APPROXIMATION SCHEMES FOR MACHINE SCHEDULING

MARTEN MAACK

Dissertation
zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften
(Dr. rer. nat.)
der Technischen Fakultät
der Christian-Albrechts-Universität zu Kiel
eingereicht im Jahr 2019

1. GUTACHTER Prof. Dr. Klaus Jansen

2. GUTACHTER Prof. Dr. Friedhelm Meyer auf der Heide

3. GUTACHTER Prof. Dr. Asaf Levin

DATUM DER DISPUTATION 7. Mai 2020

ABSTRACT

In the classical problem of makespan minimization on identical parallel machines, or machine scheduling for short, a set of jobs has to be assigned to a set of machines. The jobs have a processing time and the goal is to minimize the latest finishing time of the jobs. Machine scheduling is well known to be NP-hard and thus there is no polynomial time algorithm for this problem that is guaranteed to find an optimal solution unless P=NP. There is, however, a polynomial time approximation scheme (PTAS) for machine scheduling, that is, a family of $(1+\varepsilon)$-approximations for each $\varepsilon > 0$. Whether a problem admits an approximation scheme or not is a fundamental question in approximation theory. In the present work, we consider this question for several variants of machine scheduling.

For instance, we study the problem where the machines are partitioned into a constant number of types and the processing time of the jobs is also dependent on the machine type it is assigned to. We present an efficient PTAS (EPTAS)—a PTAS whose running time is a product of some function in $1/\varepsilon$ and a polynomial in the input length—for this problem and variants thereof.

In the restricted assignment problem, each job may only be assigned to a given subset of machines. We show that certain cases of restricted assignment do not admit a PTAS unless P=NP, e.g., the case in which the machines are linearly ordered and each job is eligible on a set of consecutive machines. Moreover, we introduce a graph framework based on the restrictions of the jobs and use it in the design of approximation schemes for several variants of restricted assignment generalizing and unifying many of the known PTAS results.

Furthermore, we introduce an enhanced integer programming formulation for assignment problems, show that it can be efficiently solved, and use it in the EPTAS design for variants of machine scheduling with setup times. For one of the problems, we show that there is also a PTAS in the case with uniform machines, where machines have speeds influencing the processing times of the jobs.

Lastly, we consider cases in which each job requires a certain amount of a shared renewable resource and the processing time is depended on the amount of resource it receives or not. We present asymptotic fully polynomial time approximation schemes (AFPTAS) for the problems: For any $\varepsilon > 0$ a schedule is provided whose length lies within a factor of $(1+\varepsilon)$ of the optimum value except for an additional additive error depending on the maximal processing time and $1/\varepsilon$.

ZUSAMMENFASSUNG

Im klassischen Problem des Machine Scheduling muss eine Menge von Jobs einer Menge von Maschinen zugewiesen werden. Die Jobs haben eine Bearbeitungszeit und das Ziel ist es, den Zeitpunkt zu minimieren an dem der letzte Job vollständig bearbeitet ist. Machine Scheduling ist NP-vollständig und daher gibt es keinen Algorithmus, der in polynomieller Zeit garantiert eine optimale Lösung für das Problem findet, es sei denn P=NP. Es gibt jedoch ein Approximationsschema mit polynomieller Laufzeit (PTAS) für Machine Scheduling, d.h. eine Familie von $(1+\varepsilon)$-Approximationen für jedes $\varepsilon > 0$. Ob ein Problem ein Approximationsschema zulässt oder nicht, ist eine grundlegende Frage der Approximationstheorie. In dieser Arbeit betrachten wir diese Frage für mehrere Varianten des Machine Scheduling.

So betrachten wir beispielsweise Varianten, bei denen die Maschinen in eine konstante Anzahl von Typen partitioniert sind und die Bearbeitungszeit der Jobs vom Maschinentyp abhängt. Wir präsentieren ein effizientes PTAS (EPTAS) für dieses Problem, d.h. ein PTAS dessen Laufzeit durch das Produkt einer Funktion in $1/\varepsilon$ und eines Polynoms in der Eingabelänge abgeschätzt werden kann.

Bei dem Restricted Assignment Problem darf jeder Job nur einer gegebenen Teilmenge von Maschinen zugeordnet werden. Wir zeigen, dass bestimmte Spezialfälle von Restricted Assignment kein PTAS zulassen (sofern P≠NP), z.B. der Fall, in dem die Maschinen linear geordnet sind und jeder Job auf einer Menge von konsekutiven Maschinen zulässig ist. Weiterhin führen wir ein auf den Zulässigkeiten basierendes Graphen-Framework ein und verwenden es beim PTAS-Design für diverse Varianten des Restricted Assignment.

Darüber hinaus entwickeln wir eine erweiterte ganzzahlige Optimierungsformulierung für Zuweisungsprobleme, zeigen, dass sie effizient gelöst werden kann, und verwenden sie im EPTAS-Design für Varianten von Machine Scheduling mit Setup Zeiten. Für eine dieser Varianten zeigen wir zusätzlich, dass es auch für den Fall mit uniformen Maschinen ein PTAS gibt. Uniforme Maschinen haben Geschwindigkeiten, die die Bearbeitungszeiten der Jobs beeinflussen.

Schließlich betrachten wir Fälle, in denen jeder Job eine bestimmte Menge einer geteilten erneuerbaren Ressource benötigt und die Bearbeitungszeit von der Menge der erhaltenen Ressource abhängt oder nicht. Wir präsentieren asymptotische, vollständig polynomielle Approximationsschemata (AFPTAS) für die Probleme: Für jedes $\varepsilon > 0$ wird ein Schedule gefunden dessen Länge, bis auf einen zusätzlichen additiven Fehler abhängig von der maximalen Ausführungszeit und $1/\varepsilon$, höchstens um einen Faktor von $(1+\varepsilon)$ vom Optimum abweicht.

