3 - p_2) + C_3\,. \tag{10.1}$$

Recall that in Phases 1 and 3 the algorithm follows the Mrin. Consider the execution of Mrin on the same instance. Due to Phase 2, in Phase 3 MrinTrust does not have to serve more requests than the ones served by Mrin after time $\alpha \cdot \mathrm{PRD}$. But, since MrinTrust starts Phase 3 at point p_3 and Mrin is at point $p_2 \leq p_3$ at time $\alpha \cdot \mathrm{PRD}$, Phase 3 may take additional time

equal to $p_3 - p_2$ compared to MRIN to move back if there are no requests to the right of p_2. As MRIN takes at most $\frac{3}{2}$OPT time for the instance, we conclude that

$$\alpha \cdot \text{PRD} + C_3 \leq \frac{3}{2}\text{OPT} + (p_3 - p_2) \,.$$

By Equation (10.1) and by the definition of p_3, we obtain

$$\text{ALG} \leq 2(p_3 - p_2) + \frac{3}{2}\text{OPT} \leq (1-\alpha)\text{PRD} + \frac{3}{2}\text{OPT} \,. \tag{10.2}$$

Since we assumed that MRINTRUST does not terminate in Phase 1, the time required by MRIN for the same instance is at least $\alpha \cdot \text{PRD}$. Thus, $\alpha \cdot \text{PRD} \leq \frac{3}{2}\text{OPT}$, and together with (10.2), we have

$$\text{ALG} \leq \frac{3(1-\alpha)}{2\alpha}\text{OPT} + \frac{3}{2}\text{OPT} = \frac{3}{2\alpha}\text{OPT} \,,$$

which concludes the proof. □

Lemma 10.11. *For any $\alpha \in (0, \frac{1}{2}]$, MRINTRUST is $(1+\alpha) \cdot \left(1 + \frac{\Lambda_1}{\text{OPT}}\right)$-competitive.*

Proof. We can assume that the instance consists of at least one request, as otherwise, we would immediately receive an end-signal in the origin and clearly achieve the stated competitive ratio. For the case where MRINTRUST reaches Phase 3, let t_3 denote the point in time when Phase 3 begins, that is, by the definition of p_2 and p_3,

$$t_3 = \alpha \cdot \text{PRD} + (p_3 - p_2) = \frac{1}{2}((1+\alpha)\text{PRD} - p_2) \,.$$

We denote again by ALG the cost given by MRINTRUST. We distinguish two cases. In each case we prove that $\text{ALG} \leq (1+\alpha)(\text{OPT} + |\text{PRD} - \text{OPT}|)$, which concludes the proof via Proposition 10.7.

Case $\text{OPT} \leq \text{PRD}$. We first show $\text{ALG} \leq (1+\alpha)\text{PRD}$. If the algorithm does not reach Phase 2, clearly $\text{ALG} \leq \alpha \cdot \text{PRD}$. Now suppose that the algorithm reaches Phase 2 (and therefore Phase 3) and the server is at position p_3 at time t_3. Observe that, in this case, every request released at time $t_3 + \delta$ cannot be to the right of $p_3 - \delta$ for any $0 \leq \delta \leq p_3$. Indeed, the existence of such a request $(x, t_3 + \delta)$ with $x > p_3 - \delta$ would imply

$$\begin{aligned}\text{OPT} &\geq x + t_3 + \delta > p_3 - \delta + t_3 + \delta \\ &= p_3 + t_3 = \frac{1}{2}((1-\alpha)\text{PRD} + p_2) + \frac{1}{2}((1+\alpha)\text{PRD} - p_2) = \text{PRD} \,,\end{aligned}$$

a contradiction. Therefore, the algorithm starts Phase 3 by serving all requests to the right of p_3 that were released before time t_3 (if there are any). Then, the server goes straight back to the origin while serving all remaining requests. Since all requests are in the interval $[0, \frac{1}{2}\text{OPT}] \subseteq [0, \frac{1}{2}\text{PRD}]$, the algorithm needs at most PRD time for Phases 2 and 3, giving a makespan of at most $(1+\alpha) \cdot \text{PRD}$. This gives us the desired bound:

$$\text{ALG} \leq (1+\alpha)\text{PRD} = (1+\alpha)\text{OPT} + (1+\alpha)(\text{PRD} - \text{OPT}) \,.$$

Case OPT > PRD. In this case, the algorithm must enter Phase 2, as otherwise ALG ≤ α·PRD < $\alpha \cdot$ OPT ≤ OPT contradicts that OPT is the optimal makespan. Thus, the server reaches position p_3 at time t_3, at the start of Phase 3. We claim that MrinTrust spends at most $B := \max\{\text{OPT} - \text{PRD}, \frac{1}{2}\text{OPT} - p_3\}$ time for moving *rightwards or for waiting at the origin after time* t_3. This allows us to bound the algorithm's makespan, given by the time t_3 spent on Phases 1 and 2, plus the time p_3 needed to go back to the origin from the point reached at the start of Phase 3, plus twice the time spent going to the right or sitting at the origin waiting in Phase 3, as follows:

$$\begin{aligned}
\text{ALG} &\le t_3 + p_3 + 2B = \frac{1}{2}((1+\alpha)\text{PRD} - p_2) + \frac{1}{2}((1-\alpha)\cdot\text{PRD} + p_2) + 2B \\
&= \text{PRD} + \max\{2\text{OPT} - 2\text{PRD}, \text{OPT} - (1-\alpha)\cdot\text{PRD} - p_2\} \\
&\le \max\{2\text{OPT} - \text{PRD}, \text{OPT} + \alpha\cdot\text{PRD}\} \\
&\le (1+\alpha)\text{OPT} + (\text{OPT} - \text{PRD}) ,
\end{aligned}$$

which proves the desired bound.

We prove the claim by contradiction, for which we assume that the algorithm exceeds the bound of B when serving a request (x, r). This request must exist, as otherwise the server would not move rightwards or wait at the origin. If $r \le t_3$, the server directly moves from p_3 to x at the beginning of Phase 3. Our assumption implies $x - p_3 > B$, and thus,

$$\text{OPT} \ge 2x > 2B + 2p_3 \ge 2\left(\frac{\text{OPT}}{2} - p_3\right) + 2p_3 = \text{OPT} ,$$

a contradiction. If $r > t_3$, denote by L the total time spent by the server moving leftwards between time t_3 and r. Thus, $r - t_3 - L$ is equal to the time the server spent moving rightwards or waiting in the origin between time t_3 and r. Let $p(r)$ be the position of the server at time r. Note that $p(r) \le x$. Due to our assumption, $x - p(r) + (r - t_3 - L) > B$. Since the server moved L units to the left after time t_3, it cannot be to the left of point $p_3 - L$ at time r, that is, $p(r) \ge p_3 - L$. We conclude

$$\begin{aligned}
\text{OPT} \ge x + r &> B - (r - t_3 - L) + p(r) + r \\
&\ge (\text{OPT} - \text{PRD}) - (r - t_3 - L) + (p_3 - L) + r \\
&= \text{OPT} - \text{PRD} + t_3 + p_3 = \text{OPT} ,
\end{aligned}$$

again a contradiction. □

We conclude this section by showing that the robustness factor $\frac{3}{2\alpha}$ is tight for our algorithm.

Lemma 10.12. *For any $\alpha \in (0, \frac{1}{2}]$, MrinTrust has a robustness factor at least $\frac{3}{2\alpha}$.*

Proof. Consider an instance consisting of a single request $\sigma = (\frac{1}{3}\alpha, \frac{1}{3}\alpha + \varepsilon)$ for a small $\varepsilon > 0$, and suppose that we give MrinTrust the prediction PRD = 1. The algorithm serves σ at time $\frac{2}{3}\alpha + \varepsilon$ and the server is at position ε at the start of Phase 2, that is, at time α. Thus, it does not receive an end-signal but the server reaches point $p_3 = \frac{1}{2}(1 - \alpha + \varepsilon)$ at time $\frac{1}{2}(1 + \alpha - \varepsilon)$. As there are no further requests, it moves back to the origin. Since an optimal solution serves σ

at time $\frac{1}{3}\alpha + \varepsilon$ and is back at the origin at time $\frac{2}{3}\alpha + \varepsilon$, we conclude that the prediction is not perfect, and that the robustness ratio of the algorithm is at least

$$\frac{\frac{1}{2}(1+\alpha-\varepsilon)+\frac{1}{2}(1-\alpha+\varepsilon)}{\frac{2}{3}\alpha+\varepsilon} = \frac{1}{\frac{2}{3}\alpha+\varepsilon},$$

which tends to $\frac{3}{2\alpha}$ as ε goes to 0. □

10.4 Online Metric TSP

We move to the OlTSP problem for general metric spaces. Our strategy is similar to the half-line case. We use an initial delay phase in which we follow a specific online algorithm up to some predetermined time depending on the cost PRD of an optimal tour $\hat{T}$ of the prediction $\hat{R}$. After that, we start following $\hat{T}$, adjusting it whenever the actual requests deviate from the predictions. We call this greedy strategy PredictReplan (PredReplan for short), due to the analogy with the classic Replan heuristic (see, for example, [AKR00]). While this algorithm might move towards predicted requests that are known to be absent to make the analysis clearer, a practical implementation ignores those and thereby only improves its performance.

Algorithm 11: PredReplan

1 Follow the optimal predicted tour $\hat{T}$.
2 Whenever an unexpected request (x, r) is released, recompute and follow a fastest tour from $p(r)$ (the current location of the server) to the origin serving all unserved predicted requests and all the unserved unexpected requests.
3 If the server receives an end signal in the origin, terminate.

We can now define our final algorithm SmartTrust. It is composed of three phases and uses both PredReplan and SmartStart (cf. Algorithm 9) as subroutines.

Algorithm 12: SmartTrust

1 Execute SmartStart with the following stopping criteria. If SmartStart decides to follow a tour S of length $\ell(S)$ at time t such that $t + \ell(S) > \alpha \cdot \text{PRD}$, go to Phase 2. If SmartStart sleeps or idles at time $\alpha \cdot \text{PRD}$, go to Phase 3.
2 Wait until time at least $\frac{\alpha}{2} \cdot \text{PRD}$, then go to Phase 3.
3 Follow the PredReplan strategy until the end.

In the remaining section, we analyze SmartTrust with respect to Λ_1. To this end, we define the cost $\gamma^{\text{TSP}}(R', (x', r'))$ of a hyperedge $R' \cup \{(x', r')\}$ in the cover error as the optimal makespan for serving instance R' from origin x' and initial time r'. We prove the following theorem.

Theorem 10.13. *For any $\alpha > 0$, SmartTrust has a competitive ratio of at most*

$$\min\left\{(1+\alpha)\left(1+\frac{2\cdot\Lambda_1}{\text{OPT}}\right), 2+\frac{2}{\alpha}\right\}$$

for the OlTSP problem.

We separately prove both bounds of Theorem 10.13 in Lemmas 10.14 and 10.15.

Lemma 10.14. *For any $\alpha \geq 0$, SmartTrust is $(1+\alpha)\left(1+2\cdot\frac{\Lambda_1}{\mathrm{OPT}}\right)$-competitive.*

Proof. We first bound PRD. Fix a min-cost ∞-hyperedge cover of $\hat{R}$ by R and an optimal tour T^* for R. For every hyperedge $\hat{R}' \cup \{(x,r)\}$ in the cover, we extend T^* by adding the optimal offline OlTSP tour for $\hat{R}'$ that starts at x at the time t at which T^* serves x. Note that, since $r \leq t$, the makespan of this subtour is bounded by the cost of $\hat{R}' \cup \{(x,r)\}$. Since every predicted request is covered by at least one hyperedge, the constructed tour serves $\hat{R}$ and we conclude that $\mathrm{PRD} \leq \mathrm{OPT} + \Gamma_\infty(\hat{R}, R)$.

We now bound the algorithm's makespan. If the algorithm terminates in Phases 1 or 2, its makespan is at most $\alpha \cdot \mathrm{PRD} \leq \alpha \cdot (\mathrm{OPT} + \Gamma_\infty(\hat{R}, R)) \leq (1+\alpha)\cdot(\mathrm{OPT} + \Lambda_1)$.

Otherwise, the algorithm reaches Phase 3. There, it first computes an optimal tour $\hat{T}$ of length at most PRD serving all unserved predicted requests. The makespan only increases when unexpected requests arrive. To this end, fix a min-cost 1-hyperedge cover of R by $\hat{R}$ and a hyperedge $\{(x',r')\} \cup \{(\hat{x},\hat{r})\}$ of this cover. We upper bound the additional cost due to (x',r') by the cost of an excursion that serves (x',r') from the algorithm's current tour. The algorithm might find a faster tour to serve all unserved requests and henceforth uses that. We distinguish two cases depending on the algorithm's remaining tour when request (x',r') arrives, and show that the total time for this excursion is bounded by $2 \cdot \gamma^{\mathrm{TSP}}(\{(x',r')\}, (\hat{x},\hat{r}))$.

The algorithm served $(\hat{x},\hat{r})$ at time t with $\hat{r} \leq t < r'$. We consider an excursion that immediately deviates from $p(r')$ to serve (x',r') and then returns to $p(r')$. By the triangle inequality, the length of this excursion is bounded by twice the distance between $p(r')$ and $\hat{x}$, plus twice the distance between x' and $\hat{x}$. Due to our assumption, the algorithm's server is at most $r' - \hat{r}$ units away from $\hat{x}$ at time r'. Therefore, the total time incurred for this excursion is bounded by $2\cdot(r'-\hat{r}) + 2\cdot d(x',\hat{x})$.

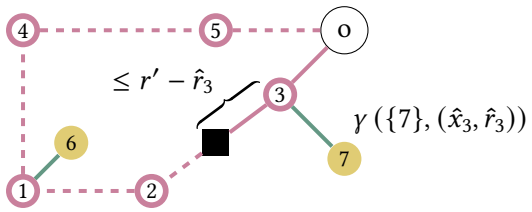
We now bound this value by $2 \cdot \gamma^{\mathrm{TSP}}(\{(x',r')\}, (\hat{x},\hat{r}))$. To this end, we again distinguish two cases. If $r' - \hat{r} \leq d(x',\hat{x})$, the optimal solution considered in the fixed hyperedge has no waiting time, that is, $\gamma^{\mathrm{TSP}}(\{(x',r')\}, (\hat{x},\hat{r})) = 2\cdot d(x',\hat{x})$. Hence, $2(r'-\hat{r}) + 2\cdot d(x',\hat{x}) \leq 4\cdot d(x',\hat{x}) = 2\gamma^{\mathrm{TSP}}(\{(x',r')\}, (\hat{x},\hat{r}))$. In the other case, that is, when $r' - \hat{r} > d(x',\hat{x})$, the optimal solution considered in the fixed hyperedge has to wait at some point for the release time r' of x', that is, $\gamma^{\mathrm{TSP}}(\{(x',r')\}, (\hat{x},\hat{r})) = (r'-\hat{r}) + d(x',\hat{x})$, yielding $2(r'-\hat{r}) + 2\cdot d(x',\hat{x}) = 2\cdot\gamma^{\mathrm{TSP}}(\{(x',r')\}, (\hat{x},\hat{r}))$.

The algorithm plans to serve $(\hat{x},\hat{r})$ at or after time r'. In this case, we wait with the excursion until the algorithm reaches $\hat{x}$ at some time $t \geq \hat{r}$, and then serve (x',r') using at most $2\cdot d(x',\hat{x}) \leq \gamma^{\mathrm{TSP}}(\{(x',r')\}, (\hat{x},\hat{r}))$ additional time.

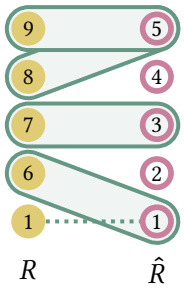
Figure 10.2 illustrates both cases.

Since every actual request is covered by one hyperedge, we conclude that Phase 3 takes time at most $\mathrm{PRD} + 2\cdot\Gamma_1(R,\hat{R})$. Adding the time for Phases 1 and 2 gives a makespan of at most

$$(1+\alpha)\mathrm{PRD} + 2\cdot\Gamma_1(R,\hat{R}) \leq (1+\alpha)\left(\mathrm{OPT} + \Gamma_\infty(\hat{R},R) + 2\cdot\Gamma_1(R,\hat{R})\right) \leq (1+\alpha)\left(\mathrm{OPT} + 2\cdot\Lambda_1\right),$$



(a) The server (square) plans (dashed) and follows (solid) the predicted tour. Two hyperedges are completely released: {7} is covered by 3, which is already served, and {6} is covered by 1, which will be served in the future. Note that neither 1, 8 nor 9 have been released yet. Also notice that PredReplan might find a faster tour.



(b) Min-cost 1-hyperedge cover of R

Figure 10.2: Illustration of the proof of Lemma 10.14.

which concludes the proof. □

Lemma 10.15. *For any $\alpha > 0$, SmartTrust is $(2 + \frac{2}{\alpha})$-competitive.*

Proof. Let ALG denote the makespan of SmartTrust's tour. If the algorithm terminates during Phase 1, then the competitive ratio is at most 2 [AKR00]. Suppose the algorithm enters Phases 2 and 3. Let C_S be the cost of SmartStart when serving the whole actual online instance. Since SmartStart is 2-competitive, it holds $C_S \leq 2\mathrm{OPT}$, and, moreover, because SmartTrust reaches Phase 3, is must be that $\alpha \cdot \mathrm{PRD} < C_S$. Thus, $\alpha \cdot \mathrm{PRD} < 2\mathrm{OPT}$.

Let t_3 be the time at which Phase 3 starts, t_{abort} be the time at which Phase 2 begins (that is, SmartStart is aborted at time t_{abort}), and r_{last} be the latest actual release date. We distinguish two cases.

Case $r_{\mathrm{last}} < t_3$. Since SmartTrust reaches Phase 3, it also entered Phase 2 at time t_{abort}. Let C_{abort} be the length of an optimal tour serving all actual requests that are unserved at time t_{abort}. Note that $t_{\mathrm{abort}} + C_{\mathrm{abort}} > \alpha \cdot \mathrm{PRD}$, as otherwise this tour would have been started in Phase 1. We further distinguish two subcases.

If $t_{\mathrm{abort}} \geq \frac{\alpha}{2} \cdot \mathrm{PRD}$, SmartTrust does not have to wait in Phase 2, and thus we have $t_{\mathrm{abort}} = t_3$. Observe that $t_{\mathrm{abort}} + C_{\mathrm{abort}} \leq C_S \leq 2 \cdot \mathrm{OPT}$. Since $r_{\mathrm{last}} < t_3 = t_{\mathrm{abort}}$, the length of Phase 3 is at most $\mathrm{PRD} + C_{\mathrm{abort}}$, and we conclude

$$\mathrm{ALG} \leq t_{\mathrm{abort}} + C_{\mathrm{abort}} + \mathrm{PRD} \leq 2 \cdot \mathrm{OPT} + \mathrm{PRD} \leq \left(2 + \frac{2}{\alpha}\right) \mathrm{OPT}\,.$$

Otherwise, it holds that $t_{\mathrm{abort}} < \frac{\alpha}{2} \cdot \mathrm{PRD}$. Let C_3 be the length of an optimal tour of all unserved actual requests at time t_3. Since we assume $r_{\mathrm{last}} < t_3$, Phase 3 takes at most $C_3 + \mathrm{PRD}$ time. Using that Phase 3 starts at time $\frac{\alpha}{2} \cdot \mathrm{PRD}$, we obtain

$$\begin{aligned}\mathrm{ALG} &\leq \frac{\alpha}{2}\mathrm{PRD} + C_3 + \mathrm{PRD} = C_3 + \left(1 + \frac{\alpha}{2}\right)\mathrm{PRD} \\ &\leq \mathrm{OPT} + \left(1 + \frac{\alpha}{2}\right)\frac{2}{\alpha}\mathrm{OPT} = \left(2 + \frac{2}{\alpha}\right)\mathrm{OPT}\,.\end{aligned}$$

Case $r_{\text{last}} \geq t_3$. Once the last request has arrived in Phase 3 at time r_{last}, our tour stays fixed. The cost of the algorithm after r_{last} is the cost of following the predicted tour, adapted for incorporating the unexpected, yet unserved, requests. This is bounded from above by the cost of returning to the origin, following the predicted tour, and finally following the optimal tour. Note that the cost for returning to the origin is at most $r_{\text{last}} - t_3$. Further, $r_{\text{last}} \leq \text{OPT}$ and Phase 2 ensures that $t_3 \geq \frac{\alpha}{2} \cdot \text{PRD}$. Hence,

$$\begin{aligned}\text{ALG} &\leq r_{\text{last}} + (r_{\text{last}} - t_3) + \text{PRD} + \text{OPT} \leq r_{\text{last}} + \left(r_{\text{last}} - \frac{\alpha}{2}\text{PRD}\right) + \text{PRD} + \text{OPT} \\ &\leq 2r_{\text{last}} + \left(1 - \frac{\alpha}{2}\right)\frac{2}{\alpha}\text{OPT} + \text{OPT} \leq \left(3 + \left(1 - \frac{\alpha}{2}\right)\frac{2}{\alpha}\right)\text{OPT} = \left(2 + \frac{2}{\alpha}\right)\text{OPT}\,.\end{aligned}$$

This concludes the proof of the lemma. □

Finally, we show that the robustness bound of SmartTrust given in Lemma 10.15 is tight.

Lemma 10.16. *For any $\alpha > 0$, SmartTrust has a robustness factor of at least $2 + \frac{2}{\alpha}$.*

Proof. Consider the metric space $X = \mathbb{R}$. Let $\hat{\sigma} = (-\frac{1}{2}, \frac{1}{2})$ be the only predicted request, while the only actual request is $\sigma = (\frac{\alpha}{4} + \varepsilon, \frac{\alpha}{4})$. Clearly, $\text{OPT} = \frac{\alpha}{2} + 2\varepsilon$ and $\text{PRD} = 1$. In Phase 1, SmartTrust executes SmartStart until time $\alpha \cdot \text{PRD} = \alpha$. Since the length of the tour that serves σ is equal to $\frac{\alpha}{2} + 2\varepsilon$, when σ is released SmartStart decides to sleep until time $\frac{\alpha}{2} + 2\varepsilon$. When SmartStart wakes up, the condition of Phase 1 forbids to execute the tour, and SmartTrust immediately goes to Phase 3. We conclude that at time $\frac{\alpha}{2} + 2\varepsilon$, SmartTrust is at the origin and σ is still unserved. Then, the algorithm needs at least $1 + \frac{\alpha}{2} + 2\varepsilon$ time to serve σ and follow the predicted tour. This gives a robustness factor of at least

$$\frac{\alpha/2 + 2\varepsilon + 1 + \alpha/2 + 2\varepsilon}{\alpha/2 + 2\varepsilon} = \frac{2\alpha + 2 + 8\varepsilon}{\alpha + 4\varepsilon}\,,$$

which tends to $2 + \frac{2}{\alpha}$ for arbitrarily small values of ε. □

10.5 Online Metric TSP with Polynomial Running Time

In this section, we introduce a variant of SmartTrust that runs in polynomial time. The only component in SmartTrust for which no polynomial-time algorithm is possible unless P = NP is finding optimal metric TSP tours. To this end, we instead use any ν-approximation algorithm for metric TSP and metric path TSP. The classic approximation algorithms by Christofides and Serdyukov [Chr76; Chr22; Ser78] for metric TSP have an approximate factor of $\frac{3}{2}$. We note that recent breakthroughs provide approximation factors slightly below $\frac{3}{2}$ [KKG21; KKG23]. Metric path TSP can be reduced to metric TSP at a loss of $(1 + \varepsilon)$ for any $\varepsilon > 0$ [TVZ22]. Thus, we can assume that there exists a better-than-$\frac{3}{2}$-approximation algorithm for metric path TSP.

Using such an ν-approximation algorithm for metric (path) TSP, it is known that SmartStart can be implemented to run in polynomial time, and has a competitive ratio of at most $\frac{1}{4}(4\nu + 1 + \sqrt{1 + 8\nu}) = \frac{1}{4}(7 + \sqrt{13}) \approx 2.6514$ [AKR00]. In our polynomial-time variant of SmartTrust, we therefore also use this polynomial-time variant of SmartStart in Phase 1.

Further, we have to adapt PredReplan to run in polynomial time. Since PredReplan relies on computing a solution to path TSP with release dates, we now use an ν-approximation algorithm for offline metric path TSP on the set of remaining requests (both predicted and unexpected) disregarding the release times. We then visit these requests and wait, in case of predicted requests, until the predicted release time, if required. We have the following simple proposition.

Proposition 10.17. *Given a polynomial time ν-approximation algorithm for offline metric path TSP without release times, we obtain a polynomial time $(1+\nu)$-approximation algorithm for offline metric path TSP with release times by following a ν-approximate path and wait whenever we arrive at a request early.*

Proof. The given algorithm returns a path of length at most $\nu \cdot \mathrm{OPT}$ in polynomial time. The total time spent waiting for the visited request to arrive is at most OPT, because no new requests arrive after time OPT. Therefore the algorithm has a makespan of at most $\nu \cdot \mathrm{OPT} + \mathrm{OPT}$. □

Using $(1+\nu)$-approximate paths instead of optimal paths whenever an unexpected request arrives can, however, break our argumentation for the error-dependency, because planned excursions might be changed. Therefore, we actively also compute the fastest excursion to an unexpected request from any predicted request in the current path. Finding such a predicted request $(\hat{x}, \hat{r})$ for every unexpected request (x, r) can be done efficiently in time $O(|R \setminus \hat{R}| \cdot |\hat{R}|)$. This ensures that we achieve an error-dependency with respect to Λ_1. However, it is not hard to see that this greedy approach does not offer robustness. To fix this, we additionally always recompute an $(1+\nu)$-approximate through all remaining actual and predicted requests, and follow the faster one of both paths.

Algorithm 13: Polynomial-time PredReplan

1 Initially, use Proposition 10.17 to compute and follow a tour on the predicted requests $\hat{R}$ of length PRD^*.
2 Whenever an unexpected request $(x, r) \in R \setminus \hat{R}$ appears, find the predicted request $(\hat{x}, \hat{r}) \in \hat{R}$ that minimizes $\gamma^{\mathrm{TSP}}(\{(\hat{x}, \hat{r})\}, (x, r))$. Modify the current remaining tour to the origin as follows. If $\hat{x}$ is part of the current remaining tour, add an excursion that starts at $\hat{x}$ at some time $t \geq \hat{r}$, serves (x, r) and finally returns to $\hat{x}$. Otherwise, add an immediate deviation from $p(r)$ to (x, r) and back to $p(r)$ on the current tour. Let T_1 be the computed tour. Additionally, compute a new $(1+\nu)$-approximate path T_2 from $p(r)$ to the origin through all unserved requests (including (x, r)) using Proposition 10.17. Follow the shorter tour in $\{T_1, T_2\}$.
3 If the server receives a signal in the origin, terminate.

For Phase 1 of polynomial-time SmartTrust, we first compute via Proposition 10.17 an $(1+\nu)$-approximate tour for $\hat{R}$, and denote its makespan by PRD^*.

Using these adaptations, we prove the following main theorem of this section.

Theorem 10.18. *For any $\alpha > 0$, the polynomial-time SmartTrust algorithm has a competitive ratio of at most*

$$\min\left\{(1+\nu)(1+\alpha)\left(1+\frac{\Lambda_1}{\mathrm{OPT}}\right), \max\{\rho, 1+\nu\} + \frac{\rho}{\alpha}\right\},$$

Algorithm 14: Polynomial-time SmartTrust

1 Execute SmartStart with the following stopping criteria. If SmartStart decides to follow a tour S of length $\ell(S)$ at time t such that $t + \ell(S) > \alpha \cdot \text{PRD}^*$, go to Phase 2. If SmartStart sleeps or idles at time $\alpha \cdot \text{PRD}^*$, go to Phase 3.
2 Wait until time at least $\frac{\alpha}{\rho} \cdot \text{PRD}^*$, then go to Phase 3.
3 Follow the polynomial-time PredReplan strategy until the end.

where $\rho = \rho(\nu)$ is the competitive ratio of the polynomial-time variant of SmartStart, and ν the approximation factor of a polynomial-time approximation algorithm for metric path TSP.

Using $\nu = \frac{3}{2}$, this bound is at most

$$\min\left\{\frac{5}{2}(1+\alpha)\left(1+\frac{\Lambda_1}{\text{OPT}}\right), 2.6514 \cdot \left(1+\frac{1}{\alpha}\right)\right\}.$$

In the following, we prove Theorem 10.18 by separately proving the error-dependent bound in Lemma 10.19 and the robustness bound in Lemma 10.20. We use PRD to denote the length of an optimal tour on the predicted requests.

Lemma 10.19. *For any $\alpha \geq 0$, the polynomial-time SmartTrust algorithm has a competitive ratio of at most $(1+\nu)(1+\alpha)\left(1+\frac{\Lambda_1}{\text{OPT}}\right)$.*

Proof. We first bound PRD. Fix a min-cost ∞-hyperedge cover of $\hat{R}$ by R and an optimal tour T^* for R. For every hyperedge $\hat{R}' \cup \{(x, r)\}$ in the cover, we extend T^* by adding the optimal offline OlTSP tour for $\hat{R}'$ that starts at x at the time t at which T^* serves x. Note that, since $r \leq t$, the makespan of this subtour is bounded by the cost of $\hat{R}' \cup \{(x, r)\}$. Since every predicted request is covered by at least one hyperedge, the constructed tour serves $\hat{R}$, and we conclude that $\text{PRD} \leq \text{OPT} + \Gamma_\infty(\hat{R}, R)$. Hence, for the length of the fixed approximate tour on all predicted requests holds $\text{PRD}^* \leq (1+\nu)\text{PRD} \leq (1+\nu)(\text{OPT} + \Gamma_\infty(\hat{R}, R))$.

We now bound the makespan of the tour of the algorithm. If the algorithm terminates in Phases 1 or 2, its makespan is at most $(1+\nu)\alpha \cdot \text{PRD} \leq (1+\nu)\alpha(\text{OPT} + \Gamma_\infty(\hat{R}, R)) \leq (1+\nu)(1+\alpha)(\text{OPT} + \Lambda_\infty)$.

Otherwise, the algorithm reaches Phase 3. There, it first computes a tour that serves all predicted requests of length PRD^*. Whenever an unexpected request (x', r') appears, the tour selection policy of the polynomial-time PredReplan algorithm ensures that the makespan of the remaining tour increases at most by the length of the computed excursion to serve (x', r'). Let $(\hat{x}, \hat{r})$ be the predicted request that the algorithm computes and uses for the excursion. We distinguish two cases depending on the algorithm's remaining tour when request (x', r') arrives, and show that the total time for this excursion is bounded by $2 \cdot \gamma^{\text{TSP}}(\{(x', r')\}, (\hat{x}, \hat{r}))$.

The algorithm served $(\hat{x}, \hat{r})$ at time t with $\hat{r} \leq t < r'$. In this case, the algorithm immediately deviates from $p(r')$ to serve (x', r') and then returns to $p(r')$. By the triangle inequality, the length of this excursion is bounded by twice the distance between $p(r')$ and $\hat{x}$, plus twice the distance between x' and $\hat{x}$. Since the algorithm's server is at most

$r' - \hat{r}$ units away from $\hat{x}$ at time r', the total time incurred for this excursion is bounded by $2 \cdot (r' - \hat{r}) + 2 \cdot d(x', \hat{x})$.

We now bound this value by $2 \cdot \gamma^{\mathrm{TSP}}(\{(x', r')\}, (\hat{x}, \hat{r}))$. To this end, we again distinguish two cases. If $r' - \hat{r} \leq d(x', \hat{x})$, the optimal solution considered in the fixed hyperedge has no waiting time, that is, $\gamma^{\mathrm{TSP}}(\{(x', r')\}, (\hat{x}, \hat{r})) = 2 \cdot d(x', \hat{x})$. Hence, $2(r' - \hat{r}) + 2 \cdot d(x', \hat{x}) \leq 4 \cdot d(x', \hat{x}) = 2\gamma^{\mathrm{TSP}}(\{(x', r')\}, (\hat{x}, \hat{r}))$. In the other case, that is, when $r' - \hat{r} > d(x', \hat{x})$, the optimal solution considered in the fixed hyperedge has to wait at some point for the release time r' of x', that is, $\gamma^{\mathrm{TSP}}(\{(x', r')\}, (\hat{x}, \hat{r})) = (r' - \hat{r}) + d(x', \hat{x})$, yielding $2(r' - \hat{r}) + 2 \cdot d(x', \hat{x}) = 2 \cdot \gamma^{\mathrm{TSP}}(\{(x', r')\}, (\hat{x}, \hat{r}))$.

The algorithm plans to serve $(\hat{x}, \hat{r})$ at or after time r'. Here, the algorithm serves (x', r') via an excursion from $\hat{x}$ at some time $t \geq \hat{r}$. This takes at most $\gamma^{\mathrm{TSP}}(\{(x', r')\}, (\hat{x}, \hat{r}))$ additional time.

By adding up the additional costs for all unexpected requests, we conclude that Phase 3 takes at most $\mathrm{PRD}^* + 2 \cdot \Gamma_1(R, \hat{R})$ time. Adding the time for Phases 1 and 2 and using that $\nu \geq 1$ implies that the algorithm's makespan is at most

$$\begin{aligned}(1+\alpha)\mathrm{PRD}^* + 2 \cdot \Gamma_1(R, \hat{R}) &\leq (1+\nu)(1+\alpha)\mathrm{PRD} + 2 \cdot \Gamma_1(R, \hat{R}) \\ &\leq (1+\nu)(1+\alpha)\left(\mathrm{OPT} + \Gamma_\infty(\hat{R}, R)\right) + 2 \cdot \Gamma_1(R, \hat{R}) \\ &\leq (1+\nu)(1+\alpha)\left(\mathrm{OPT} + \Lambda_1\right) \ .\end{aligned}$$

This concludes the proof of the lemma. □

Lemma 10.20. *For any $\alpha > 0$, the polynomial-time* SmartTrust *algorithm has a competitive ratio of at most* $\max\{\rho, 1+\nu\} + \frac{\rho}{\alpha}$.

Proof. Let ALG denote the makespan of SmartTrust's tour. Suppose the algorithm enters Phases 2 and 3. Let C_S be the cost of SmartStart when serving the whole actual online instance: as SmartStart is ρ-competitive, it holds $C_S \leq \rho\mathrm{OPT}$. Moreover, since SmartTrust reaches Phase 3, is must be that $\alpha \cdot \mathrm{PRD}^* < C_S$. Thus, $\alpha \cdot \mathrm{PRD}^* < \rho \cdot \mathrm{OPT}$.

Let t_3 be the time at which Phase 3 starts, t_{abort} be the time at which Phase 2 begins (that is, SmartStart is aborted at time t_{abort}), and r_{last} be the latest actual release date. We distinguish two cases.

Case $r_{\mathrm{last}} < t_3$. Since SmartTrust reaches Phase 3, it also entered Phase 2 at time t_{abort}. Let C_{abort} be the length of an ν-approximate tour serving all actual requests that are unserved at time t_{abort}. Note that $t_{\mathrm{abort}} + C_{\mathrm{abort}} > \alpha \cdot \mathrm{PRD}^*$, as otherwise this tour would have been started in Phase 1. We further distinguish two subcases.