PUBLICATIONS

The present work is based on the following publications:

[78] Klaus Jansen and Marten Maack. "An EPTAS for Scheduling on Unrelated Machines of Few Different Types." In: *Algorithmica* 81.10 (2019), pp. 4134–4164. DOI: 10.1007/s00453-019-00581-w.

[79] Klaus Jansen, Marten Maack, and Alexander Mäcker. "Scheduling on (Un-)Related Machines with Setup Times." In: *2019 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2019, Rio de Janeiro, Brazil, May 20-24, 2019*. 2019, pp. 145–154. DOI: 10.1109/IPDPS.2019.00025.

[81] Klaus Jansen, Marten Maack, and Malin Rau. "Approximation Schemes for Machine Scheduling with Resource (In-)dependent Processing Times." In: *ACM Trans. Algorithms* 15.3 (2019), 31:1–31:28. DOI: 10.1145/3302250.

[82] Klaus Jansen, Marten Maack, and Roberto Solis-Oba. "Structural Parameters for Scheduling with Assignment Restrictions." In: *Algorithms and Complexity - 10th International Conference, CIAC 2017, Athens, Greece, May 24-26, 2017, Proceedings*. 2017, pp. 357–368. DOI: 10.1007/978-3-319-57586-5_30.

[86] Klaus Jansen, Kim-Manuel Klein, Marten Maack, and Malin Rau. "Empowering the Configuration-IP - New PTAS Results for Scheduling with Setups Times." In: *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*. 2019, 44:1–44:19. DOI: 10.4230/LIPIcs.ITCS.2019.44.

[111] Marten Maack and Klaus Jansen. "Inapproximability Results for Scheduling with Interval and Resource Restrictions." In: *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France*. Vol. 154. 2020, 5:1–5:18. DOI: 10.4230/LIPIcs.STACS.2020.5.

All of the above publications have been peer-reviewed. Conference versions of [78] and [81] have previously been published in the proceedings of WADS 2017 [77] and SODA 2016 [80], respectively.

ACKNOWLEDGMENTS

I sincerely thank my advisor Klaus Jansen for supporting this work. Many thanks to my other colleagues in the algorithms and complexity group in Kiel, namely, Sebastian Berndt, Max Deppert, Kilian Grage, Ute Iaquinto, Maren Kaluza, Kim-Manuel Klein, Stefan Kraft, Felix and Kati Land, Alexandra Lassota, Parvaneh Massouleh, Niklas Paulsen, Malin Rau, and Lars Rohwedder. I had a marvelous time collaborating, traveling, debating, and working along side with you day to day.

As part of my studies, I visited the Western University in London, Ontario for six months. I wish to thank Roberto Solis-Oba for supervising my work during that time and for generally being a great host. Thanks also to Daniel Page with whom I collaborated while visiting and shared an office as well as many exciting conversations.

I would like to thank all my co-authors with whom I have worked on the papers contributing to this work, namely, Klaus Jansen, Kim-Manuel Klein, Alexander Mäcker, Malin Rau, and Roberto Solis-Oba.

A special thanks goes to Kim-Manuel Klein, Alexandra Lassota, and Daniel Page for proofreading parts of this work.

Finally, I thank my family and friends, and especially my wife Beate, my mother Gabi, and my father Rainer for their unconditional support and advice, and for generally putting up with me.

CONTENTS

1

INTRODUCTION

Consider the problem of makespan minimization on identical parallel machines, called *machine scheduling* for short: Given a set $\mathcal{M}$ of m machines and a set $\mathcal{J}$ of n jobs each with a processing time or size p_j, the goal is to find an assignment $\sigma : \mathcal{J} \to \mathcal{M}$ of the jobs to the machines such that the maximal load any machine receives is minimized (see Figure 1.1). The maximal load $C_{\max}(\sigma) = \max_{i \in \mathcal{M}} \sum_{j \in \sigma^{-1}(i)} p_j$ is also called the *makespan* and σ a *schedule*. Machine scheduling is a classical combinatorial optimization problem—problems with a discrete and usually finite space of feasible solutions in which the goal is to find an optimal solution with respect to a given objective function. Like for many combinatorial optimization problems, the study of machine scheduling can be motivated from practical applications, e.g., from scheduling in production facilities or the assignment of computational tasks to identical processors. On the other hand, their simple combinatorial structure arguably already motivates their study from a purely theoretical perspective. In this work, we study algorithms for problems closely related to machine scheduling. We focus on the theoretical perspective, but consider many variants that capture additional difficulties that typically arise in practice, like, e.g., assignment restrictions, setup times or the influence of additional resources.

ALGORITHMS FOR MACHINE SCHEDULING. Finding a sensible assignment for a given machine scheduling instance is not very hard. For example, one could simply consider one job after another and assign it to a machine that so far received minimal load. We call this heuristic algorithm HEU in the following. However, the solutions generated by this procedure will usually not be optimal (see Figure 1.2). One could improve the procedure by first sorting the jobs by

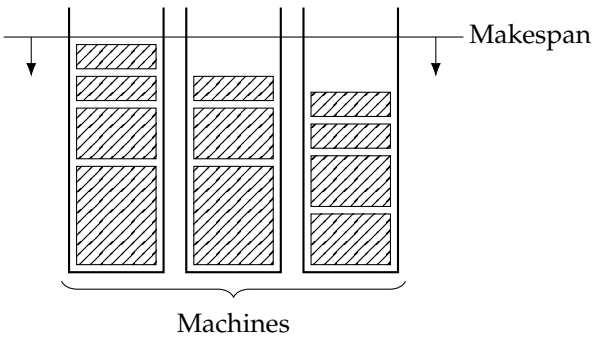


Figure 1.1: A schedule for a machine scheduling instance with 11 jobs and 3 machines. The jobs are represented by the hatched rectangles.

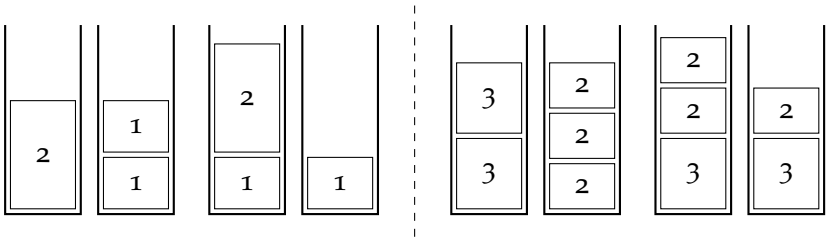


Figure 1.2: Two simple examples visualizing that HEU and HEU* are not guaranteed to find optimal solutions. In the example on the left, an optimal solution is depicted left and the solution produced by HEU when the big job is considered last on the right. On the right, again an optimal solution is depicted left and a solution produced by HEU* on the right. The depicted numbers correspond to the processing times.

decreasing processing times, but still the generated solutions are not guaranteed to be optimal (see Figure 1.2). We call the improved heuristic HEU*.

On the other hand, it is also not particularly difficult to find an optimal solution—we could try each of the m machine as the place of assignment for each of the n jobs yielding m^n possibilities. We can improve upon this brute force approach, e.g., by using a dynamic program in which the machines are considered one after another and a best partial solution for each subset of jobs is remembered. This approach yields a running time of $2^{\mathcal{O}(n)}m$ (see Section 5.2). While there are more elaborate known approaches to solve this problem, they all share the property that their running time is exponential in the input length. Therefore, they scale very badly: Say we had an algorithm with running time 2^n and an instance that is solved by it in 1 second. Then adding 10 jobs increases the running time to a quarter hour, another 10 to a dozen days, another 10 to a few decades, and adding 60 jobs in total would give a running time that is much longer than the estimated age of the earth. The heuristic procedures HEU and HEU*, on the other hand, have a running time that is at most quadratic in the input length, that is, increasing the number of jobs by a *factor* of 60 would give a running time of at most an hour in a corresponding sample calculation.

P VS. NP. Algorithms whose running times are bounded by linear or quadratic functions in the input length are deemed desirable in most contexts. However, considering the asymptotic scaling behaviour, any polynomial is preferable to any exponential function. Indeed, the question of whether a problem admits a polynomial time algorithm is considered fundamental in theoretical computer science. An early discussion of this idea can be found in a work by Edmonds [41] from 1965, who studied a polynomial time algorithm for the maximum matching problem and wrote:

> There is an obvious finite algorithm, but that algorithm increases in difficulty exponentially with the size of the graph. It is by no means obvious whether *or not* there exists an algorithm whose difficulty increases only algebraically [i.e., polynomially] with the size of the graph.
>
> The mathematical significance of this paper rests largely on the assumption that the two preceding sentences have mathematical meaning.[...]
>
> For practical purposes the difference between algebraic and exponential order is often more crucial than the difference between finite and non-finite.

In the early 1970s, Cook [33] and Karp [91] introduced the class of NP-complete decision problems[1], which are widely believed not to admit (deterministic) polynomial time algorithms. The class of decision problems that can be solved in polynomial time is denoted as P, while NP denotes the class of decision problems for which, roughly speaking, there exist solutions for positive instances that can be *verified* in polynomial time. The former class is a subclass of the latter, and P≠NP is a widely accepted and famous conjecture. Resolving it is one of the so called Millennium Prize Problems—a collection of problems in mathematics that are considered to be particularly important and hard. Now, a problem is called NP-complete if it is both in NP and NP-hard. The latter property essentially means that a polynomial time algorithm for any of them would imply such an algorithm for all problems in NP[2]. Many fundamental problems are well-known to be NP-complete including the decision versions[3] of numerous combinatorial optimization problems (see [51]). This is also the case for machine scheduling. Since solving the optimization version implies solving the decision version of a problem, there is no polynomial time algorithm for machine scheduling that is guaranteed to find an optimal solution unless P=NP. However, there are intriguing approaches to deal with this problem.

APPROXIMATION ALGORITHMS. Consider the heuristic algorithms HEU and HEU*. Both algorithms assign each job to a least loaded machine, and therefore no machine may receive more load than the average load and one additional job which may have maximal size. Hence, the load of each machine is upper bounded by $\max_{j\in\mathcal{J}} p_j + (\sum_{j\in\mathcal{J}} p_j)/m$. Both $\max_{j\in\mathcal{J}} p_j$ and $(\sum_{j\in\mathcal{J}} p_j)/m$ are lower bounds for the optimal objective value, and therefore the objective value of a solution produced by the heuristics is at most twice as big as the optimal one.

1 Problems for which a yes-or-no question has to be answered.

2 For proper definitions and background information on complexity theory, we refer to the textbook by Arora and Barak [8].

3 For optimization problems, one can state the question of whether a solution with at most/at least a certain objective value exists.

Let $\textsc{alg}(I)$ denote the objective value an algorithm ALG computes for an input instance I and $\textsc{opt}(I)$ be the corresponding optimal value. An algorithm ALG is called *approximation algorithm* with *ratio* α or α-*approximation* if $\textsc{alg}(I) \leqslant \alpha\,\textsc{opt}(I)$ in case of a minimization problem or $\alpha\,\textsc{alg}(I) \geqslant \textsc{opt}(I)$ in case of a maximization problem. The ratio α is also called the *approximation guarantee* or *rate* of the algorithm. Furthermore, if not stated otherwise, we use the term approximation algorithm as a short hand for *polynomial time* approximation algorithm. Using the above notation, the heuristics HEU and HEU* are 2-approximations.