If $t_{\mathrm{abort}} \geq \frac{\alpha}{\rho} \cdot \mathrm{PRD}^*$, SmartTrust does not have to wait in Phase 2, and thus, we have $t_{\mathrm{abort}} = t_3$. Observe that $t_{\mathrm{abort}} + C_{\mathrm{abort}} \leq C_S \leq \rho \cdot \mathrm{OPT}$. Since $r_{\mathrm{last}} < t_3 = t_{\mathrm{abort}}$, the length of Phase 3 is at most $\mathrm{PRD}^* + C_{\mathrm{abort}}$, and we conclude

$$\mathrm{ALG} \leq t_{\mathrm{abort}} + C_{\mathrm{abort}} + \mathrm{PRD}^* \leq \rho \cdot \mathrm{OPT} + \mathrm{PRD}^* \leq \left(\rho + \frac{\rho}{\alpha}\right)\mathrm{OPT} \ .$$

Otherwise, it holds that $t_{\text{abort}} < \frac{\alpha}{\rho} \cdot \text{PRD}$. Let C_3 be the length of an ν-approximate tour of all released but unserved actual requests at time t_3. Since we assume $r_{\text{last}} < t_3$, Phase 3 takes at most $C_3 + \text{PRD}^*$ time. Using that Phase 3 starts at time $\frac{\alpha}{\rho} \cdot \text{PRD}^*$, we obtain

$$\begin{aligned}\text{ALG} \leq \frac{\alpha}{2}\text{PRD}^* + C_3 + \text{PRD}^* = C_3 + \left(1 + \frac{\alpha}{2}\right)\text{PRD}^* &\leq \nu \cdot \text{OPT} + \left(1 + \frac{\alpha}{\rho}\right)\frac{\rho}{\alpha}\text{OPT} \\ &= \left(1 + \nu + \frac{\rho}{\alpha}\right) \cdot \text{OPT} .\end{aligned}$$

Case $r_{\text{last}} \geq t_3$. Once the last request has arrived in Phase 3 at time r_{last}, our tour stays fixed. The algorithm's cost after r_{last} is the cost of following the predicted $(1+\nu)$-approximate tour, adapted for incorporating the unexpected, yet unserved, requests. We can bound this by the cost of returning to the origin, following the predicted $(1+\nu)$-approximate tour and finally following an ν-approximate optimal tour through all actual released requests. Note that the cost for returning to the origin is at most $r_{\text{last}} - t_3$. Further, $r_{\text{last}} \leq \text{OPT}$ and Phase 2 ensures that $t_3 \geq \frac{\alpha}{\rho} \cdot \text{PRD}^*$. Hence,

$$\begin{aligned}\text{ALG} &\leq r_{\text{last}} + (r_{\text{last}} - t_3) + \text{PRD}^* + \nu \cdot \text{OPT} \\ &\leq r_{\text{last}} + \left(r_{\text{last}} - \frac{\alpha}{\rho}\text{PRD}^*\right) + \text{PRD}^* + \nu \cdot \text{OPT} \\ &\leq 2r_{\text{last}} + \left(1 - \frac{\alpha}{\rho}\right)\frac{\rho}{\alpha}\text{OPT} + \nu \cdot \text{OPT} \leq \left(1 + \nu + \frac{\rho}{\alpha}\right)\text{OPT} .\end{aligned}$$

This concludes the proof of the lemma. □

10.6 Online Metric Dial-a-Ride

In this section, we show that the techniques we have developed for OlTSP in Section 10.4 can also be used for the more general OlDARP. We consider a slight modification of SmartTrust from Section 10.4 that enables it for OlDARP. That is, we recompute tours in PredReplan only if the server is currently *not* carrying a request.

In the remaining part of this section, we explain how to adapt the proofs for Lemma 10.14 and Lemma 10.15 from OlTSP to OlDARP. But first we define the cost of a hyperedge for the cover error for OlDARP. By lifting the cost computed by γ^{TSP} to subinstances of OlDARP (that is, for transportation requests), we define the cost of a hyperedge $R' \cup \{(x^s, x^d, r)\}$ as

$$\gamma^{\text{DaRP}}(R', (x^s, x^d, r)) := \min\left\{\gamma^{\text{TSP}}(R', (x^s, r)), \gamma^{\text{TSP}}(R', (x^d, r + d(x^s, x^d)))\right\} + D ,$$

where $D := \max_{(x^s,x^d,r)\in R\cap\hat{R}} d(x^s, x^d)$. Further, for any correctly predicted request $(x^s, x^d, r) \in R \cap \hat{R}$, we set $\gamma^{\text{DaRP}}(\{(x^s, x^d, r)\}, (x^s, x^d, r)) = 0$.

We prove the following main theorem for OlDARP.

Theorem 10.21. *For any $\alpha > 0$, the (modified) SmartTrust algorithm has a competitive ratio of at most*

$$\min\left\{(1+\alpha)\left(1 + \frac{3 \cdot \Lambda_1}{\text{OPT}}\right), 2 + \frac{2}{\alpha}\right\}$$

for the OLDARP problem.

We first prove an error-dependent competitive ratio for our variant of SMARTTRUST.

Lemma 10.22. *For any $\alpha \geq 0$, SMARTTRUST is $(1+\alpha)\left(1+3\cdot\frac{\Lambda_1}{\mathrm{OPT}}\right)$-competitive for OLDARP.*

Proof. We first bound PRD. Fix a min-cost ∞-hyperedge cover of $\hat{R}$ by R and an optimal tour T^* for R. For every hyperedge $\hat{R}' \cup \{(x^s, x^d, r)\}$ in the cover, we extend T^* by adding the optimal offline OLDARP tour for $\hat{R}'$ that is considered in $\gamma^{\mathrm{DaRP}}(\hat{R}', (x^s, x^d, r))$. Since every predicted request is covered by at least one hyperedge, the constructed tour serves $\hat{R}$, and we conclude that $\mathrm{PRD} \leq \mathrm{OPT} + \Gamma_\infty(\hat{R}, R)$.

We now bound the makespan of the tour of the algorithm. If the algorithm terminates in Phases 1 or 2, its makespan is at most $\alpha \cdot \mathrm{PRD} \leq \alpha \cdot (\mathrm{OPT} + \Gamma_\infty(\hat{R}, R)) \leq (1+\alpha)\cdot(\mathrm{OPT} + \Lambda_1)$.

Otherwise, the algorithm reaches Phase 3. There, it first computes an optimal tour $\hat{T}$ serving all predicted requests of length PRD. The makespan only increases when unexpected requests arrive. To this end, fix a min-cost 1-hyperedge cover of R by $\hat{R}$ and a hyperedge $\{(x^s, x^d, r)\} \cup \{(\hat{x}^s, \hat{x}^d, \hat{r})\}$ of this cover. We upper bound the additional cost due to (x^s, x^d, r) by the cost of an excursion that serves (x^s, x^d, r) from the algorithm's current tour. The algorithm might find a faster tour to serve all remaining requests and henceforth uses that. We assume that the minimum in $\gamma^{\mathrm{DaRP}}(\{(x^s, x^d, r)\}, (\hat{x}^s, \hat{x}^d, \hat{r}))$ is attained by $\gamma^{\mathrm{TSP}}(\{(x^s, x^d, r)\}, (\hat{x}^s, \hat{r}))$, and only note differences for the other case. We distinguish two cases depending on the algorithm's remaining tour before request (x^s, x^d, r) arrived.

The algorithm served $(\hat{x}^s, \hat{x}^d, \hat{r})$ at time t, $r \geq t \geq \hat{r}$ ($r \geq t \geq \hat{r} + d(\hat{x}^s, \hat{x}^d)$). Consider an excursion that starts at the next point in time when the server reaches a point $\hat{x}$ of a predicted request, serves (x^s, x^d, r) and returns to $\hat{x}$. We now bound the additional cost for this excursion. If the server follows another excursion at time r, it must have visited point $\hat{x}$ before this excursion, so especially before time r, but after time $\hat{r}$ (respectively, $\hat{r} + d(\hat{x}^s, \hat{x}^d)$). If the server serves a correctly predicted request at time r, it will reach $\hat{x}$ (which is in this case the destination point of the currently served request) latest at time $r + D$. In both cases, the distance between $\hat{x}^s$ (respectively, $\hat{x}^d$) and $\hat{x}$ is at most $D + r - \hat{r}$ (respectively, $D + r - (\hat{r} + d(\hat{x}^s, \hat{x}^d))$). By the triangle inequality we observe that the length of the excursion for (x^s, x^d, r) is bounded by twice the distance from $\hat{x}$ to $\hat{x}^s$ (respectively, $\hat{x}^d$) and the cost for serving (x^s, x^d, r) from $\hat{x}^s$ at time $\hat{r}$ (respectively, $\hat{r} + d(\hat{x}^s, \hat{x}^d)$), that is

$$2 \cdot (D + r - \hat{r}) + \gamma^{\mathrm{TSP}}(\{(x^s, x^d, r)\}, (\hat{x}^s, \hat{r})) \leq 3 \cdot \gamma^{\mathrm{DaRP}}(\{(x^s, x^d, r)\}, (\hat{x}^s, \hat{x}^d, \hat{r}))\ .$$

Note that the inequality holds, because (x^s, x^d, r) can only be served after its release date, that is, $r - \hat{r} \leq \gamma^{\mathrm{TSP}}(\{(x^s, x^d, r)\}, (\hat{x}^s, \hat{r})) - D$ (respectively, $r - (\hat{r} + d(\hat{x}^s, \hat{x}^d)) \leq \gamma^{\mathrm{TSP}}(\{(x^s, x^d, r)\}, (\hat{x}^d, \hat{r} + d(\hat{x}^s, \hat{x}^d))) - D$).

The algorithm plans to visit $\hat{x}^s$ ($\hat{x}^d$) at or after time r'. In this case, we wait until the algorithm reaches $\hat{x}^s$ (respectively, $\hat{x}^d$) at some time $t \geq \hat{r}$ (respectively, $t \geq \hat{r} + d(\hat{x}^s, \hat{x}^d)$), and then serve (x^s, x^d, r) using at most $\gamma^{\mathrm{DaRP}}(\{(x^s, x^d, r)\}, (\hat{x}^s, \hat{x}^d, \hat{r}))$ additional time.

Since every actual request is covered by one hyperedge, we conclude that Phase 3 takes time at most $\mathrm{PRD} + 3 \cdot \Gamma_1(R, \hat{R})$. Adding the time for Phases 1 and 2 gives a makespan of at most

$$(1+\alpha)\mathrm{PRD} + 3 \cdot \Gamma_1(R, \hat{R}) \leq (1+\alpha)\left(\mathrm{OPT} + \Gamma_\infty(\hat{R}, R) + 3 \cdot \Gamma_1(R, \hat{R})\right) \leq (1+\alpha)\left(\mathrm{OPT} + 3 \cdot \Lambda_1\right).$$

This concludes the proof of the lemma. □

For the robustness bound of SmartTrust, note that at time r_{last} when the last request arrives the server might be in the process of serving a ride, which has remaining cost C_{ride} at time r_{last}. While we have to add this value to the time until the algorithm computes its final tour T_{final} due to our modifications, we can also subtract it from the length of T_{final}, because this particular ride is already served when computing T_{final}. Using the other arguments of the proof of Lemma 10.15, we conclude the following bound.

Lemma 10.23. *For any $\alpha > 0$, SmartTrust has a competitive ratio of at most $2 + \frac{2}{\alpha}$ for OlDARP.*

10.7 Application of the Cover Error to Online Network Design

In this final section of the chapter, we apply the cover error Λ_k to other problems and show that it improves guarantees of existing learning-augmented algorithms. More specifically, we consider two online network design problems, which are, compared to the *online-time* problem OlTSP and OlDARP, *online-list* problems. That is, the instance contains a list of requests, which are presented to the algorithm one-by-one. An algorithm has to serve the current request by executing a certain irrevocable action before the next request is revealed. Moreover, an online algorithm has no knowledge about future requests. We refer to the book of Borodin and El-Yaniv [BE98] for more details on online-list problems. We first define the problems that we consider in this section.

Online Steiner Tree Problem. In the *Online Steiner Tree Problem*, a sequence of vertices $R \subseteq V$ (called terminals) of a weighted graph $G = (V, E, c)$ with a distinct root ρ is revealed one-by-one. An algorithm maintains a solution S of selected edges and adds edges to S such that every arriving terminal is connected to ρ via edges in S. The objective is to minimize the total cost of selected edges $c(S) = \sum_{e \in S} c(e)$.

Online Steiner Forest Problem. In the *Online Steiner Forest Problem*, a sequence of vertex pairs $R \subseteq V \times V$ (called terminal pairs) of a weighted graph $G = (V, E, c)$ is revealed one-by-one. An algorithm maintains a solution S of selected edges and adds edges to S such that for every arriving terminal pair (s, t) the vertices s and t are connected via edges in S. The objective is to minimize the total cost of selected edges $c(S) = \sum_{e \in S} c(e)$. Note that this problem generalizes the online Steiner tree problem.

For the online Steiner tree problem and thus for the online Steiner forest problem, every deterministic online algorithm has a competitive ratio of at least $\Omega(\log |R|)$ [IW91]. For online Steiner tree, this competitive ratio is matched by the straightforward greedy algorithm, which adds the currently cheapest set of edges to connect the arriving request [IW91]. For

online Steiner forest, an online primal-dual algorithm also achieves a tight competitive ratio of $O(\log |R|)$ [BC97]. The greedy algorithm for online Steiner forest has a competitive ratio of at most $O(\log^2 |R|)$ [AAB04], and there has been recent progress on an improved analysis for special instances [BDM22].

In the context of learning-augmented algorithms, both problems have been studied by Azar et al. [APT22a] and Xu and Moseley [XM22]. Both works consider the input prediction model, where a set of predicted terminals (respectively, terminal pairs) $\hat{R}$ is given in advance, and give the following results.

Theorem 10.24 (Xu and Moseley [XM22])**.** *There exists a learning-augmented algorithm for the online Steiner tree problem with a competitive ratio of at most* $O(\log(\min\{|R|, \eta\}))$*, where* $\eta = \max\{|\hat{R}|, |R|\} - |\hat{R} \cap R|$.

Their algorithm achieves a best-possible robustness and is $O(1)$-consistent if $\hat{R} = R$.

Theorem 10.25 (Azar et al. [APT22a])**.** *There exists a learning-augmented algorithm for the online Steiner tree and online Steiner forest problem that has, for any* $T \subseteq R$ *and* $\hat{T} \subseteq \hat{R}$ *with* $|T| = |\hat{T}|$*, a competitive ratio of at most* $O(\log(\min\{|R|, \Delta\})) + \frac{D}{\mathrm{OPT}}$*, where* D *is the value of a min-cost perfect matching between* T *and* $\hat{T}$ *with respect to the shortest path metric of* G*, and* $\Delta = |R \setminus T| + |\hat{R} \setminus \hat{T}|$.

The interesting detail about this latter result is that it actually provides a family of performance guarantees, for every possible choice of T and $\hat{T}$. The intuition of this controllable error measure is that all (predicted) requests that are not in T or $\hat{T}$ are *outliers* that allow us to potentially match T with $\hat{T}$ in a very cheap way, hence giving a good performance guarantee. This algorithm also matches the best-possible robustness of $O(\log |R|)$ and is $O(1)$-consistent if $R = \hat{R}$.

The goal of this section is to analyze the learning-augmented algorithm of Azar et al. with respect to our cover error Λ_k. This shows that the cover error can also be applied to online-list problem. To this end, we use the following natural cost functions for hyperedges:

Steiner tree: $\gamma^{\mathrm{ST}}(R', x') :=$ cost of an optimal Steiner tree for terminals R' with root x'.

Steiner forest: $\gamma^{\mathrm{SF}}(R', x') :=$ cost of an optimal Steiner forest for terminal pairs R' when connecting via $x' = (s', t')$ is free, that is, there is an edge (s', t') in G with cost 0.

Then, we have the following theorem.

Theorem 10.26. *There exist learning-augmented algorithms that have, for every* $k \geq 1$*, a total cost of at most*

- $O(1) \cdot \mathrm{OPT} + O(\log k) \cdot \Lambda_k$ *for the online Steiner tree problem, and*
- $O(1) \cdot \mathrm{OPT} + O(k) \cdot \Lambda_k$ *for the online Steiner forest problem.*

Further, they have a robustness ratio of at most $O(\log |R|)$.

Algorithm 15: The Algorithm of Azar et al. [APT22a]

```
1 B ← 0, B̂ ← 0 and S ← ∅
2 Procedure UponRequest(r)
3     Send r to ON and augment S. Increase B by the cost incurred by ON for r.
4     if B ≥ 2B̂ then
5         B̂ ← B
6         Compute the smallest u such that 0 ≤ u ≤ |R̂| and c(Partial(R̂, u)) ≤ 3νB̂.
7         Augment S with the elements of Partial(R̂, u)
8         Start a new instance of the online algorithm ON where all edges in S have cost 0.
```

We emphasize that these bounds hold *simultaneously* for any $k \geq 1$. Before proving this theorem, we show the benefit of the cover error compared to the previous results.

We give a family of instances of the online Steiner tree problem with n actual requests and an input prediction for which the algorithms of Azar et al. [APT22a] and Xu and Moseley [XM22] perform arguably well, but their error measures and analyses yield a bound of $O(\log n) \cdot \mathrm{OPT} + O(\varepsilon)$, which could be guaranteed even without predictions. For some $\varepsilon > 0$, the instance is composed of one terminal request at x_1 and $n-1$ requests in an ε-ball around the Steiner point x_2, but no request is exactly on x_2. Both x_1 and x_2 are predicted, that is, $\hat{R} = \{x_1, x_2\}$. We can immediately observe that the error measure of Xu and Moseley evaluates in this case to $\eta = n - 1$. For the error of Azar et al., note that any perfect matching is composed of at most two matches, therefore $\Delta \geq n - 2$ and $D = O(\varepsilon)$. On the other hand, our cover error is bounded by $\Lambda_k \leq \Lambda_1 = O(n \cdot \varepsilon)$ for any k, because x_2 covers all requests in the ε-ball around it. Then, Theorem 10.26 implies that the algorithm of Azar et al. is indeed constant competitive for this instance when $\varepsilon \to 0$.

10.7.1 The Algorithm of Azar, Panigrahi, and Touitou

In this section, we introduce the algorithmic framework of Azar et al. [APT22a] and finally prove an auxiliary lemma that helps to abstract parts of the connection between their algorithm and our cover error. Then, we prove Theorem 10.26 separately for each problem.