The concept of approximation algorithms was formally introduced by Johnson [87] in 1974, however, there were earlier results proving approximation guarantees of algorithms. For instance, in 1966 and 1969 Graham [57, 58] considered the heuristics HEU and HEU* and proved them to be 2- and $4/3$-approximations, respectively. The algorithm HEU is usually called *list scheduling* and the prioritization of large jobs used in HEU* is known as the *LPT* (largest processing time first) rule . These are considered among the first results in the field of approximation algorithms.

When studying approximation algorithms, it is rather natural to ask which is the best possible approximation ratio one could obtain for a given problem or even whether there exists a best ratio. For machine scheduling, the answer to the latter question turns out to be negative.

APPROXIMATION SCHEMES. For a given problem, a *polynomial time approximation scheme (PTAS)* is a family of algorithms that includes a $(1+\varepsilon)$-approximation for each $\varepsilon > 0$. We also study approximation schemes with higher requirements on the running time. In particular, a PTAS is called *efficient* (EPTAS) if it has a running time of the form $f(1/\varepsilon)|I|^{O(1)}$, where f is some computable function and $|I|$ the input length. Furthermore, an EPTAS is called *fully polynomial* (FPTAS) if the function f is a polynomial.

Whether a problem admits a PTAS or not is a fundamental question in approximation. In 1987, Hochbaum and Shmoys [65] designed a PTAS for machine scheduling and hence showed that any approximation guarantee strictly above 1 can be achieved for this problem[4]. Since then, there have been many more results concerning approximation schemes for machine scheduling and variants thereof (like, e.g., [5, 46, 47, 66, 72, 73, 119, 122, 133]).

The present work aims at deepening the understanding in this line of research. To this end, we design approximation schemes for many important variants of machine scheduling. We introduce new techniques in this context and show that our results work for a wide variety of problems. Furthermore, we present inapproximability re-

4 We present an example PTAS for machine scheduling in Section 2.2 which is closely related to this result.

sults that rule out the existence of approximation schemes for certain cases under the assumption that P≠NP.

CLASSICAL SCHEDULING PROBLEMS. In order to discuss the results of this work in more detail, we first have to introduce some classical variants of machine scheduling studied in the literature. The most common two are called makespan minimization on *uniformly related* and *unrelated* parallel machines, and are abbreviated as *uniform* and *unrelated scheduling* in the following. In the problem of unrelated scheduling, the processing time of a job depends on the machine it is processed on, that is, the input includes a processing time p_{ij} for each machine i and job j. Uniform scheduling can be seen as a special case of unrelated scheduling where each job j has a size p_j, each machine i a speed s_i, and the processing time of job j on machine i is given by $p_{ij} = p_j/s_i$.

For uniform scheduling, there is a PTAS due to Hochbaum and Shmoys [66], and both machine [5] and uniform scheduling [72] are known to admit an EPTAS. On the other hand, the problems are well known to be strongly NP-hard[5], which implies that there is no FPTAS for them unless P=NP. If the number of machines is considered constant, already unrelated scheduling admits an FPTAS [67]. However, there is no approximation algorithm with a ratio smaller than 1.5 for unrelated scheduling unless P=NP. This was shown by Lenstra, Shmoys and Tardos [108] and already holds for the so called *restricted assignment* problem, where each job has a size p_j and $p_{ij} \in \{p_j, \infty\}$. For each job j, the machines i with $p_{ij} = p_j$ are called eligible and we say that the assignment of j to other machines is restricted. Lestra et al. also provided a 2-approximation for unrelated scheduling. Closing (or narrowing) the gap between 1.5-inapproximability and 2-approximation for either unrelated scheduling or restricted assignment is considered one of the most important open problems in approximation [145] and scheduling theory [132]. This also motivates the study of special cases, and whether these cases admit approximation schemes in particular, in order to better understand the approximability of these problems.

For a broader overview of scheduling theory, we refer to the following textbook [127] and survey [28].

CONTRIBUTIONS AND ORGANIZATION

In Chapter 2, we discuss further basic concepts and notation and present an example PTAS for machine scheduling. Each of the remaining chapters is based on a prior publication and can, for the most part, be read independently. Exceptions to this rule are Chapter 5 and 7 which are closely related to the respective direct predecessor chapters and therefore cover the related literature and motivation of the

5 They remain NP-hard, even if the numbers in the input have unary encoding.

work only to a limited extent. In the following, we briefly discuss the problems considered in each chapter as well as the obtained results.

CHAPTER 3. In this chapter, we consider the variant of unrelated scheduling with a constant number K of machine types. Two machines have the same type if all jobs have the same processing time for them. Note that for $K = 1$, we have the problem of machine scheduling. We present an EPTAS for the problem with a running time of

$$2^{\mathcal{O}(K \log(K) 1/\varepsilon \log^4 1/\varepsilon)} + \text{poly}(|I|),$$

where $|I|$ denotes the input length. Furthermore, we study three other problem variants and present an EPTAS for each of them: The Santa Claus problem, where the minimum machine load has to be maximized; the case of unrelated scheduling with a constant number of *uniform* types, where machines of the same type behave like uniform machines; and the multidimensional vector scheduling variant of the problem, where both the dimension and the number of machine types are constant. For the Santa Claus problem we achieve the same asymptotic running time. The results are obtained, using mixed integer linear programming and rounding techniques. This chapter is based on [77].

CHAPTER 4. We consider two variants of the restricted assignment problem. In the case of interval restrictions the machines can be totally ordered such that jobs are eligible on consecutive machines. We show that there is no PTAS unless P=NP for this variant. The question of whether a PTAS for this problem exists was stated as an open problem before, and PTAS results for special cases of this variant are known. Interestingly, there is a paper claiming to have found a PTAS for this problem but it is flawed.

Furthermore, we consider a variant with resource restrictions where the sets of eligible machines are of the following form: There is a fixed number of (renewable) resources, each machine has a capacity, and each job a demand for each resource. A job is eligible on a machine if its demand is at most as big as the capacity of the machine for each resource. For one resource, this problem has been intensively studied under several different names and is known to admit a PTAS, and for two resources the variant with interval restrictions is contained as a special case. Moreover, the version with multiple resources is closely related to unrelated scheduling with a low rank processing time matrix. We show that there is no polynomial time approximation algorithm with a rate smaller than $48/47 \approx 1.02$ or 1.5 for scheduling with resource restrictions with 2 or 4 resources, respectively, unless P=NP. All our results can be extended to the Santa Claus variants of the problems where the goal is to maximize the minimal processing time any machine receives. This chapter is based on [111].

CHAPTER 5. Similar to the previous chapter, we consider special types of assignment restrictions. We introduce a graph framework based on the restrictions:

In the primal graph, the jobs are the nodes and are adjacent if they share an eligible machine. In the dual graph, on the other hand, the machines are the nodes and two machines are adjacent if there is a job that is eligible on both of them. Lastly, the incidence graph is a bipartite graph that includes both jobs and machines as nodes, and a job node is adjacent to a machine node if the job is eligible on the machine. We study cases in which these graphs have certain structural properties.

We show that the variant of restricted assignment where the incidence graph has a constant rank- or cliquewidth admits a PTAS. This generalizes and unifies several known PTAS results.

Furthermore, we consider the treewidth for each of the three graphs. We show that a constant treewidth for the dual or incidence graph implies the existence of an FPTAS. Furthermore, we design so called fixed parameter tractable (FPT) algorithms, that is, exact algorithms with a running time of the form $f(c)\text{poly}(|I|)$ where c is a parameter of the instance I, f some computable function, and $|I|$ the input length. We present FPT results for each of the three graph. For instance, we show that restricted assignment is FPT with respect to the treewidth of the primal graph. All results concerning the treewidth also work for the more general case of unrelated scheduling with restrictions. This chapter is based on [82].

CHAPTER 6. Integer linear programs of configurations, or configuration IPs, are a classical tool in the design of algorithms for scheduling and packing problems where a set of items has to be placed in multiple target locations. Herein, a configuration describes a possible placement on one of the target locations, and the IP is used to choose suitable configurations covering the items. We present an augmented IP formulation, which we call the module configuration IP. It can be described within the framework of n-fold integer programming and, therefore, be solved efficiently. As an application, we consider variants of machine scheduling with setup times. For instance, we investigate the case that jobs can be split and scheduled on multiple machines. However, before a part of a job can be processed, an uninterrupted setup depending on the job has to be paid. For both of the variants that jobs can be executed in parallel or not, we obtain an efficient polynomial time approximation scheme (EPTAS) of running time $f(1/\varepsilon)\text{poly}(|I|)$ with a single exponential term in f for the first and a double exponential one for the second case. Previously, only constant factor approximations of $5/3$ and $4/3+\varepsilon$, respectively, were known. Furthermore, we present an EPTAS for a problem where classes of (non-splittable) jobs are given, and a setup has to be paid for each

class of jobs being executed on one machine. This chapter is based on [86].

CHAPTER 7. In this chapter, we consider the variant of uniform scheduling in which the jobs are partitioned into classes and class depended setup has to be paid for each class present on a machine. This is the uniform version of the non-splittable problem considered in the last chapter. We design a PTAS for this problem. This chapter is based on parts of [79].

CHAPTER 8. We consider two related variants of machine scheduling: single resource constrained scheduling on identical parallel machines and a generalization with resource dependent processing times. In both problems, jobs require a certain amount of an additional resource and have to be scheduled on machines minimizing the makespan, while at every point in time a given resource capacity is not exceeded. In the first variant of the problem, the processing times and resource amounts are fixed, while, in the second, the former depends on the latter.

Both problems contain the problem of bin packing with cardinality constraints as a special case, and, therefore, these problems are strongly NP-complete even for a constant number of machines larger than three, which can be proven by a reduction from 3-Partition. Furthermore, if the number of machines is part of the input, we cannot hope for an approximation algorithm with absolute ratio smaller than $3/2$.

We present asymptotic fully polynomial time approximation schemes (AFPTAS) for the problems: For any $\varepsilon > 0$ a schedule of length at most $(1+\varepsilon)$ times the optimum plus an additive term of $\mathcal{O}(p_{\max} \log(1/\varepsilon)/\varepsilon)$ is provided, and the running time is polynomially bounded in $1/\varepsilon$ and the input length. Up to now, only approximation algorithms with absolute approximation ratios were known. This chapter is based on [81].

2 PRELIMINARIES

In this chapter, we present some basic concepts and notation and provide an example approximation scheme for machine scheduling introducing some of the standard techniques used throughout this work. However, there are many concepts and techniques widely used in the field, like e.g., integer and dynamic programming, that we apply with little or no further explanation. For introductions to these concepts and general background information, we refer to the textbooks by Williamson and Shmoys [145] concerning approximation algorithms, by Papadimitriou and Steiglitz [126] concerning combinatorial optimization, and the one by Arora and Barak [8] concerning complexity theory. As a general introduction to the theory of computation, we would also like to highlight the beautiful textbook by Moore and Mertens [118].

2.1 BASIC CONCEPTS AND NOTATION

PROBLEMS AND INSTANCES. All the problems considered in this work are variants of machine scheduling, and definitions of some of the most important ones have already been provided in the last chapter. Furthermore, for the sake of readability, each chapter includes the definitions of the problems studied therein. In each context, $\mathcal{J}$ denotes the set of jobs, $n = |\mathcal{J}|$ the number of jobs, $\mathcal{M}$ the set of machines, and $m = |\mathcal{M}|$ the number of machines. Sometimes we assume $\mathcal{M} = [m]$ and adjust the notation accordingly. Moreover, we usually assume in each context that some instance denoted as I is given and we denote the encoding length of the instance as $|I|$.