Let $G = (V, E, c)$ be a weighted graph, $R \subseteq V$ be the actual input sequence, and $\hat{R} \subseteq V$ a predicted input. The framework (Algorithm 15) is the same for both, Steiner tree and Steiner forest. It combines in an iterative manner the execution of a plain online algorithm ON for the problem on the actual request set R with partial solutions for $\hat{R}$, which are computed using an ν-approximation algorithm PC for the price-collecting variant of the corresponding offline problem. In these variants, every request $r \in R$ has an associated penalty $\pi(r)$, and the goal is compute a solution $S \subseteq E$ that serves requests $R_S \subseteq R$ and minimizes $c(S) + \sum_{r \in R \setminus R_S} \pi(r)$. For a set of requests Q and a penalty function π, we denote the solution of PC for Q and π by $\mathrm{PC}(Q, \pi)$.

The execution of the framework for the ith request is called the ith *iteration*. Whenever in an iteration the total cost paid by the online algorithm so far doubles with respect to the last iteration it doubled (cf. Line 4), an offline solution for some part of $\hat{R}$ is added using the subroutine Partial (cf. Algorithm 16). Such an iteration is called *major* iteration, and we call

Algorithm 16: The PARTIAL Subroutine [APT22a]

Subroutine $\textsc{Partial}(\hat{R}, u)$
  if $v \cdot u \geq |\hat{R}|$ **then return** $\emptyset$
  For every integer x, let π_x be the penalty function such that $\forall r \in \hat{R} : \pi_x(r) = x$.
  Let i be the minimum integer s.t. $\mathrm{PC}(\hat{R}, \pi_{2^i})$ does not satisfy at most $v \cdot u$ requests.
  Let $S_1 = \mathrm{PC}(\hat{R}, \pi_{2^{i-1}})$ and $S_2 = \mathrm{PC}(\hat{R}, \pi_{2^i})$, and u_1, u_2 be the corresponding number of request from $\hat{R}$ that are not satisfied by S_1, S_2, respectively.
  if $v \cdot u \geq \frac{1}{2}(u_1 + u_2)$ **then return** S_1 **else return** S_2

the sequence of the following iterations until the next major iteration (including this next one) a *phase*. Let m be the total number of phases. We denote by Q_j the set of requests served by the online algorithm in phase j, and for any $Q'_j \subseteq Q_j$, $\mathrm{ON}_j(Q'_j)$ is the total cost of the online algorithm incurred to serve the requests in Q'_j in phase j. We write ON_j for $\mathrm{ON}_j(Q_j)$. Note that the online algorithm is always allowed to use the augmented solutions by the prediction with zero cost. Further, $\hat{B}_i$ denotes the total cost of the online algorithm until the latest previous major iteration before iteration i.

We now state two auxiliary results of Azar et al. for their framework. The first lemma characterized solutions returned by PARTIAL.

Lemma 10.27 ([APT22a]). *The solution S returned by $\textsc{Partial}(\hat{R}, u)$ has the following properties:*

(1) S satisfies at least $|\hat{R}| - 2v \cdot u$ requests of $\hat{R}$, and

(2) $c(S) \leq 3v \cdot c(S^*)$, where S^* is the minimum cost solution satisfying at least $|R| - u$ requests from $\hat{R}$.

The next lemma allows us to bound the total cost of the framework by bounding two separate parts, which are defined by splitting the execution at any major iteration.

Lemma 10.28 ([APT22a]). *Fix any major iteration i. Then,*

(1) the total cost for iterations $1, \ldots, i$ is at most $O(1) \cdot \mathrm{OPT} + O(1) \cdot \hat{B}_{i-1}$, and

(2) the total cost for iterations $i+1, \ldots, |R|$ is at most $O(1) \cdot \max\{\mathrm{ON}_{j-1}, \mathrm{ON}_j\}$.

Our goal is to apply Lemma 10.28 to the major iteration i in which all predicted requests become satisfied by our solution S (if such an iteration exists). Then, we can use a ∞-hyperedge cover of $\hat{R}$ by R to bound to the total cost of the first i iterations, and a k-hyperedge cover of R by $\hat{R}$ to bound the total cost of the remaining iterations. Since this idea depends on ON and the definition of the hyperedge costs, we formulate it in the following conditional lemma, which we later apply for each concrete problem.

Lemma 10.29. *Suppose that for some function $\mu : \mathbb{N} \to \mathbb{R}^+$ and some integer $k \geq 1$ it holds that*

(a) $\mathrm{OPT}(\hat{R}) \leq \mathrm{OPT} + \Gamma_\infty(\hat{R}, R)$, and

(b) if there is a major iteration i in which all predicted requests become served, then we require $\mathrm{ON}_j(H) \leq O(\mu(k)) \cdot \gamma(H, \hat{x})$ for any $H \subseteq Q_j$ with $|H| \leq k$, $j \in \{m-1, m\}$ and $\hat{x} \in \hat{R}$.

Then, the total cost of Algorithm 15 is at most $O(1) \cdot \mathrm{OPT} + O(\mu(k)) \cdot \Lambda_k$.

Proof. If there exists a major iteration in which all predicted requests become satisfied, let this be the ith iteration. Otherwise, let i be the last iteration. By Lemma 10.28, it suffices to bound $\hat{B}_{i-1}$ and $\max\{\mathrm{ON}_{j-1}, \mathrm{ON}_j\}$ to get an overall bound on the algorithm's total cost.

We start with $\hat{B}_{i-1}$. Let $i' < i$ be the iteration in which $\hat{B}_{i-1}$ was set in Line 5. Thus, some predicted requests were not satisfied by Partial in iteration i'. By the definition of the algorithm, we conclude that the total cost Partial must pay for satisfying all predicted requests in iteration i', $c(\textsc{Partial}(\hat{R}, 0))$, is strictly larger than $3\nu \cdot \hat{B}_{i'} = 3\nu \cdot \hat{B}_{i-1}$. By Lemma 10.27, $\textsc{Partial}(\hat{R}, 0)$ approximates the cheapest solution that satisfies all predicted requests within a factor of 3ν. This implies that the optimal solution for $\hat{R}$, which is such a solution, has cost of at least $\hat{B}_{i-1}$, that is, $\mathrm{OPT}(\hat{R}) \geq \hat{B}_{i-1}$. Condition (a) therefore implies

$$\hat{B}_{i-1} \leq \mathrm{OPT}(\hat{R}) \leq \mathrm{OPT} + \Gamma_\infty(\hat{R}, R) \,. \tag{10.3}$$

Second, we bound $\max\{\mathrm{ON}_{m-1}, \mathrm{ON}_m\}$, and assume that all predicted requests $\hat{R}$ are satisfied in iteration i. Let $j \in \{m-1, m\}$. Fix a min-cost k-hyperedge cover of R by $\hat{R}$, a hyperedge $R' \cup \{\hat{x}\}$ of the cover, and let $H_j = Q_j \cap R'$ be the subset of requests of the hyperedge that ON_j serves in phase j. Since $|R'| \leq k$, we have $|H_j| \leq k$, and thus, Condition (b) implies

$$\mathrm{ON}_j(H_j) \leq O(\mu(k)) \cdot \gamma(H_j, \hat{x}) \,.$$

By the choice of the hyperedge cover, every request in Q_j is in at least one hyperedge. Therefore, summing over all hyperedges of the cover yields

$$\mathrm{ON}_j(Q_j) \leq O(\mu(k)) \cdot \Gamma_k(R, \hat{R}) \,,$$

and thus,

$$\max\{\mathrm{ON}_{m-1}, \mathrm{ON}_m\} \leq O(\mu(k)) \cdot \Gamma_k(R, \hat{R}) \,. \tag{10.4}$$

Combining (10.3) and (10.4) with Lemma 10.28 implies that the total cost is at most

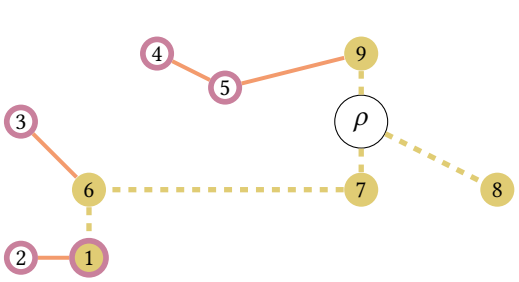
$$O(1) \cdot \mathrm{OPT} + O(\Gamma_\infty(\hat{R}, R)) + O(\mu(k)) \cdot \Gamma_k(R, \hat{R}) \leq O(1) \cdot \mathrm{OPT} + O(\mu(k)) \cdot \Lambda_k \,,$$

which concludes the proof of the lemma. □

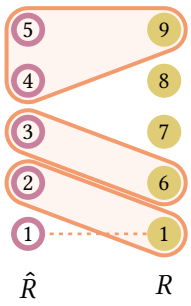
10.7.2 Online Steiner Tree

The goal of this section is to configure Algorithm 15 for the online Steiner tree problem, and prove Lemma 10.29 with $\mu(k) = \log k$, which implies the first bound of Theorem 10.26.

As baseline online algorithm we use the Greedy algorithm [IW91], which connects an arriving terminal to the root using the shortest path to the current solution. We denote it by $\mathrm{ON}^{\mathrm{ST}}$. As approximation algorithm for the price-collecting Steiner tree problem, we use the algorithm by Goemans and Williamson [GW95]. These algorithms are also used in [APT22a]. For the cover error, we define the cost $\gamma^{\mathrm{ST}}(R', x')$ of a hyperedge $R' \cup \{x'\}$ as the cost of an optimal offline Steiner Tree for terminals R' with root x'.



(a) The optimal Steiner tree for R (dashed), and the augmented Steiner tree for $\hat{R}$.



(b) Min-cost k-hyperedge cover of $\hat{R}$

Figure 10.3: Augmentation of an optimal Steiner tree for R to a Steiner tree that serves $\hat{R}$ using solutions of subinstances (solid) induced by hyperedges of a min-cost k-hyperedge cover of $\hat{R}$.

Then, we verify the conditions of Lemma 10.29 in the following two lemmas, which implies that the total cost of the algorithm is, for any integer $k \geq 1$, at most $O(1) \cdot \mathrm{OPT} + O(\log k) \cdot \Lambda_k$.

Lemma 10.30. $\mathrm{OPT}(\hat{R}) \leq \mathrm{OPT} + \Gamma_\infty(\hat{R}, R)$.

Proof. Consider a min-cost ∞-hyperedge cover of $\hat{R}$ by R. For every hyperedge $\hat{R}' \cup \{x\}$ in the cover, we connect $x \in R$ with all predicted requests in $\hat{R}'$ using the Steiner tree considered in $\gamma^{\mathrm{ST}}(\hat{R}', x)$. Adding an optimal Steiner Tree for the terminal set R thus also satisfies all requests in $\hat{R}$ (cf. Figure 10.3), and since the augmentation cost are at most the hyperedge cost, we conclude that $\mathrm{OPT}(\hat{R}) \leq \mathrm{OPT} + \Gamma_\infty(\hat{R}, R)$. □

Lemma 10.31. *If there is a major iteration i in which all predicted requests become served, then it holds that $\mathrm{ON}_j(H) \leq O(\log k) \cdot \gamma(H, \hat{x})$ for any $H \subseteq Q_j$ with $|H| \leq k$, $j \in \{m-1, m\}$ and $\hat{x} \in \hat{R}$.*

Proof. Assume that there is such a major iteration i, and let $j \in \{m-1, m\}$, $H_j \subseteq Q_j$ such that $|H_j| \leq k$ and $\hat{x} \in \hat{R}$. We first argue that $\mathrm{ON}_j^{\mathrm{ST}}(H_j) \leq O(\log k) \cdot \mathrm{OPT}_j(H_j)$, where $\mathrm{OPT}_j(H_j)$ is the value of the optimal Steiner Tree for H_j in which edges that were bought before phase j have zero cost. This is because for each terminal $r \in H_j$, the cheapest path to a previous terminal in an instance with terminals R is at most as expensive as the cheapest path to a previous terminal in an instance with terminals H_j, and the greedy algorithm is $O(\log|H_j|) \leq O(\log k)$ competitive in an instance with terminal H_j. This argumentation has also been used by Azar et al. [APT22a, Proof of Lemma 3.2].

Since we assume that $\hat{x}$ has already been served in or before iteration i, a feasible solution for H_j in phase j is given by connecting H_j optimally to $\hat{x}$. Therefore, $\mathrm{OPT}_j(H_j) \leq \gamma^{\mathrm{ST}}(H_j, \hat{x})$, and thus,

$$\mathrm{ON}_j^{\mathrm{ST}}(H_j) \leq O(\log k) \cdot \mathrm{OPT}_j(H_j) \leq O(\log k) \cdot \gamma^{\mathrm{ST}}(H_j, \hat{x}) \,,$$

which concludes the proof of the lemma. □

10.7.3 Online Steiner Forest

The goal of this section is to configure Algorithm 15 for the online Steiner forest problem, and prove Lemma 10.29 with $\mu(k) = k$, which implies the second bound of Theorem 10.26.

As baseline online algorithm, Azar et al. [APT22a] introduces a variant of the $O(\log |R|)$-competitive online algorithm of Berman and Coulston [BC97], which we denote by $\mathrm{ON}^{\mathrm{SF}}$. As approximation algorithm for the price-collecting Steiner tree problem, we use the algorithm by Hajiaghayi and Jain [HJ06]. For the cover error, we define the cost $\gamma^{\mathrm{SF}}(R', (s', t'))$ of a hyperedge $R' \cup \{(s', t')\}$ as the value of the optimal offline Steiner forest for terminal pairs R' where the distance between s' and t' has zero cost.

We verify the conditions of Lemma 10.29 in the following two lemmas, which implies that the total cost of the algorithm is, for any integer $k \geq 1$, at most $O(1) \cdot \mathrm{OPT} + O(k) \cdot \Lambda_k$.

Lemma 10.32. $\mathrm{OPT}(\hat{R}) \leq \mathrm{OPT} + \Gamma_\infty(\hat{R}, R)$.

Proof. Fix an optimal solution $\mathrm{OPT}(R)$ for R and fix a min-cost ∞-hyperedge cover of $\hat{R}$ by R. For every hyperedge $\hat{R}' \cup \{(s, t)\}$ in the cover, we augment $\mathrm{OPT}(R)$ with the solution of the subinstance that is considered in $\gamma^{\mathrm{SF}}(\hat{R}', (s, t))$. Note that any terminal pair $(\hat{s}, \hat{t}) \in \hat{R}$ that is connected via connecting to s and t in this solution, remains connected, because s and t are connected in the solution of R. Since every client in $\hat{R}$ is contained in at least one hyperedge, the final solution satisfies $\hat{R}$, and we conclude $\mathrm{OPT}(\hat{R}) \leq \mathrm{OPT} + \Gamma_\infty(\hat{R}, R)$. □

Lemma 10.33. *If there is a major iteration i in which all predicted requests become served, then it holds $\mathrm{ON}_j(H) \leq O(\log k) \cdot \gamma(H, \hat{x})$ for any $H \subseteq Q_j$ with $|H| \leq k$, $j \in \{m-1, m\}$ and $\hat{x} \in \hat{R}$.*

Proof. Assume that there is such a major iteration i, and let $j \in \{m-1, m\}$, $H_j \subseteq Q_j$ such that $|H_j| \leq k$ and $\hat{x} \in \hat{R}$. For every $(s, t) \in H_j$, $\mathrm{ON}^{\mathrm{SF}}$ pays at most twice the distance to any previous request [APT22a, Proof of Lemma 4.3]. Since we are assuming that $(\hat{s}, \hat{t})$ has already been connected before Phase j, we conclude that $\mathrm{ON}^{\mathrm{SF}}_j(\{(s, t)\}) \leq 2 \cdot \gamma^{\mathrm{SF}}(\{(s, t)\}, (\hat{s}, \hat{t}))$. Summing over all requests in H_j gives

$$\mathrm{ON}^{\mathrm{SF}}_j(H_j) \leq 2k \cdot \max_{(s,t) \in H_j} \gamma^{\mathrm{SF}}(\{(s, t)\}, (\hat{s}, \hat{t})) \leq 2k \cdot \gamma^{\mathrm{SF}}(H_j, (\hat{s}, \hat{t})) = O(k) \cdot \gamma^{\mathrm{SF}}(H_j, (\hat{s}, \hat{t})) \,,$$

which concludes the proof of the lemma. □

10.8 Concluding Remarks

In this chapter, we showed that additional information on the input can drastically improve the performance of an online algorithm for OLTSP. Moreover, our algorithms achieve robustness using a delay strategy. Further, we extended these results to OLDARP and polynomial-time algorithms, and showed improved bounds for the half-line metric.

We introduced the cover error, a universal error measure for input predictions. We showed that the cover error can be applied to both, online-time routing problems and online-list network design problems. For the latter, the cover error was able to give tighter bounds for some instances and predictions compared to previous prediction errors. It would be interesting to see whether the cover error can also successfully be applied to other problems.

Bibliographic Note

This chapter is based on joint work with Giulia Bernardini, Alberto Marchetti-Spaccamela, Nicole Megow, Leen Stougie, and Michelle Sweering [Ber+22a]. Thus, some parts of this chapter are identical with [Ber+22a]. Some parts of [Ber+22a] also appear in the Ph.D. thesis of Michelle Sweering.

Bibliography

[Afr+99] Foto N. Afrati, Evripidis Bampis, Chandra Chekuri, David R. Karger, Claire Kenyon, Sanjeev Khanna, Ioannis Milis, Maurice Queyranne, Martin Skutella, Clifford Stein, and Maxim Sviridenko. *Approximation Schemes for Minimizing Average Weighted Completion Time with Release Dates*. In: *FOCS*. IEEE Computer Society, 1999, pp. 32–44.

[Agr+22] Priyank Agrawal, Eric Balkanski, Vasilis Gkatzelis, Tingting Ou, and Xizhi Tan. *Learning-Augmented Mechanism Design: Leveraging Predictions for Facility Location*. In: *EC*. ACM, 2022, pp. 497–528.