APPROXIMATION SCHEMES. As stated in the last chapter, a polynomial time approximation scheme (PTAS) is a family of algorithms that includes a $(1+\varepsilon)$-approximation for each $\varepsilon > 0$; an efficient PTAS (EPTAS) is a PTAS with a running time of the form $f(1/\varepsilon)\mathrm{poly}(|I|)$, where f is some computable function, and $\mathrm{poly}(\cdot)$ some polynomial function; and an EPTAS is called fully polynomial (FPTAS) if the function f is a polynomial as well.

Throughout this work $\varepsilon > 0$ denotes the accuracy parameter of some approximation scheme. However, we usually provide an $(1+c\varepsilon)$-approximation for some constant c instead of an $(1+\varepsilon)$-approximation. We do so for the sake of simple presentation. Note that we may simply apply the described procedures with a modified parameter $\varepsilon' = \varepsilon/c$. This does not effect any of the running times claimed in this work due

to the $\mathcal{O}$-notation. Moreover, we sometimes require $\varepsilon \leqslant c$ for some constant c. Again, this is not a problem, since we may simply use the $(1+c)$-approximation for the case that $\varepsilon > c$. Lastly, in some instances, we assume that $1/\varepsilon$ is an integer. Taking the two considerations above into account, this can easily be realized: On the one hand, we may assume $\varepsilon \leqslant 1$, and, on the other, each number lies between two succeeding powers of 2.

FIXED-PARAMETER TRACTABILITY. Some of the topics studied in this work have close connections to the field of *parameterized complexity* and *fixed-parameter tractable* (FPT) algorithms. Formally, fixed-parameter tractability is introduced for variants of decision problems in which an additional parameter $k \in \mathbb{N}$ is given as part of the input. The problem is included in the complexity class FPT if there is an algorithm that solves the problem and has a running time of the form $f(k)\text{poly}(|I|)$, where f is some computable function. This definition can easily be extended to multiple parameters, e.g., by summing them up. For more background information on this subject, we refer to the book by Cygan et al. [37].

We also talk about FPT algorithms in the context of optimization problems and usually mean that the problem can be optimally solved by an algorithm with FPT running time $f(k)\text{poly}(|I|)$, where k is some parameter that is part of the input. Furthermore, note that an EPTAS may be considered as an algorithm with FPT running time for the parameter $1/\varepsilon$. For a discussion of connections between parameterized complexity and approximation algorithms, we refer to the work by Marx [113].

MISCELLANEOUS. For any integer n, we denote the set $\{1, \dots, n\}$ by $[n]$, we write $\log(\cdot)$ for the logarithm with basis 2, and we use $\text{poly}(n)$ and $\text{polylog}(n)$ as synonyms for $\mathcal{O}(n)^{\mathcal{O}(1)}$ and $\mathcal{O}(\log(n))^{\mathcal{O}(1)}$, respectively. Furthermore, for any two sets X, Y, we write Y^X for the set of functions $f : X \to Y$. If X is finite, we say that Y is indexed by X and sometimes denote the function value of f for the argument $x \in X$ by f_x. We may also say that f is a vector indexed by X. If Y is an additive group that is clear from the context, $f + f'$ for two functions $f, f' \in Y^X$ denotes the component-wise addition of the two functions, that is, $f + f' : X \to Y$ with $x \mapsto f(x) + f'(x)$. When considering the union of two disjoint sets A and B we sometimes use the disjoint union notation $A \dot\cup B$ which emphasizes the fact that the sets are disjoint. Lastly, for two functions $f : A \to C$ and $g : B \to C$ (with A and B disjoint) $f \cup g$ denotes the function that maps elements of $a \in A$ to $f(a)$ and elements $b \in B$ to $g(b)$. This is consistent with the definition of functions as sets of pairs.

2.2 EXAMPLE APPROXIMATION SCHEME

We present an approximation scheme for machine scheduling as a first example. The scheme can be seen as a variation of the works by Hochbaum and Shmoys [65] and Alon et al. [5] and does not contain any new ideas.

DUAL APPROXIMATION. Consider the *dual approximation* framework introduced by Hochbaum and Shmoys [65]: Instead of solving the minimization version of a problem directly, it is often sufficient to find a procedure that for a given bound T on the objective value either correctly reports that there is no solution with value T, or returns a solution with value at most $(1 + c\varepsilon)T$ for some constant c. If we have some initial upper bound B for the optimal makespan OPT with $B \leqslant \beta \operatorname{OPT}$ for some factor β, we can define a PTAS by trying different values T from the interval $[B/\beta, B]$ in a binary search fashion, and find a value $T^* \leqslant (1 + \mathcal{O}(\varepsilon)) \operatorname{OPT}$ after $\mathcal{O}(\log \beta/\varepsilon)$ iterations. This it sufficient as long as $\log \beta$ is bounded by a polynomial in the input length.

For nearly all of the problems considered in this work a constant factor approximation algorithm is known providing us with a suitable B and constant β. Moreover, it is usually easy for the types of problems considered in this work to find trivial m-approximation, e.g., by scheduling all the jobs in sequence on one machine.

For machine scheduling, we may do the latter or use one of the simple heuristics HEU and HEU* presented in the last chapter. Hence, we assume from now on that a target makespan T is given, and, furthermore, that all the processing times are upper bounded by T, because otherwise we can reject T immediately. Furthermore, we assume that $1/\varepsilon \in \mathbb{Z}_{>0}$.

ROUNDING AND SCALING. Next, we simplify the processing times of the jobs. We call a job j big if it has a size of at least εT, i.e., $p_j \geqslant \varepsilon T$. Otherwise, j is called small.