[AE20] Susanne Albers and Alexander Eckl. *Explorable Uncertainty in Scheduling with Non-uniform Testing Times*. In: *WAOA*. Vol. 12806. Lecture Notes in Computer Science. Springer, 2020, pp. 127–142.

[AS01] Susanne Albers and Günter Schmidt. *Scheduling with unexpected machine breakdowns*. In: *Discret. Appl. Math.* 110.2-3 (2001), pp. 85–99.

[All70] Frances E Allen. *Control flow analysis*. In: *ACM Sigplan Notices* 5.7 (1970), pp. 1–19.

[AM09] Christoph Ambühl and Monaldo Mastrolilli. *Single Machine Precedence Constrained Scheduling Is a Vertex Cover Problem*. In: *Algorithmica* 53.4 (2009), pp. 488–503.

[Ana+22] Keerti Anand, Rong Ge, Amit Kumar, and Debmalya Panigrahi. *Online Algorithms with Multiple Predictions*. In: *ICML*. Vol. 162. Proceedings of Machine Learning Research. PMLR, 2022, pp. 582–598.

[AGK12] S. Anand, Naveen Garg, and Amit Kumar. *Resource augmentation for weighted flow-time explained by dual fitting*. In: *SODA*. SIAM, 2012, pp. 1228–1241.

[AP04] Edward J. Anderson and Chris N. Potts. *Online Scheduling of a Single Machine to Minimize Total Weighted Completion Time*. In: *Math. Oper. Res.* 29.3 (2004), pp. 686–697.

[ALN15] Spyros Angelopoulos, Giorgio Lucarelli, and Kim Thang Nguyen. *Primal-Dual and Dual-Fitting Analysis of Online Scheduling Algorithms for Generalized Flow Time Problems*. In: *ESA*. Vol. 9294. Lecture Notes in Computer Science. Springer, 2015, pp. 35–46.

[Ann19] Chidambaram Annamalai. *Lazy Local Search Meets Machine Scheduling*. In: *SIAM J. Comput.* 48.5 (2019), pp. 1503–1543.

[AKS17] Chidambaram Annamalai, Christos Kalaitzis, and Ola Svensson. *Combinatorial Algorithm for Restricted Max-Min Fair Allocation*. In: *ACM Trans. Algorithms* 13.3 (2017), 37:1–37:28.

[Ant+23a] Antonios Antoniadis, Joan Boyar, Marek Eliás, Lene Monrad Favrholdt, Ruben Hoeksma, Kim S. Larsen, Adam Polak, and Bertrand Simon. *Paging with Succinct Predictions*. In: *ICML*. Vol. 202. Proceedings of Machine Learning Research. PMLR, 2023, pp. 952–968.

[Ant+20] Antonios Antoniadis, Christian Coester, Marek Eliás, Adam Polak, and Bertrand Simon. *Online metric algorithms with untrusted predictions*. In: *ICML*. Vol. 119. Proceedings of Machine Learning Research. PMLR, 2020, pp. 345–355.

[Ant+23b] Antonios Antoniadis, Christian Coester, Marek Eliás, Adam Polak, and Bertrand Simon. *Online Metric Algorithms with Untrusted Predictions*. In: *ACM Trans. Algorithms* 19.2 (2023), 19:1–19:34.

[AJS22] Antonios Antoniadis, Peyman Jabbarzade, and Golnoosh Shahkarami. *A Novel Prediction Setup for Online Speed-Scaling*. In: *SWAT*. Vol. 227. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 9:1–9:20.

[Ara+18] Luciana Arantes, Evripidis Bampis, Alexander V. Kononov, Manthos Letsios, Giorgio Lucarelli, and Pierre Sens. *Scheduling under Uncertainty: A Query-based Approach*. In: *IJCAI*. ijcai.org, 2018, pp. 4646–4652.

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

[AFS12] Arash Asadpour, Uriel Feige, and Amin Saberi. *Santa claus meets hypergraph matchings*. In: *ACM Trans. Algorithms* 8.3 (2012), 24:1–24:9.

[AKR00] Norbert Ascheuer, Sven Oliver Krumke, and Jörg Rambau. *Online Dial-a-Ride Problems: Minimizing the Completion Time*. In: *STACS*. Vol. 1770. Lecture Notes in Computer Science. Springer, 2000, pp. 639–650.

[Asp+97] James Aspnes, Yossi Azar, Amos Fiat, Serge A. Plotkin, and Orli Waarts. *Online routing of virtual circuits with applications to load balancing and machine scheduling*. In: *J. ACM* 44.3 (1997), pp. 486–504.

[Aus+01] Giorgio Ausiello, Esteban Feuerstein, Stefano Leonardi, Leen Stougie, and Maurizio Talamo. *Algorithms for the On-Line Travelling Salesman*. In: *Algorithmica* 29.4 (2001), pp. 560–581.

[AA03] Nir Avrahami and Yossi Azar. *Minimizing total flow time and total completion time with immediate dispatching*. In: *SPAA*. ACM, 2003, pp. 11–18.

[AA07] Nir Avrahami and Yossi Azar. *Minimizing Total Flow Time and Total Completion Time with Immediate Dispatching*. In: *Algorithmica* 47.3 (2007), pp. 253–268.

[AAB04] Baruch Awerbuch, Yossi Azar, and Yair Bartal. *On-line generalized Steiner problem*. In: *Theor. Comput. Sci.* 324.2-3 (2004), pp. 313–324.

[ACL18] Yossi Azar, Jaya Prakash Champati, and Ben Liang. *2-Approximation algorithm for a generalization of scheduling on unrelated parallel machines*. In: *Inf. Process. Lett.* 139 (2018), pp. 39–43.

[Aza+15] Yossi Azar, Inna Kalp-Shaltiel, Brendan Lucier, Ishai Menache, Joseph Naor, and Jonathan Yaniv. *Truthful Online Scheduling with Commitments*. In: *EC*. ACM, 2015, pp. 715–732.

[ALT21] Yossi Azar, Stefano Leonardi, and Noam Touitou. *Flow time scheduling with uncertain processing time*. In: *STOC*. ACM, 2021, pp. 1070–1080.

[ALT22] Yossi Azar, Stefano Leonardi, and Noam Touitou. *Distortion-Oblivious Algorithms for Minimizing Flow Time*. In: *SODA*. SIAM, 2022, pp. 252–274.

[ANR95] Yossi Azar, Joseph Naor, and Raphael Rom. *The Competitiveness of On-Line Assignments*. In: *J. Algorithms* 18.2 (1995), pp. 221–237.

[APT22a] Yossi Azar, Debmalya Panigrahi, and Noam Touitou. *Online Graph Algorithms with Predictions*. In: *SODA*. SIAM, 2022, pp. 35–66.

[APT22b] Yossi Azar, Eldad Peretz, and Noam Touitou. *Distortion-Oblivious Algorithms for Scheduling on Multiple Machines*. In: *ISAAC*. Vol. 248. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 16:1–16:18.

[AV16] Yossi Azar and Adi Vardi. *Dynamic Traveling Repair Problem with an Arbitrary Time Window*. In: *WAOA*. Vol. 10138. Lecture Notes in Computer Science. Springer, 2016, pp. 14–26.

[BC23] Xingjian Bai and Christian Coester. *Sorting with Predictions*. In: *NeurIPS*. 2023.

[Bal+23a] Júlia Baligács, Yann Disser, Farehe Soheil, and David Weckbecker. *Tight Analysis of the Lazy Algorithm for Open Online Dial-a-Ride*. In: *WADS*. Vol. 14079. Lecture Notes in Computer Science. Springer, 2023, pp. 43–64.

[BGT23] Eric Balkanski, Vasilis Gkatzelis, and Xizhi Tan. *Strategyproof Scheduling with Predictions*. In: *ITCS*. Vol. 251. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, 11:1–11:22.

[Bal+23b] Eric Balkanski, Tingting Ou, Clifford Stein, and Hao-Ting Wei. *Scheduling with Speed Predictions*. In: *WAOA*. Vol. 14297. Lecture Notes in Computer Science. Springer, 2023, pp. 74–89.

[Bal+23c] Eric Balkanski, Noémie Périvier, Clifford Stein, and Hao-Ting Wei. *Energy-Efficient Scheduling with Predictions*. In: *NeurIPS*. 2023.

[BDM22] Étienne Bamas, Marina Drygala, and Andreas Maggiori. *An Improved Analysis of Greedy for Online Steiner Forest*. In: *SODA*. SIAM, 2022, pp. 3202–3229.

[Bam+24] Étienne Bamas, Alexander Lindermayr, Nicole Megow, Lars Rohwedder, and Jens Schlöter. *Santa Claus meets Makespan and Matroids: Algorithms and Reductions*. In: *SODA*. SIAM, 2024, pp. 2829–2860.

[Bam+20] Étienne Bamas, Andreas Maggiori, Lars Rohwedder, and Ola Svensson. *Learning Augmented Energy Minimization via Speed Scaling*. In: *NeurIPS*. 2020.

[BMS20] Étienne Bamas, Andreas Maggiori, and Ola Svensson. *The Primal-Dual method for Learning Augmented Algorithms*. In: *NeurIPS*. 2020.

[BR23] Étienne Bamas and Lars Rohwedder. *Better Trees for Santa Claus*. In: *STOC*. ACM, 2023, pp. 1862–1875.

[Bam+22] Evripidis Bampis, Konstantinos Dogeas, Alexander V. Kononov, Giorgio Lucarelli, and Fanny Pascual. *Scheduling with Untrusted Predictions*. In: *IJCAI*. ijcai.org, 2022, pp. 4581–4587.

[Bam+23a] Evripidis Bampis, Bruno Escoffier, Themis Gouleakis, Niklas Hahn, Kostas Lakis, Golnoosh Shahkarami, and Michalis Xefteris. *Learning-Augmented Online TSP on Rings, Trees, Flowers and (Almost) Everywhere Else*. In: *ESA*. Vol. 274. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, 12:1–12:17.

[Bam+23b] Evripidis Bampis, Bruno Escoffier, Niklas Hahn, and Michalis Xefteris. *Online TSP with Known Locations*. In: *WADS*. Vol. 14079. Lecture Notes in Computer Science. Springer, 2023, pp. 65–78.

[Bam+23c] Evripidis Bampis, Alexander V. Kononov, Giorgio Lucarelli, and Fanny Pascual. *Non-Clairvoyant Makespan Minimization Scheduling with Predictions*. In: *ISAAC*. Vol. 283. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, 9:1–9:15.

[Ban17] Nikhil Bansal. *Scheduling: Open problems old and new*. Presentation at MAPSP. 2017.

[BCS08] Nikhil Bansal, Don Coppersmith, and Maxim Sviridenko. *Improved Approximation Algorithms for Broadcast Scheduling*. In: *SIAM J. Comput.* 38.3 (2008), pp. 1157–1174.

[BS06] Nikhil Bansal and Maxim Sviridenko. *The Santa Claus problem*. In: *STOC*. ACM, 2006, pp. 31–40.

[Bap+07] Philippe Baptiste, Peter Brucker, Marek Chrobak, Christoph Dürr, Svetlana A. Kravchenko, and Francis Sourd. *The complexity of mean flow time scheduling problems with release times*. In: *J. Sched.* 10.2 (2007), pp. 139–146.

[Bar00] Reuven Bar-Yehuda. *One for the Price of Two: a Unified Approach for Approximating Covering Problems*. In: *Algorithmica* 27.2 (2000), pp. 131–144.

[BE85] Reuven Bar-Yehuda and Shimon Even. *A local-ratio theorem for approximating the weighted vertex cover problem*. In: *North-Holland Mathematics Studies*. Vol. 109. Elsevier, 1985, pp. 27–45.

[BR01] Reuven Bar-Yehuda and Dror Rawitz. *On the Equivalence between the Primal-Dual Schema and the Local-Ratio Technique*. In: *RANDOM-APPROX*. Vol. 2129. Lecture Notes in Computer Science. Springer, 2001, pp. 24–35.

[BCG09] MohammadHossein Bateni, Moses Charikar, and Venkatesan Guruswami. *MaxMin allocation via degree lower-bounded arborescences*. In: *STOC*. ACM, 2009, pp. 543–552.

[Bea+12] Olivier Beaumont, Nicolas Bonichon, Lionel Eyraud-Dubois, and Loris Marchal. *Minimizing Weighted Mean Completion Time for Malleable Tasks Scheduling*. In: *IPDPS*. IEEE Computer Society, 2012, pp. 273–284.

[Bel+15] Odile Bellenguez-Morineau, Marek Chrobak, Christoph Dürr, and Damien Prot. *A note on NP-hardness of preemptive mean flow-time scheduling for parallel machines.* In: *J. Sched.* 18.3 (2015), pp. 299–304.

[BP23] Ziyad Benomar and Vianney Perchet. *Advice Querying under Budget Constraint for Online Algorithms.* In: *NeurIPS*. 2023.

[BP24] Ziyad Benomar and Vianney Perchet. *Non-clairvoyant Scheduling with Partial Predictions.* In: *ICML*. OpenReview.net, 2024.

[BC97] Piotr Berman and Chris Coulston. *On-Line Algorithms for Steiner Tree Problems (Extended Abstract).* In: *STOC*. ACM, 1997, pp. 344–353.

[Ber+22a] Giulia Bernardini, Alexander Lindermayr, Alberto Marchetti-Spaccamela, Nicole Megow, Leen Stougie, and Michelle Sweering. *A Universal Error Measure for Input Predictions Applied to Online Graph Problems.* In: *NeurIPS*. 2022.

[Ber+22b] Giulia Bernardini, Alexander Lindermayr, Alberto Marchetti-Spaccamela, Nicole Megow, Leen Stougie, and Michelle Sweering. *A Universal Error Measure for Input Predictions Applied to Online Graph Problems.* In: *CoRR* abs/2205.12850 (2022).

[BT97] Dimitris Bertsimas and John N. Tsitsiklis. *Introduction to linear optimization.* Vol. 6. Athena scientific optimization and computation series. Athena Scientific, 1997.

[BD05] Ivona Bezáková and Varsha Dani. *Allocating indivisible goods.* In: *SIGecom Exch.* 5.3 (2005), pp. 11–18.

[Bha+14] Sayan Bhattacharya, Sungjin Im, Janardhan Kulkarni, and Kamesh Munagala. *Coordination mechanisms from (almost) all scheduling policies.* In: *ITCS*. ACM, 2014, pp. 121–134.

[BKL21] Marcin Bienkowski, Artur Kraska, and Hsiang-Hsuan Liu. *Traveling Repairperson, Unrelated Machines, and Other Stories About Average Completion Times.* In: *ICALP*. Vol. 198. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 28:1–28:20.

[BL19] Marcin Bienkowski and Hsiang-Hsuan Liu. *An Improved Online Algorithm for the Traveling Repairperson Problem on a Line.* In: *MFCS*. Vol. 138. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 6:1–6:12.

[BD20] Alexander Birx and Yann Disser. *Tight Analysis of the Smartstart Algorithm for Online Dial-a-Ride on the Line.* In: *SIAM J. Discret. Math.* 34.2 (2020), pp. 1409–1443.

[BDS23] Alexander Birx, Yann Disser, and Kevin Schewior. *Improved Bounds for Open Online Dial-a-Ride on the Line.* In: *Algorithmica* 85.5 (2023), pp. 1372–1414.

[Bje+21] Antje Bjelde, Jan Hackfeld, Yann Disser, Christoph Hansknecht, Maarten Lipmann, Julie Meißner, Miriam Schlöter, Kevin Schewior, and Leen Stougie. *Tight Bounds for Online TSP on the Line.* In: *ACM Trans. Algorithms* 17.1 (2021), 3:1–3:58.

[Blo+01] Michiel Blom, Sven Oliver Krumke, Willem de Paepe, and Leen Stougie. *The Online TSP Against Fair Adversaries.* In: *INFORMS J. Comput.* 13.2 (2001), pp. 138–148.

[BS09] Vincenzo Bonifaci and Leen Stougie. *Online* k*-Server Routing Problems.* In: *Theory Comput. Syst.* 45.3 (2009), pp. 470–485.

[BE98] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.

[Boy+23] Joan Boyar, Lene M. Favrholdt, Shahin Kamali, and Kim S. Larsen. *Online Interval Scheduling with Predictions*. In: *WADS*. Vol. 14079. Lecture Notes in Computer Science. Springer, 2023, pp. 193–207.

[Boy+17] Joan Boyar, Lene M. Favrholdt, Christian Kudahl, Kim S. Larsen, and Jesper W. Mikkelsen. *Online Algorithms with Advice: A Survey*. In: *ACM Comput. Surv.* 50.2 (2017), 19:1–19:34.

[BV14] Stephen P. Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2014.

[Bra+24] Jan van den Brand, Sebastian Forster, Yasamin Nazari, and Adam Polak. *On Dynamic Graph Algorithms with Predictions*. In: *SODA*. SIAM, 2024, pp. 3534–3557.

[BK04] Peter Brucker and Svetlana A Kravchenko. *Complexity of mean flow time scheduling problems with release dates*. Universität Osnabrück. Fachbereich Mathematik/Informatik, 2004.

[BJS74] John L. Bruno, Edward G. Coffman Jr., and Ravi Sethi. *Scheduling Independent Tasks to Reduce Mean Finishing Time*. In: *Commun. ACM* 17.7 (1974), pp. 382–387.

[BJN07] Niv Buchbinder, Kamal Jain, and Joseph Naor. *Online Primal-Dual Algorithms for Maximizing Ad-Auctions Revenue*. In: *ESA*. Vol. 4698. Lecture Notes in Computer Science. Springer, 2007, pp. 253–264.

[BN09] Niv Buchbinder and Joseph Naor. *The Design of Competitive Online Algorithms via a Primal-Dual Approach*. In: *Found. Trends Theor. Comput. Sci.* 3.2-3 (2009), pp. 93–263.

[Cha+09] Jivitej S. Chadha, Naveen Garg, Amit Kumar, and V. N. Muralidhara. *A competitive algorithm for minimizing weighted flow time on unrelatedmachines with speed augmentation*. In: *STOC*. ACM, 2009, pp. 679–684.