We perform an *arithmetic rounding* step: The sizes of the big jobs are rounded up to the next integer multiple of $\varepsilon^2 T$, that is, $p'_j = \lceil p_j/\varepsilon^2 T \rceil \varepsilon^2 T$ for each big job j. For the resulting instance I', we have:

- There is only a constant number of big job sizes. More precisely, we have $|\{p'_j \mid j \in \mathcal{J}, p_j \geqslant \varepsilon T\}| = \mathcal{O}(1/\varepsilon^2)$.
- If there is a schedule with makespan at most T for I, then there is also schedule with makespan at most $T' := (1 + \varepsilon)T$ for I'.

For the second property, note that each job size was increased at most by a factor of $(1 + \varepsilon)$.

Another classic rounding strategy called *geometric rounding* leads to similar results. In this approach, the sizes are rounded up to the next

integer power of $(1+\varepsilon)$, that is, $p'_j = (1+\varepsilon)^{\lceil p_j/(1+\varepsilon)\rceil}$ for each big job j. This yields $\mathcal{O}(\log_{1+\varepsilon}(1/\varepsilon))$ distinct processing times for the big jobs. However, an advantage of arithmetic rounding is that the instance can easily be scaled by $(\varepsilon^2 T)^{-1}$ to get an instance in which all the big processing times are integers. Hence, we assume $\varepsilon^2 T = 1$ in the following. This implies that $T = 1/\varepsilon^2$ and $T' = (1+\varepsilon)T = 1/\varepsilon^2 + 1/\varepsilon$ are integers as well since we assumed $1/\varepsilon$ to be an integer.

DEALING WITH SMALL JOBS. In the case of machine scheduling, it is very easy to deal with the small jobs. One strategy is to simply remove the small jobs and search for a schedule with makespan at most T' for the resulting instance. If there is no such schedule, there neither is one for I'; and if we have such a schedule, we can insert the small jobs via HEU, that is, assign them one after another in arbitrary order to a least loaded machine. Then either the resulting schedule has a makespan of at most $T' + \varepsilon T = (1+2\varepsilon)T$, or each machine has load of more than T' and there cannot be a schedule with makespan at most T' for I'. Hence, we assume in the following that there are no small jobs present in I'.

SOLVING THE SIMPLIFIED INSTANCE. We present two approaches to find a schedule with makespan at most T' for I'. The first is based on dynamic programming and yields a PTAS, and the second is based on integer programming and yields an EPTAS.

Let $P = \{p'_j \mid j \in \mathcal{J}\}$ be the set of (big) job sizes, and n_p be the number of jobs with size p for each $p \in P$. Note that in a given schedule we could permute jobs with equal size and hence we can focus on the job sizes placed on each machine rather than the actual jobs. Let $\Xi = \{0, 1, \ldots, n\}^P$ be a set of multiplicity vectors for job sizes and $\Lambda(\xi) = \sum_{p\in P} \xi_p p$ be the size of a vector $\xi \in \Xi$. Each such vector corresponds to a selection of job or job sizes and $\Lambda(\xi)$ to their summed up size. Let $\mathcal{C} = \{\kappa \in \Xi \mid \Lambda(\kappa) \leqslant T'\}$ be the set of vectors with size at most T' called the set of *configurations*. A configuration $\kappa \in \mathcal{C}$ corresponds to a suitable placement of jobs on a single machine. Let $\xi^* \in \Xi$ be the vector that corresponds to the jobs in the instance, that is, $\xi^*_p = n_p$ for each $p \in P$. It is easy to see that any schedule for I' with makespan T' can be translated into a selection of m configurations that cover all jobs, that is, sum up to ξ^*, and vice-versa.

For the dynamic program, we first define a layered graph with $m+1$ layers. In the first layer (layer 0), there is only one node α, and, for each $i \in [m]$, layer i contains one node $v(i, \xi)$ for each $\xi \in \Xi$. Edges exclusively go from one layer to the next. The node α is connected to the nodes $v(1, \kappa)$ with $\kappa \in \mathcal{C}$, and for each $i \in [m-1]$ and $\xi \in \Xi$, the node $v(i, \xi)$ is connected to each node $v(i+1, \xi')$ with $\xi' - \xi \in \mathcal{C}$ and $\xi' \in \Xi$. Any path with length ℓ corresponds to a selection of ℓ configurations and a path from α to $v(m, \xi^*)$, in particular, to a

selection of m configurations that sum up to ξ^*. Moreover, if such a selection exists there also exists such a path. Hence, we essentially have to solve a reachability problem in a graph with $(n+1)^{\mathcal{O}(1/\varepsilon^2)}m$ nodes. This can be done in polynomial time in the input length, but $1/\varepsilon$ occurs in the exponent and hence we get a PTAS but no EPTAS.

The second approach is to solve the so called configuration ILP (integer linear program). In this ILP we introduce a variable $x_\kappa \in \mathbb{Z}_{\geqslant 0}$ for each configuration $\kappa \in \mathcal{C}$ and search for a solution that satisfies the following constraints:

$$\sum_{\kappa \in \mathcal{C}} x_\kappa = m \tag{2.1}$$

$$\sum_{\kappa \in \mathcal{C}} \kappa_p x_\kappa = n_p \qquad \forall p \in P \tag{2.2}$$

This ILP selects m configurations due to the Constraint (2.1), and these configurations cover all the jobs due to Constraint (2.2). We can solve it using the following classical result by Lenstra [88] and Kannan [89]:

THEOREM 2.1: An integer linear program with d integral variables and encoding size s can be solved in time $d^{\mathcal{O}(d)}\text{poly}(s)$.

The above ILP has $|\mathcal{C}|$ many variables, and we have:

$$|\mathcal{C}| = \mathcal{O}(1/\varepsilon^2)^{\mathcal{O}(1/\varepsilon)} = 2^{\mathcal{O}(1/\varepsilon \log(1/\varepsilon))}$$

Note that we have $\mathcal{O}(1/\varepsilon^2)$ many constraints and all the occurring numbers are polynomially bounded in the input length and $1/\varepsilon$. Hence, using this approach yields an EPTAS.