[Cha+96] Soumen Chakrabarti, Cynthia A. Phillips, Andreas S. Schulz, David B. Shmoys, Clifford Stein, and Joel Wein. *Improved Scheduling Algorithms for Minsum Criteria*. In: *ICALP*. Vol. 1099. Lecture Notes in Computer Science. Springer, 1996, pp. 646–657.

[CCK09] Deeparnab Chakrabarty, Julia Chuzhoy, and Sanjeev Khanna. *On Allocating Goods to Maximize Fairness*. In: *FOCS*. IEEE Computer Society, 2009, pp. 107–116.

[CKL15] Deeparnab Chakrabarty, Sanjeev Khanna, and Shi Li. *On (1,ε)-Restricted Assignment Makespan Minimization*. In: *SODA*. SIAM, 2015, pp. 1087–1101.

[CTW18] T.-H. Hubert Chan, Zhihao Gavin Tang, and Xiaowei Wu. *On (1,ε)-Restricted Max-Min Fair Allocation Problem*. In: *Algorithmica* 80.7 (2018), pp. 2181–2200.

[CC24] Shuchi Chawla and Dimitris Christou. *Online Time-Windows TSP with Predictions*. In: *APPROX/RANDOM*. Vol. 317. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024, 2:1–2:21.

[Che+04] Chandra Chekuri, Ashish Goel, Sanjeev Khanna, and Amit Kumar. *Multi-processor scheduling to minimize flow time with epsilon resource augmentation.* In: *STOC*. ACM, 2004, pp. 363–372.

[CK01] Chandra Chekuri and Sanjeev Khanna. *A PTAS for Minimizing Weighted Completion Time on Uniformly Related Machines.* In: *ICALP*. Vol. 2076. Lecture Notes in Computer Science. Springer, 2001, pp. 848–861.

[Che+22] Justin Y. Chen, Sandeep Silwal, Ali Vakilian, and Fred Zhang. *Faster Fundamental Graph Algorithms via Learned Predictions.* In: *ICML*. Vol. 162. Proceedings of Machine Learning Research. PMLR, 2022, pp. 3583–3602.

[Che+16] Lin Chen, Klaus Jansen, Wenchang Luo, and Guochuan Zhang. *An Efficient PTAS for Parallel Machine Scheduling with Capacity Constraints.* In: *COCOA*. Vol. 10043. Lecture Notes in Computer Science. Springer, 2016, pp. 608–623.

[CM18a] Siu-Wing Cheng and Yuchen Mao. *Integrality Gap of the Configuration LP for the Restricted Max-Min Fair Allocation.* In: *CoRR* abs/1807.04152 (2018).

[CM18b] Siu-Wing Cheng and Yuchen Mao. *Restricted Max-Min Fair Allocation.* In: *ICALP*. Vol. 107. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 37:1–37:13.

[CM19] Siu-Wing Cheng and Yuchen Mao. *Restricted Max-Min Allocation: Approximation and Integrality Gap.* In: *ICALP*. Vol. 132. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 38:1–38:13.

[Cho+18] Anamitra Roy Choudhury, Syamantak Das, Naveen Garg, and Amit Kumar. *Rejecting jobs to minimize load and maximum flow-time.* In: *J. Comput. Syst. Sci.* 91 (2018), pp. 42–68.

[Chr76] Nicos Christofides. *Worst-case analysis of a new heuristic for the travelling salesman problem.* Technical Report 388. Graduate School of Industrial Administration, Carnegie Mellon University, 1976.

[Chr22] Nicos Christofides. *Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem.* In: *Oper. Res. Forum* 3.1 (2022).

[Chv79] Vasek Chvátal. *A Greedy Heuristic for the Set-Covering Problem.* In: *Math. Oper. Res.* 4.3 (1979), pp. 233–235.

[CP23] Ilan Reuven Cohen and Debmalya Panigrahi. *A General Framework for Learning-Augmented Online Allocation.* In: *ICALP*. Vol. 261. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, 43:1–43:21.

[Coh+24] Vincent Cohen-Addad, Tommaso d'Orsi, Anupam Gupta, Euiwoong Lee, and Debmalya Panigrahi. *Max-Cut with ϵ-Accurate Predictions.* In: *NeurIPS*. 2024.

[CQ12] José R Correa and Maurice Queyranne. *Efficiency of equilibria in restricted uniform machine scheduling with total weighted completion time as social cost.* In: *Naval Research Logistics (NRL)* 59.5 (2012), pp. 384–395.

[CS05] José R. Correa and Andreas S. Schulz. *Single-Machine Scheduling with Precedence Constraints.* In: *Math. Oper. Res.* 30.4 (2005), pp. 1005–1021.

[CW09] José R. Correa and Michael R. Wagner. *LP-based online scheduling: from single to parallel machines*. In: *Math. Program.* 119.1 (2009), pp. 109–136.

[DFF56] George Bernard Dantzig, Lester Randolph Ford, and Delbert Ray Fulkerson. *A primal-dual algorithm*. California Rand Corporation, 1956.

[DW17] Syamantak Das and Andreas Wiese. *On Minimizing the Makespan When Some Jobs Cannot Be Assigned on the Same Machine*. In: *ESA*. Vol. 87. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 31:1–31:14.

[Dav+23] Sami Davies, Benjamin Moseley, Sergei Vassilvitskii, and Yuyan Wang. *Predictive Flows for Faster Ford-Fulkerson*. In: *ICML*. Vol. 202. Proceedings of Machine Learning Research. PMLR, 2023, pp. 7231–7248.

[DRZ20] Sami Davies, Thomas Rothvoss, and Yihao Zhang. *A Tale of Santa Claus, Hypergraphs and Matroids*. In: *SODA*. SIAM, 2020, pp. 2748–2757.

[Den+00] Xiaotie Deng, Nian Gu, Tim Brecht, and KaiCheng Lu. *Preemptive Scheduling of Parallel Jobs on Multiprocessors*. In: *SIAM J. Comput.* 30.1 (2000), pp. 145–160.

[DJK13] Nikhil R. Devanur, Kamal Jain, and Robert D. Kleinberg. *Randomized Primal-Dual analysis of RANKING for Online BiPartite Matching*. In: *SODA*. SIAM, 2013, pp. 101–107.

[Die+09] Florian Diedrich, Klaus Jansen, Ulrich M. Schwarz, and Denis Trystram. *A Survey on Approximation Algorithms for Scheduling with Machine Unavailability*. In: *Algorithmics of Large and Complex Networks*. Vol. 5515. Lecture Notes in Computer Science. Springer, 2009, pp. 50–64.

[Din+21] Michael Dinitz, Sungjin Im, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. *Faster Matchings via Learned Duals*. In: *NeurIPS*. 2021, pp. 10393–10406.

[Din+22] Michael Dinitz, Sungjin Im, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. *Algorithms with Prediction Portfolios*. In: *NeurIPS*. 2022.

[DNS23] Marina Drygala, Sai Ganesh Nagarajan, and Ola Svensson. *Online Algorithms with Costly Predictions*. In: *AISTATS*. Vol. 206. Proceedings of Machine Learning Research. PMLR, 2023, pp. 8078–8101.

[DLY91] Jianzhong Du, Joseph Y.-T. Leung, and Gilbert H. Young. *Scheduling Chain-Structured Tasks to Minimize Makespan and Mean Flow Time*. In: *Inf. Comput.* 92.2 (1991), pp. 219–236.

[Dür+20] Christoph Dürr, Thomas Erlebach, Nicole Megow, and Julie Meißner. *An Adversarial Model for Scheduling with Testing*. In: *Algorithmica* 82.12 (2020), pp. 3630–3675.

[DW90] Martin E. Dyer and Laurence A. Wolsey. *Formulating the single machine sequencing problem with release dates as a mixed integer program*. In: *Discret. Appl. Math.* 26.2-3 (1990), pp. 255–270.

[EEI64] Willard L Eastman, Shimon Even, and I Martin Isaacs. *Bounds for the optimal scheduling of n jobs on m processors*. In: *Management science* 11.2 (1964), pp. 268–279.

[EKS14] Tomás Ebenlendr, Marek Krcál, and Jirí Sgall. *Graph Balancing: A Special Case of Scheduling Unrelated Parallel Machines*. In: *Algorithmica* 68.1 (2014), pp. 62–80.

[Ebe+21] Franziska Eberle, Ruben Hoeksma, Nicole Megow, Lukas Nölke, Kevin Schewior, and Bertrand Simon. *Speed-Robust Scheduling - Sand, Bricks, and Rocks*. In: *IPCO*. Vol. 12707. Lecture Notes in Computer Science. Springer, 2021, pp. 283–296.

[Ebe+22] Franziska Eberle, Alexander Lindermayr, Nicole Megow, Lukas Nölke, and Jens Schlöter. *Robustification of Online Graph Exploration Methods*. In: *AAAI*. AAAI Press, 2022, pp. 9732–9740.

[Edm65] Jack Edmonds. *Paths, trees, and flowers*. In: *Canadian Journal of mathematics* 17 (1965), pp. 449–467.

[EP12] Jeff Edmonds and Kirk Pruhs. *Scalably scheduling processes with arbitrary speedup curves*. In: *ACM Trans. Algorithms* 8.3 (2012), 28:1–28:10.

[EG59] Edmund Eisenberg and David Gale. *Consensus of Subjective Probabilities: The Pari-Mutuel Method*. In: *Ann. Math. Statist.* 30.1 (1959), pp. 165–168.

[Eli+24] Marek Eliás, Haim Kaplan, Yishay Mansour, and Shay Moran. *Learning-Augmented Algorithms with Explicit Predictors*. In: *NeurIPS*. 2024.

[Eps+17] Leah Epstein, Asaf Levin, Alan J. Soper, and Vitaly A. Strusevich. *Power of Preemption for Minimizing Total Completion Time on Uniform Parallel Machines*. In: *SIAM J. Discret. Math.* 31.1 (2017), pp. 101–123.

[Erl+22] Thomas Erlebach, Murilo Santos de Lima, Nicole Megow, and Jens Schlöter. *Learning-Augmented Query Policies for Minimum Spanning Tree with Uncertainty*. In: *ESA*. Vol. 244. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 49:1–49:18.

[Fei08] Uriel Feige. *On allocations that maximize fairness*. In: *SODA*. SIAM, 2008, pp. 287–293.

[FR98] Dror G. Feitelson and Larry Rudolph. *Metrics and Benchmarking for Parallel Job Scheduling*. In: *JSSPP*. Vol. 1459. Lecture Notes in Computer Science. Springer, 1998, pp. 1–24.

[Fel+08] Jon Feldman, S. Muthukrishnan, Evdokia Nikolova, and Martin Pál. *A Truthful Mechanism for Offline Ad Slot Scheduling*. In: *SAGT*. Vol. 4997. Lecture Notes in Computer Science. Springer, 2008, pp. 182–193.

[FN21] Yiding Feng and Rad Niazadeh. *Batching and Optimal Multi-Stage Bipartite Allocations (Extended Abstract)*. In: *ITCS*. Vol. 185. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 88:1–88:1.

[FNS21] Yiding Feng, Rad Niazadeh, and Amin Saberi. *Two-stage Stochastic Matching with Application to Ride Hailing*. In: *SODA*. SIAM, 2021, pp. 2862–2877.

[FS01] Esteban Feuerstein and Leen Stougie. *On-line single-server dial-a-ride problems*. In: *Theor. Comput. Sci.* 268.1 (2001), pp. 91–105.

[FR03] Ari Freund and Dror Rawitz. *Combinatorial Interpretations of Dual Fitting and Primal Fitting*. In: *WAOA*. Vol. 2909. Lecture Notes in Computer Science. Springer, 2003, pp. 137–150.

[GMW07] Martin Gairing, Burkhard Monien, and Andreas Woclaw. *A faster combinatorial approximation algorithm for scheduling unrelated parallel machines*. In: *Theor. Comput. Sci.* 380.1-2 (2007), pp. 87–99.

[GJ79] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[GTV22] Jugal Garg, Thorben Tröbst, and Vijay V. Vazirani. *One-Sided Matching Markets with Endowments: Equilibria and Algorithms*. In: *AAMAS*. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS), 2022, pp. 463–471.

[Gar+19] Naveen Garg, Anupam Gupta, Amit Kumar, and Sahil Singla. *Non-Clairvoyant Precedence Constrained Scheduling*. In: *ICALP*. Vol. 132. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 63:1–63:14.

[Gho+11] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. *Dominant Resource Fairness: Fair Allocation of Multiple Resource Types*. In: *NSDI*. USENIX Association, 2011.

[Gka+22] Vasilis Gkatzelis, Kostas Kollias, Alkmini Sgouritsa, and Xizhi Tan. *Improved Price of Anarchy via Predictions*. In: *EC*. ACM, 2022, pp. 529–557.

[Goe96] Michel X. Goemans. *A Supermodular Relaxation for Scheduling with Release Dates*. In: *IPCO*. Vol. 1084. Lecture Notes in Computer Science. Springer, 1996, pp. 288–300.

[GW95] Michel X. Goemans and David P. Williamson. *A General Approximation Technique for Constrained Forest Problems*. In: *SIAM J. Comput.* 24.2 (1995), pp. 296–317.

[Gon77] Teofilo Gonzalez. *Optimal mean finish time preemptive schedules*. In: *Technical Report 220*. Computer Science Department, Pennsylvania State University Chichester, 1977.

[GS78] Teofilo F. Gonzalez and Sartaj Sahni. *Preemptive Scheduling of Uniform Processor Systems*. In: *J. ACM* 25.1 (1978), pp. 92–101.

[GLS22] Themis Gouleakis, Konstantinos Lakis, and Golnoosh Shahkarami. *Learning-Augmented Algorithms for Online TSP on the Line*. In: *CoRR* abs/2206.00655 (2022).

[GLS23] Themis Gouleakis, Konstantinos Lakis, and Golnoosh Shahkarami. *Learning-Augmented Algorithms for Online TSP on the Line*. In: *AAAI*. AAAI Press, 2023, pp. 11989–11996.

[GJK19] Kilian Grage, Klaus Jansen, and Kim-Manuel Klein. *An EPTAS for Machine Scheduling with Bag-Constraints*. In: *SPAA*. ACM, 2019, pp. 135–144.

[Gra66] Ronald L. Graham. *Bounds for certain multiprocessing anomalies*. In: *Bell system technical journal* 45.9 (1966), pp. 1563–1581.

[Gra69] Ronald L. Graham. *Bounds on Multiprocessing Timing Anomalies*. In: *SIAM Journal of Applied Mathematics* 17.2 (1969), pp. 416–429.

[Gra+79] Ronald Lewis Graham, Eugene Leighton Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. *Optimization and approximation in deterministic sequencing and scheduling: a survey*. In: *Annals of discrete mathematics*. Vol. 5. Elsevier, 1979, pp. 287–326.

[GLS88] Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Vol. 2. Algorithms and Combinatorics. Springer, 1988.

[Gup+12] Anupam Gupta, Sungjin Im, Ravishankar Krishnaswamy, Benjamin Moseley, and Kirk Pruhs. *Scheduling heterogeneous processors isn't as easy as you think*. In: *SODA*. SIAM, 2012, pp. 1242–1253.

[GKP10] Anupam Gupta, Ravishankar Krishnaswamy, and Kirk Pruhs. *Scalably Scheduling Power-Heterogeneous Processors*. In: *ICALP (1)*. Vol. 6198. Lecture Notes in Computer Science. Springer, 2010, pp. 312–323.

[GKP12] Anupam Gupta, Ravishankar Krishnaswamy, and Kirk Pruhs. *Online Primal-Dual for Non-linear Optimization with Applications to Speed Scaling*. In: *WAOA*. Vol. 7846. Lecture Notes in Computer Science. Springer, 2012, pp. 173–186.

[GKL18] Anupam Gupta, Amit Kumar, and Jason Li. *Non-Preemptive Flow-Time Minimization via Rejections*. In: *ICALP*. Vol. 107. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 70:1–70:13.

[GKS21] Anupam Gupta, Amit Kumar, and Sahil Singla. *Bag-Of-Tasks Scheduling on Related Machines*. In: *APPROX-RANDOM*. Vol. 207. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 3:1–3:16.

[Gup+18] Ujjwal Gupta, Manoj Babu, Raid Ayoub, Michael Kishinevsky, Francesco Paterna, and Ümit Y. Ogras. *STAFF: online learning with stabilized adaptive forgetting factor and feature selection algorithm*. In: *DAC*. ACM, 2018, 177:1–177:6.

[Gup+21] Varun Gupta, Benjamin Moseley, Marc Uetz, and Qiaomin Xie. *Corrigendum: Greed Works - Online Algorithms for Unrelated Machine Stochastic Scheduling*. In: *Math. Oper. Res.* 46.3 (2021), pp. 1230–1234.

[HSS11] Bernhard Haeupler, Barna Saha, and Aravind Srinivasan. *New Constructive Aspects of the Lovász Local Lemma*. In: *J. ACM* 58.6 (2011), 28:1–28:28.

[HJ06] Mohammad Taghi Hajiaghayi and Kamal Jain. *The prize-collecting generalized steiner tree problem via a new approach of primal-dual schema*. In: *SODA*. ACM Press, 2006, pp. 631–640.

[Hal+97] Leslie A. Hall, Andreas S. Schulz, David B. Shmoys, and Joel Wein. *Scheduling to Minimize Average Completion Time: Off-Line and On-Line Approximation Algorithms*. In: *Math. Oper. Res.* 22.3 (1997), pp. 513–544.

[Har24] David G. Harris. *Dependent rounding with strong negative-correlation, and scheduling on unrelated machines to minimize completion time*. In: *SODA*. SIAM, 2024, pp. 2275–2304.

[HS23] Penny Haxell and Tibor Szabó. *Improved Integrality Gap in Max-Min Allocation: or Topology at the North Pole*. In: *SODA*. SIAM, 2023, pp. 2875–2897.

[Hax95] Penny E. Haxell. *A condition for matchability in hypergraphs*. In: *Graphs Comb.* 11.3 (1995), pp. 245–248.

[Hay08] Brian Hayes. *Cloud computing*. In: *Commun. ACM* 51.7 (2008), pp. 9–11.

[Hen+24] Monika Henzinger, Barna Saha, Martin P. Seybold, and Christopher Ye. *On the Complexity of Algorithms with Predictions for Dynamic Graph Problems*. In: *ITCS*. Vol. 287. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024, 62:1–62:25.

[HS87] Dorit S. Hochbaum and David B. Shmoys. *Using dual approximation algorithms for scheduling problems theoretical and practical results*. In: *J. ACM* 34.1 (1987), pp. 144–162.

[HV96] Han Hoogeveen and Arjen P. A. Vestjens. *Optimal On-Line Algorithms for Single-Machine Scheduling*. In: *IPCO*. Vol. 1084. Lecture Notes in Computer Science. Springer, 1996, pp. 404–414.

[Hor73] W. A. Horn. *Technical Note - Minimizing Average Flow Time with Parallel Machines*. In: *Oper. Res.* 21.3 (1973), pp. 846–847.

[HS76] Ellis Horowitz and Sartaj Sahni. *Exact and Approximate Algorithms for Scheduling Nonidentical Processors*. In: *J. ACM* 23.2 (1976), pp. 317–327.

[HLS77] Edward C. Horvath, Shui Lam, and Ravi Sethi. *A Level Algorithm for Preemptive Scheduling*. In: *J. ACM* 24.1 (1977), pp. 32–43.

[Hu+22] Hsiao-Yu Hu, Hao-Ting Wei, Meng-Hsi Li, Kai-Min Chung, and Chung-Shou Liao. *Online TSP with Predictions*. In: *CoRR* abs/2206.15364 (2022).

[HJ18] Dawsen Hwang and Patrick Jaillet. *Online scheduling with multi-state machines*. In: *Networks* 71.3 (2018), pp. 209–251.

[HZ79] Aanund Hylland and Richard Zeckhauser. *The Efficient Allocation of Individuals to Positions*. In: *Journal of Political Economy* 87.2 (1979), pp. 293–314.

[IK16] Sungjin Im and Janardhan Kulkarni. *Fair Online Scheduling for Selfish Jobs on Heterogeneous Machines*. In: *SPAA*. ACM, 2016, pp. 185–194.

[IKM15a] Sungjin Im, Janardhan Kulkarni, and Benjamin Moseley. *Temporal Fairness of Round Robin: Competitive Analysis for Lk-norms of Flow Time*. In: *SPAA*. ACM, 2015, pp. 155–160.

[IKM14] Sungjin Im, Janardhan Kulkarni, and Kamesh Munagala. *Competitive algorithms from competitive equilibria: non-clairvoyant scheduling under polyhedral constraints*. In: *STOC*. ACM, 2014, pp. 313–322.

[IKM15b] Sungjin Im, Janardhan Kulkarni, and Kamesh Munagala. *Competitive Flow Time Algorithms for Polyhedral Scheduling*. In: *FOCS*. IEEE Computer Society, 2015, pp. 506–524.

[IKM18] Sungjin Im, Janardhan Kulkarni, and Kamesh Munagala. *Competitive Algorithms from Competitive Equilibria: Non-Clairvoyant Scheduling under Polyhedral Constraints*. In: *J. ACM* 65.1 (2018), 3:1–3:33.

[IKM] Sungjin Im, Janardhan Kulkarni, and Kamesh Munagala. Erratum. URL: https://faculty.ucmerced.edu/sim3/papers/jacm-psp-errata.txt.

[Im+14] Sungjin Im, Janardhan Kulkarni, Kamesh Munagala, and Kirk Pruhs. *Selfish-Migrate: A Scalable Algorithm for Non-clairvoyantly Scheduling Heterogeneous Processors*. In: *FOCS*. IEEE Computer Society, 2014, pp. 531–540.

[Im+23] Sungjin Im, Ravi Kumar, Mahshid Montazer Qaem, and Manish Purohit. *Non-clairvoyant Scheduling with Predictions*. In: *ACM Trans. Parallel Comput.* 10.4 (2023), 19:1–19:26.

[IL16] Sungjin Im and Shi Li. *Better Unrelated Machine Scheduling for Weighted Completion Time via Random Offsets from Non-uniform Distributions*. In: *FOCS*. IEEE Computer Society, 2016, pp. 138–147.

[IL23] Sungjin Im and Shi Li. *Improved Approximations for Unrelated Machine Scheduling*. In: *SODA*. SIAM, 2023, pp. 2917–2946.

[IM11] Sungjin Im and Benjamin Moseley. *Online Scalable Algorithm for Minimizing ℓ_k-norms of Weighted Flow Time On Unrelated Machines*. In: *SODA*. SIAM, 2011, pp. 95–108.

[IM15] Sungjin Im and Benjamin Moseley. *Scheduling in Bandwidth Constrained Tree Networks*. In: *SPAA*. ACM, 2015, pp. 171–180.

[IW91] Makoto Imase and Bernard M. Waxman. *Dynamic Steiner Tree Problem*. In: *SIAM J. Discret. Math.* 4.3 (1991), pp. 369–384.

[Jäg23] Sven Jäger. *An improved greedy algorithm for stochastic online scheduling on unrelated machines*. In: *Discret. Optim.* 47 (2023), p. 100753.

[JLM25] Sven Jäger, Alexander Lindermayr, and Nicole Megow. *The Power of Proportional Fairness for Non-Clairvoyant Scheduling under Polyhedral Constraints*. In: *SODA*. To appear. SIAM, 2025.

[Jäg+23] Sven Jäger, Guillaume Sagnol, Daniel Schmidt genannt Waldschmidt, and Philipp Warode. *Competitive Kill-and-Restart and Preemptive Strategies for Non-clairvoyant Scheduling*. In: *IPCO*. Vol. 13904. Lecture Notes in Computer Science. Springer, 2023, pp. 246–260.

[JW24] Sven Jäger and Philipp Warode. *Simple Approximation Algorithms for Minimizing the Total Weighted Completion Time of Precedence-Constrained Jobs*. In: *SOSA*. SIAM, 2024, pp. 82–96.

[Jäg21] Sven Joachim Jäger. *Approximation in deterministic and stochastic machine scheduling*. PhD thesis. Technical University of Berlin, Germany, 2021.

[JW08] Patrick Jaillet and Michael R. Wagner. *Generalized Online Routing: New Competitive Ratios, Resource Augmentation, and Asymptotic Analyses*. In: *Oper. Res.* 56.3 (2008), pp. 745–757.

[Jai+03] Kamal Jain, Mohammad Mahdian, Evangelos Markakis, Amin Saberi, and Vijay V. Vazirani. *Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP*. In: *J. ACM* 50.6 (2003), pp. 795–824.

[JV10] Kamal Jain and Vijay V. Vazirani. *Eisenberg-Gale markets: Algorithms and game-theoretic properties*. In: *Games Econ. Behav.* 70.1 (2010), pp. 84–106.

[Jai+15] Navendu Jain, Ishai Menache, Joseph Naor, and Jonathan Yaniv. *Near-Optimal Scheduling Mechanisms for Deadline-Sensitive Jobs in Large Computing Clusters*. In: *ACM Trans. Parallel Comput.* 2.1 (2015), 3:1–3:29.

[Jal+23] Devansh Jalota, Marco Pavone, Qi Qi, and Yinyu Ye. *Fisher markets with linear constraints: Equilibrium properties and efficient distributed algorithms*. In: *Games Econ. Behav.* 141 (2023), pp. 223–260.

[JS23] Samin Jamalabadi and Uwe Schwiegelshohn. *WSRPT is 1.2259-competitive for Weighted Completion Time Scheduling*. In: *CoRR* abs/2307.07739 (2023).

[JLM18] Klaus Jansen, Kati Land, and Marten Maack. *Estimating the Makespan of the Two-Valued Restricted Assignment Problem*. In: *Algorithmica* 80.4 (2018), pp. 1357–1382.

[JR19] Klaus Jansen and Lars Rohwedder. *Local Search Breaks 1.75 for Graph Balancing*. In: *ICALP*. Vol. 132. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 74:1–74:14.

[JR20a] Klaus Jansen and Lars Rohwedder. *A note on the integrality gap of the configuration LP for restricted Santa Claus*. In: *Inf. Process. Lett.* 164 (2020), p. 106025.

[JR20b] Klaus Jansen and Lars Rohwedder. *A Quasi-Polynomial Approximation for the Restricted Assignment Problem*. In: *SIAM J. Comput.* 49.6 (2020), pp. 1083–1108.

[JM22] Billy Jin and Will Ma. *Online Bipartite Matching with Advice: Tight Robustness-Consistency Tradeoffs for the Two-Stage Model*. In: *NeurIPS*. 2022.

[KP00] Bala Kalyanasundaram and Kirk Pruhs. *Speed is as powerful as clairvoyance*. In: *J. ACM* 47.4 (2000), pp. 617–643.

[KN79] Mamoru Kaneko and Kenjiro Nakamura. *The Nash Social Welfare Function*. In: *Econometrica* 47.2 (1979), pp. 423–435.

[KKG21] Anna R. Karlin, Nathan Klein, and Shayan Oveis Gharan. *A (slightly) improved approximation algorithm for metric TSP*. In: *STOC*. ACM, 2021, pp. 32–45.

[KKG23] Anna R. Karlin, Nathan Klein, and Shayan Oveis Gharan. *A Deterministic Better-than-3/2 Approximation Algorithm for Metric TSP*. In: *IPCO*. Vol. 13904. Lecture Notes in Computer Science. Springer, 2023, pp. 261–274.

[KLS15] Marek Karpinski, Michael Lampis, and Richard Schmied. *New inapproximability bounds for TSP*. In: *J. Comput. Syst. Sci.* 81.8 (2015), pp. 1665–1677.

[Kaw+21] Yasushi Kawase, Kei Kimura, Kazuhisa Makino, and Hanna Sumita. *Optimal Matroid Partitioning Problems*. In: *Algorithmica* 83.6 (2021), pp. 1653–1676.

[Khd+15] Heba Khdr, Santiago Pagani, Muhammad Shafique, and Jörg Henkel. *Thermal constrained resource management for mixed ILP-TLP workloads in dark silicon chips*. In: *DAC*. ACM, 2015, 179:1–179:6.

[Kho+22] Misha Khodak, Maria-Florina Balcan, Ameet Talwalkar, and Sergei Vassilvitskii. *Learning Predictions for Algorithms with Predictions*. In: *NeurIPS*. 2022.

[KC03] Jae-Hoon Kim and Kyung-Yong Chwa. *Non-clairvoyant scheduling for weighted flow time*. In: *Inf. Process. Lett.* 87.1 (2003), pp. 31–37.

[KV18] Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer Berlin Heidelberg, 2018.

[Kra+18] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. *The Case for Learned Index Structures*. In: *SIGMOD Conference*. ACM, 2018, pp. 489–504.

[Kru+03] Sven Oliver Krumke, Willem de Paepe, Diana Poensgen, and Leen Stougie. *News from the online traveling repairman*. In: *Theor. Comput. Sci.* 295 (2003), pp. 279–294.

[KSS21] Janardhan Kulkarni, Stefan Schmid, and Pawel Schmidt. *Scheduling Opportunistic Links in Two-Tiered Reconfigurable Datacenters*. In: *SPAA*. ACM, 2021, pp. 318–327.

[Lab+84a] J. Labetoulle, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. *Preemptive Scheduling of Uniform Machines Subject to Release Dates*. In: *Progress in Combinatorial Optimization*. Elsevier, 1984, pp. 245–261.

[Lab+84b] Jacques Labetoulle, Eugene L Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. *Preemptive scheduling of uniform machines subject to release dates*. In: *Progress in combinatorial optimization*. Elsevier, 1984, pp. 245–261.

[Las+23] Alexandra Anna Lassota, Alexander Lindermayr, Nicole Megow, and Jens Schlöter. *Minimalistic Predictions to Schedule Jobs with Online Precedence Constraints*. In: *ICML*. Vol. 202. Proceedings of Machine Learning Research. PMLR, 2023, pp. 18563–18583.

[Lat+20] Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. *Online Scheduling via Learned Weights*. In: *SODA*. SIAM, 2020, pp. 1859–1877.

[Lav+21] Thomas Lavastida, Benjamin Moseley, R. Ravi, and Chenyang Xu. *Learnable and Instance-Robust Predictions for Online Matching, Flows and Load Balancing*. In: *ESA*. Vol. 204. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 59:1–59:17.

[Law78] Eugene L. Lawler. *Sequencing jobs to minimize total weighted completion time subject to precedence constraints*. In: *Annals of Discrete Mathematics* 2 (1978), pp. 75–90.

[LK78] Jan Karel Lenstra and A. H. G. Rinnooy Kan. *Complexity of Scheduling under Precedence Constraints*. In: *Oper. Res.* 26.1 (1978), pp. 22–35.

[LKB77] Jan Karel Lenstra, AHG Rinnooy Kan, and Peter Brucker. *Complexity of machine scheduling problems*. In: *Annals of discrete mathematics*. Vol. 1. Elsevier, 1977, pp. 343–362.

[LST90] Jan Karel Lenstra, David B. Shmoys, and Éva Tardos. *Approximation Algorithms for Scheduling Unrelated Parallel Machines*. In: *Math. Program.* 46 (1990), pp. 259–271.

[Li25] Shi Li. *Approximating Unrelated Machine Weighted Completion Time Using Iterative Rounding and Computer Assisted Proofs*. In: *SODA*. To appear. SIAM, 2025.

[LX21] Shi Li and Jiayi Xian. *Online Unrelated Machine Load Balancing with Predictions Revisited*. In: *ICML*. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021, pp. 6523–6532.

[LLW22] Honghao Lin, Tian Luo, and David P. Woodruff. *Learning Augmented Binary Search Trees*. In: *ICML*. Vol. 162. Proceedings of Machine Learning Research. PMLR, 2022, pp. 13431–13440.

[LM22] Alexander Lindermayr and Nicole Megow. *Permutation Predictions for Non-Clairvoyant Scheduling*. In: *SPAA*. ACM, 2022, pp. 357–368.

[LM24] Alexander Lindermayr and Nicole Megow. *Repository of papers on algorithms with predictions*. http://algorithms-with-predictions.github.io/ [Accessed: (August 15, 2024)]. 2024.

[LMR23] Alexander Lindermayr, Nicole Megow, and Martin Rapp. *Speed-Oblivious Online Scheduling: Knowing (Precise) Speeds is not Necessary*. In: *ICML*. Vol. 202. Proceedings of Machine Learning Research. PMLR, 2023, pp. 21312–21334.

[LMS22] Alexander Lindermayr, Nicole Megow, and Bertrand Simon. *Double Coverage with Machine-Learned Advice*. In: *ITCS*. Vol. 215. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 99:1–99:18.

[Lip+04] Maarten Lipmann, Xiwen Lu, Willem de Paepe, René Sitters, and Leen Stougie. *On-Line Dial-a-Ride Problems Under a Restricted Information Model*. In: *Algorithmica* 40.4 (2004), pp. 319–329.

[Liu+09] Ming Liu, Chengbin Chu, Yinfeng Xu, and Feifeng Zheng. *Online scheduling on m uniform machines to minimize total (weighted) completion time*. In: *Theor. Comput. Sci.* 410.38-40 (2009), pp. 3875–3881.

[Lov75] László Lovász. *On the ratio of optimal integral and fractional covers*. In: *Discret. Math.* 13.4 (1975), pp. 383–390.

[LV18] Thodoris Lykouris and Sergei Vassilvitskii. *Competitive Caching with Machine Learned Advice*. In: *ICML*. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 3302–3311.

[LV21] Thodoris Lykouris and Sergei Vassilvitskii. *Competitive Caching with Machine Learned Advice*. In: *J. ACM* 68.4 (2021), 24:1–24:25.

[Lyn+17] Theo Lynn, Pierangelo Rosati, Arnaud Lejeune, and Vincent C. Emeakaroha. *A Preliminary Review of Enterprise Serverless Cloud Computing (Function-as-a-Service) Platforms*. In: *CloudCom*. IEEE Computer Society, 2017, pp. 162–169.

[MNS12] Mohammad Mahdian, Hamid Nazerzadeh, and Amin Saberi. *Online Optimization with Uncertain Information*. In: *ACM Trans. Algorithms* 8.1 (2012), 2:1–2:29.

[McC+23] Samuel McCauley, Benjamin Moseley, Aidin Niaparast, and Shikha Singh. *Online List Labeling with Predictions*. In: *NeurIPS*. 2023.

[McN59] Robert McNaughton. *Scheduling with Deadlines and Loss Functions*. In: *Manage. Sci.* 6.1 (1959), pp. 1–12.

[MV17] Andres Munoz Medina and Sergei Vassilvitskii. *Revenue Optimization with Approximate Bid Predictions*. In: *NIPS*. 2017, pp. 1858–1866.

[MS04] Nicole Megow and Andreas S. Schulz. *On-line scheduling to minimize average completion time revisited*. In: *Oper. Res. Lett.* 32.5 (2004), pp. 485–490.

[MUV04] Nicole Megow, Marc Uetz, and Tjark Vredeveld. *Stochastic Online Scheduling on Parallel Machines*. In: *WAOA*. Vol. 3351. Lecture Notes in Computer Science. Springer, 2004, pp. 167–180.

[MUV05] Nicole Megow, Marc Uetz, and Tjark Vredeveld. *Models and Algorithms for Stochastic Online Scheduling*. In: *Algorithms for Optimization with Incomplete Information*. Vol. 05031. Dagstuhl Seminar Proceedings. IBFI, Schloss Dagstuhl, Germany, 2005.

[Meh13] Aranyak Mehta. *Online Matching and Ad Allocation*. In: *Found. Trends Theor. Comput. Sci.* 8.4 (2013), pp. 265–368.

[Meh+07] Aranyak Mehta, Amin Saberi, Umesh V. Vazirani, and Vijay V. Vazirani. *AdWords and generalized online matching*. In: *J. ACM* 54.5 (2007), p. 22.

[MM16] Julián Mestre and Nicole Megow. *Universal Sequencing on an Unreliable Machine*. In: *Encyclopedia of Algorithms*. 2016, pp. 2304–2308.

[MV20] Michael Mitzenmacher and Sergei Vassilvitskii. *Algorithms with Predictions*. In: *Beyond the Worst-Case Analysis of Algorithms*. Cambridge University Press, 2020, pp. 646–662.

[MV22a] Michael Mitzenmacher and Sergei Vassilvitskii. *Algorithms with predictions*. In: *Commun. ACM* 65.7 (2022), pp. 33–35.

[MV22b] Benjamin Moseley and Shai Vardi. *The efficiency-fairness balance of Round Robin scheduling*. In: *Oper. Res. Lett.* 50.1 (2022), pp. 20–27.

[MPT94] Rajeev Motwani, Steven J. Phillips, and Eric Torng. *Non-Clairvoyant Scheduling*. In: *Theor. Comput. Sci.* 130.1 (1994), pp. 17–47.

[Mou03] Hervé Moulin. *Fair division and collective welfare*. MIT Press, 2003.

[Nas50] John F. Nash Jr. *The bargaining problem*. In: *Econometrica* 18.2 (1950), pp. 155–162.

[Ngu13] Kim Thang Nguyen. *Lagrangian Duality in Online Scheduling with Resource Augmentation and Speed Scaling*. In: *ESA*. Vol. 8125. Lecture Notes in Computer Science. Springer, 2013, pp. 755–766.

[NK13] Rad Niazadeh and Robert D. Kleinberg. *A Unified Approach to Online Allocation Algorithms via Randomized Dual Fitting*. In: *CoRR* abs/1308.5444 (2013).

[Nis+07] Noam Nisan, Tim Roughgarden, Éva Tardos, and Vijay V. Vazirani, eds. *Algorithmic Game Theory*. Cambridge University Press, 2007.

[Oxl11] James Oxley. *Matroid Theory*. 2nd ed. Oxford University Press, 2011.

[PTV21] Ioannis Panageas, Thorben Tröbst, and Vijay V. Vazirani. *Combinatorial Algorithms for Matching Markets via Nash Bargaining: One-Sided, Two-Sided and Non-Bipartite*. In: *CoRR* abs/2106.02024 (2021).

[Pin22] Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer International Publishing, 2022.

[PS16] Lukás Polácek and Ola Svensson. *Quasi-Polynomial Local Search for Restricted Max-Min Fair Allocation*. In: *ACM Trans. Algorithms* 12.2 (2016), 13:1–13:13.

[PST04] Kirk Pruhs, Jirí Sgall, and Eric Torng. *Online Scheduling*. In: *Handbook of Scheduling*. Chapman and Hall/CRC, 2004.

[PSK18] Manish Purohit, Zoya Svitkina, and Ravi Kumar. *Improving Online Algorithms via ML Predictions*. In: *NeurIPS*. 2018, pp. 9684–9693.

[QS02] Maurice Queyranne and Maxim Sviridenko. *A (2+epsilon)-approximation algorithm for the generalized preemptive open shop problem with minsum objective*. In: *J. Algorithms* 45.2 (2002), pp. 202–212.

[RS13] Alexander Rakhlin and Karthik Sridharan. *Online Learning with Predictable Sequences*. In: *COLT*. Vol. 30. JMLR Workshop and Conference Proceedings. JMLR.org, 2013, pp. 993–1019.

[Rap+21] Martin Rapp, Anuj Pathania, Tulika Mitra, and Jörg Henkel. *Neural Network-Based Performance Prediction for Task Migration on S-NUCA Many-Cores*. In: *IEEE Trans. Computers* 70.10 (2021), pp. 1691–1704.

[Rou20] Tim Roughgarden, ed. *Beyond the Worst-Case Analysis of Algorithms*. Cambridge University Press, 2020.

[SE24] Karim Abdel Sadek and Marek Eliás. *Algorithms for Caching and MTS with reduced number of predictions*. In: *ICLR*. OpenReview.net, 2024.

[Sch68] Linus Schrage. *Letter to the Editor - A Proof of the Optimality of the Shortest Remaining Processing Time Discipline*. In: *Oper. Res.* 16.3 (1968), pp. 687–690.

[Sch03] Alexander Schrijver. *Combinatorial Optimization – Polyhedra and Efficiency*. Springer, 2003.

[SS02a] Andreas S Schulz and Martin Skutella. *The power of α-points in preemptive single machine scheduling*. In: *Journal of Scheduling* 5.2 (2002), pp. 121–133.

[SS97] Andreas S. Schulz and Martin Skutella. *Random-Based Scheduling: New Approximations and LP Lower Bounds*. In: *RANDOM*. Vol. 1269. Lecture Notes in Computer Science. Springer, 1997, pp. 119–133.

[SS02b] Andreas S. Schulz and Martin Skutella. *Scheduling Unrelated Machines by Randomized Rounding*. In: *SIAM J. Discret. Math.* 15.4 (2002), pp. 450–469.

[SW99] Petra Schuurman and Gerhard J. Woeginger. *Polynomial time approximation algorithms for machine scheduling: Ten open problems*. In: *Journal of Scheduling* 2.5 (1999), pp. 203–213.

[Sch22] Uwe Schwiegelshohn. *Online Total Completion Time Scheduling on Parallel Identical Machines*. In: *CoRR* abs/2207.08550 (2022).

[Ser78] Anatoliy I. Serdyukov. *O nekotorykh ekstremalnykh obkhodakh v grafakh (On some extremal walks in graphs)*. In: *Upravlyaemye Sistemy (in Russian)* 17 (1978), pp. 76–69.

[Sev74] Kenneth C. Sevcik. *Scheduling for Minimum Total Loss Using Service Time Distributions*. In: *J. ACM* 21.1 (1974), pp. 66–75.

[SBW19] Mohammad Shahrad, Jonathan Balkind, and David Wentzlaff. *Architectural Implications of Function-as-a-Service Computing*. In: *MICRO*. ACM, 2019, pp. 1063–1075.

[SB14] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning - From Theory to Algorithms*. Cambridge University Press, 2014.

[SV05] Evgeny V. Shchepin and Nodari Vakhania. *An optimal rounding gives a better approximation for scheduling unrelated machines*. In: *Oper. Res. Lett.* 33.2 (2005), pp. 127–133.

[ST93] David B. Shmoys and Éva Tardos. *An approximation algorithm for the generalized assignment problem*. In: *Math. Program.* 62 (1993), pp. 461–474.

[Sit05] René Sitters. *Complexity of preemptive minsum scheduling on unrelated parallel machines*. In: *J. Algorithms* 57.1 (2005), pp. 37–48.

[Sit10] René Sitters. *Efficient Algorithms for Average Completion Time Scheduling*. In: *IPCO*. Vol. 6080. Lecture Notes in Computer Science. Springer, 2010, pp. 411–423.

[Sit17] René Sitters. *Approximability of average completion time scheduling on unrelated machines*. In: *Math. Program.* 161.1-2 (2017), pp. 135–158.

[Sku98] Martin Skutella. *Approximation and Randomization in Scheduling*. Doctoral Thesis. Technical University of Berlin, Germany, 1998.

[Sku01] Martin Skutella. *Convex quadratic and semidefinite programming relaxations in scheduling*. In: *J. ACM* 48.2 (2001), pp. 206–242.

[Smi56] Wayne E Smith. *Various optimizers for single-stage production*. In: *Naval Research Logistics Quarterly* 3.1-2 (1956), pp. 59–66.

[SZ20] Clifford Stein and Mingxian Zhong. *Scheduling When You Do Not Know the Number of Machines*. In: *ACM Trans. Algorithms* 16.1 (2020), 9:1–9:20.

[Sve12] Ola Svensson. *Santa Claus Schedules Jobs on Unrelated Machines*. In: *SIAM J. Comput.* 41.5 (2012), pp. 1318–1341.

[The19] The kernel development community. *Energy Aware Scheduling – The Linux Kernel Documentation*. https://www.kernel.org/doc/html/v5.3/scheduler/sched-energy.html. 2019.

[Tim03] Vadim G. Timkovsky. *Identical parallel machines vs. unit-time shops and preemptions vs. chains in scheduling complexity*. In: *Eur. J. Oper. Res.* 149.2 (2003), pp. 355–376.

[TVZ22] Vera Traub, Jens Vygen, and Rico Zenklusen. *Reducing Path TSP to TSP*. In: *SIAM J. Comput.* 51.3 (2022), pp. 20–24.

[TV24] Thorben Tröbst and Vijay V. Vazirani. *Cardinal-Utility Matching Markets: The Quest for Envy-Freeness, Pareto-Optimality, and Efficient Computability*. In: *CoRR* abs/2402.08851 (2024).

[Var76] Hal R. Varian. *Two problems in the theory of fairness*. In: *J. Public Econ.* 5.3-4 (1976), pp. 249–260.

[VY21] Vijay V. Vazirani and Mihalis Yannakakis. *Computational Complexity of the Hylland-Zeckhauser Scheme for One-Sided Matching Markets*. In: *ITCS*. Vol. 185. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 59:1–59:19.

[VW14] José Verschae and Andreas Wiese. *On the configuration-LP for scheduling on unrelated machines*. In: *J. Sched.* 17.4 (2014), pp. 371–383.

[Ves97] Arjen Vestjens. *On-line machine scheduling*. PhD thesis. Technische Universiteit Eindhoven, 1997.

[Vis21] Nisheeth K. Vishnoi. *Algorithms for Convex Optimization*. Cambridge University Press, 2021.

[WZ20] Alexander Wei and Fred Zhang. *Optimal Robustness-Consistency Trade-offs for Learning-Augmented Online Algorithms*. In: *NeurIPS*. 2020.

[Wil02] David P. Williamson. *The primal-dual method for approximation algorithms*. In: *Math. Program.* 91.3 (2002), pp. 447–478.

[WS11] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

[Woe02] Gerhard J. Woeginger. *Open problems in the theory of scheduling*. In: *Bull. EATCS* 76 (2002), pp. 67–83.

[XC12] Bo Xiong and Christine Chung. *Completion Time Scheduling and the WSRPT Algorithm*. In: *ISCO*. Vol. 7422. Lecture Notes in Computer Science. Springer, 2012, pp. 416–426.

[XL22] Chenyang Xu and Pinyan Lu. *Mechanism Design with Predictions*. In: *IJCAI*. ijcai.org, 2022, pp. 571–577.

[XM22] Chenyang Xu and Benjamin Moseley. *Learning-Augmented Algorithms for Online Steiner Tree*. In: *AAAI*. AAAI Press, 2022, pp. 8744–8752.

[ZLZ22a] Tianming Zhao, Wei Li, and Albert Y. Zomaya. *Real-Time Scheduling with Predictions*. In: *RTSS*. IEEE, 2022, pp. 331–343.

[ZLZ22b] Tianming Zhao, Wei Li, and Albert Y. Zomaya. *Uniform Machine Scheduling with Predictions*. In: *ICAPS*. AAAI Press, 2022, pp. 413–422.

Zusammenfassung

Schedulingprobleme sind grundlegende kombinatorische Optimierungsprobleme und allgegenwärtig in unserem Alltag. Beispielsweise liegt ein Schedulingproblem vor, wenn in einem Unternehmen Aufgaben auf Beschäftigte verteilt und deren Bearbeitung zeitlich geplant werden müssen. Jede Aufgabe ist unterschiedlich komplex und jeder Beschäftigte kann aufgrund seiner Spezialisierung und seiner individuellen Berufserfahrung jede Aufgabe unterschiedlich schnell bearbeiten. Da die Aufgaben möglichst schnell erledigt werden sollen, stellt sich die Frage, wann welcher Beschäftigte an welcher Aufgabe arbeiten soll. Bei wenigen Aufgaben und Beschäftigten kann ein guter Ablaufplan möglicherweise noch leicht gefunden werden. Bei vielen Beschäftigten und Aufgaben werden allerdings Algorithmen benötigt, um gute Lösungen zu finden, da es deutlich mehr mögliche Alternativen gibt und das Problem somit viel komplexer ist.

Diese Arbeit betrachtet dieses und ähnliche Schedulingprobleme in einem abstrakten Modell und stellt Lösungsalgorithmen mit beweisbaren Gütegarantien vor. In diesem abstrakten Modell besteht die Aufgabe darin, Jobs zeitlich auf Maschinen so einzuplanen, dass zu jedem Zeitpunkt auf jeder Maschine höchstens ein Job bearbeitet wird und zu jedem Zeitpunkt jeder Job auf höchstens einer Maschine bearbeitet wird. Manchmal werden auch weitere Einschränkungen betrachtet, wie zum Beispiel, dass sich ein Job über die gesamte Zeit auf höchstens einer bestimmten Maschine befinden darf. Jeder Job hat ein erforderliches Bearbeitungsvolumen und wird auf jeder Maschine mit einer bestimmten Geschwindigkeit bearbeitet. Diese Geschwindigkeiten können für jedes Job-Maschinen-Paar unterschiedlich sein; in diesem Fall nennt man die Maschinen *unverwandt* (engl. unrelated). In dieser Arbeit werden auch Spezialfälle von unverwandten Maschinen untersucht, die diese Geschwindigkeiten einschränken. Ein Job ist fertiggestellt, wenn sein benötigtes Bearbeitungsvolumen erreicht wurde. Der Zeitpunkt, zu dem ein Job fertiggestellt wird, wird sein Fertigstellungszeitpunkt genannt. Ein Ablaufplan ist zulässig, wenn er alle Jobs fertigstellt und gegebenenfalls noch weitere Eigenschaften erfüllt. In vielen Anwendungsszenarien wird ein zulässiger Ablaufplan gesucht, der eine bestimmte Zielfunktion optimiert. In dieser Arbeit werden hauptsächlich die folgenden zwei Zielfunktionen betrachtet, die minimiert werden sollen: der späteste Fertigstellungszeitpunkt und der durchschnittliche Fertigstellungszeitpunkt.

In dieser Arbeit werden Schedulingprobleme und zugehörige Algorithmen in verschiedenen Informationsmodellen betrachtet. Ein Informationsmodell beschreibt, welche Informationen einer Instanz eines Problems zu welchem Zeitpunkt einem Algorithmus zur Verfügung gestellt werden. Die in dieser Arbeit betrachteten Modelle lassen sich in drei große Kategorien einteilen: das Offline-Modell, das Online-Modell und das Learning-Augmented-Modell.

Im Offline-Modell stehen einem Algorithmus alle Informationen der Schedulinginstanz zur Verfügung. In diesem Modell werden Lösungsalgorithmen unter Laufzeitbeschränkungen untersucht. Das bedeutet, dass ein Algorithmus nur eine bestimmte Anzahl an elementaren

Operationen ausführen darf, um eine Lösung für eine Instanz zu berechnen, wobei diese Einschränkung von der Größe der Instanz abhängt, also zum Beispiel der Anzahl an Maschinen und Jobs. Unter dieser Restriktion können für viele Schedulingprobleme Algorithmen keine optimalen Lösungen für bestimmte Instanzen berechnen. Deshalb werden Approximationsalgorithmen betrachtet, die unter der Laufzeitbeschränkung Lösungen berechnen, die höchstens um einen bestimmten beweisbaren Faktor von einer optimalen Lösung abweichen. Im ersten Teil der Arbeit werden Schedulingprobleme und Approximationsalgorithmen im Offline-Modell behandelt.

Im Online-Modell werden einem Algorithmus die Informationen der Schedulinginstanz über die Zeit mitgeteilt und ein Algorithmus kann einen in der Vergangenheit liegenden Ablaufplan nicht mehr ändern. In dieser Arbeit werden zwei Online-Modelle betrachtet, die auch gemeinsam auftreten können. Im ersten Modell werden Jobs erst zu bestimmten Zeitpunkten verfügbar, von denen ein Algorithmus vorher nichts weiß. Im zweiten Modell sind die benötigten Bearbeitungsvolumina der Jobs unbekannt und ein Algorithmus kennt das genaue benötigte Bearbeitungsvolumen eines Jobs erst dann, wenn dieser fertiggestellt wird. Solche Algorithmen werden als nicht-hellseherisch bezeichnet. In Online-Modellen ist es für Algorithmen oft nicht möglich optimale Lösungen zu finden, da dazu bereits zu Beginn eines Ablaufplans Entscheidungen getroffen werden müssten, die aber erst nach Bekanntwerden der gesamten Instanz Teil einer optimalen Lösung werden. Daher wird versucht, kompetitive Algorithmen zu entwerfen, die Lösungen berechnen, die höchstens um einen bestimmten beweisbaren Faktor von einer optimalen Lösung abweichen. Im zweiten Teil der Arbeit werden diese Online-Modelle untersucht und verbesserte Algorithmen und Analysen präsentiert.

Das Learning-Augmented-Modell liegt zwischen dem Online-Modell und dem Offline-Modell. Dieses Modell ist anwendbar, wenn es in einem Online-Modell unrealistisch erscheint, dass überhaupt keine Informationen über die unsicheren Informationen zur Verfügung stehen, aber auch nicht garantiert werden kann, dass diese Informationen präzise zur Verfügung stehen, wie es im Offline-Modell der Fall ist. Stattdessen wird im Learning-Augmented-Modell angenommen, dass es eine Vorhersage über die unsicheren Instanzparameter gibt, zum Beispiel, wie viel Bearbeitungsvolumen ein Job benötigt. Diese Vorhersage muss nicht der Wahrheit entsprechen und es werden keine Annahmen über ihre Qualität getroffen. Solche Vorhersagen können beispielsweise durch maschinelles Lernen aus historischen Daten generiert werden. Das Ziel ist es, Algorithmen zu entwickeln, die diese Vorhersagen nutzen und dadurch beweisbar besser sind als Algorithmen, die keine Vorhersagen verwenden. Gleichzeitig muss sichergestellt werden, dass die Algorithmen der Vorhersage nicht zu sehr vertrauen, um auch dann kompetitiv zu sein, wenn die Vorhersage völlig falsch ist. Im dritten und letzten Teil dieser Arbeit werden Algorithmen für Learning-Augmented-Modelle verschiedener Schedulingprobleme entworfen und untersucht.