3

UNRELATED SCHEDULING WITH FEW TYPES

3.1 INTRODUCTION

We consider the problem of unrelated scheduling in which a set $\mathcal{J}$ of n jobs has to be assigned to a set $\mathcal{M}$ of m machines. Each job j has a processing time p_{ij} for each machine i, and the goal is to find a schedule $\sigma : \mathcal{J} \to \mathcal{M}$ minimizing the *makespan* $C_{\max}(\sigma) = \max_{i \in \mathcal{M}} \sum_{j \in \sigma^{-1}(i)} p_{ij}$. In particular, we study the special case where there is only a constant number K of *machine types*. Two machines i and i' have the same type if $p_{ij} = p_{i'j}$ holds for each job j. In many application scenarios this setting is plausible, e.g., when considering computers which typically only have a very limited number of different types of processing units. We denote the processing time of a job j on a machine of type $t \in [K]$ by p_{tj} and assume that the input consists of the corresponding $K \times n$ processing time matrix together with machine multiplicities m_t for each type t yielding $m = \sum_{t \in [K]} m_t$. Note that the case $K = 1$ is equivalent to the classical machine scheduling.

For unrelated scheduling, there is a 2-approximation due to Lenstra, Shmoys and Tardos [108] who also showed that there is no better than 1.5-approximation for this problem unless P=NP. For machine scheduling, on the other hand, an EPTAS is known [5, 73]. The case with a constant number of types was first considered by Bonifaci and Wiese [20] and in a closely related setting by Bonifaci, Wiese and Baruah [144]. They presented a PTAS for both cases. Furthermore, Gehrke et al. [52] presented a PTAS with an improved running time of $\mathcal{O}(Kn) + m^{\mathcal{O}(K/\varepsilon^2)}(\log(m)/\varepsilon)^{\mathcal{O}(K^2)}$. We also study three other variants of the problem:

SANTA CLAUS PROBLEM. We also consider the reverse objective of maximizing the minimum machine load, that is:

$$C_{\min}(\sigma) = \min_{i \in \mathcal{M}} \sum_{j \in \sigma^{-1}(i)} p_{ij}$$

This problem is known as max-min fair allocation or the Santa Claus problem. The intuition behind these names is that the jobs are interpreted as goods (e.g. presents), the machines as players (e.g. children), and the processing times as the values of the goods from the perspective of the different players. Finding an assignment that maximizes the minimum machine load means, therefore, finding an allocation of the goods that is in some sense fair (making the least happy kid as happy as possible). We will refer to the problem as Santa Claus problem, in the following, but otherwise stick to the scheduling terminology.

For the Santa Claus problem, no constant approximation algorithm is known. The best rate so far is $\mathcal{O}(n^{\varepsilon})$ and due to Bateni et al. [13] and Chakrabarty et al. [23] with a running time of $\mathcal{O}(n^{1/\varepsilon})$ for any $\varepsilon > 0$.

UNIFORM TYPES. Two machines i and i' have the same *uniform* machine type if there is a scaling factor s such that $p_{ij} = sp_{i'j}$ for each job j. While jobs behave on machines of the same type as they do on identical machines, they behave of machines of the same uniform type like they do on uniformly related machines. Hence, we may assume that the input consists of job sizes p_{tj} depending on the job j and the uniform type t, together with uniform machine types t_i, and machine speeds s_i such that $p_{ij} = p_{t_ij}/s_i$. Note that $K = 1$ is equivalent to uniform scheduling in this case.

To the best of our knowledge,this case has not been studied before, but we argue that it is a natural extension of the case with a constant number of regular machine types and also a sensible special case of the general unrelated scheduling. For the special case of uniform scheduling, an EPTAS due to Jansen [72] is known.

VECTOR SCHEDULING. In the D-dimensional vector scheduling variant of unrelated scheduling, for each job j and machine i a processing time vector $p_{ij} = (p_{ij}^{(1)}, \dots, p_{ij}^{(D)})$ is given, and the makespan of a schedule σ is defined as the maximum load any machine receives in any dimension:

$$C_{\max}(\sigma) = \max_{i \in \mathcal{M}} \left\| \sum_{j \in \sigma^{-1}(i)} p_{ij} \right\|_{\infty} = \max_{i \in \mathcal{M}, d \in [D]} \sum_{j \in \sigma^{-1}(i)} p_{ij}^{(d)}$$

Machine types are defined correspondingly. We consider the case that both K and D are constant and, as in the one-dimensional case, we may assume that the input consist of processing time vectors depending on types and jobs and machine multiplicities.

A PTAS for this problem has been presented in the work by Bonifaci and Wiese [20]. Before, the vector scheduling problem has been studied for the special case of identical machines by Chekuri and Khanna [26]. They achieved a PTAS for the case that D is constant and an $\mathcal{O}(\log^2 D)$-approximation for the case that D is arbitrary.

RESULTS AND METHODOLOGY. The main result of this chapter is the following:

THEOREM 3.1: There is an EPTAS for both scheduling on unrelated parallel machines and the Santa Claus problem with a constant number K of different machine types with running time $2^{\mathcal{O}(K \log(K) 1/\varepsilon \log^4 1/\varepsilon)} + \text{poly}(|I|)$.

First, we present a basic version of the EPTAS for unrelated scheduling with a running time doubly exponential in $1/\varepsilon$. For this EPTAS

we use the dual approximation approach by Hochbaum and Shmoys [65] to get a guess T of the optimal makespan OPT. Then, we further simplify the problem via geometric rounding of the processing times. Next, we formulate a mixed integer linear program (MILP) based on the classical configuration ILP with a constant number of integral variables that encodes a relaxed version of the problem. We solve it with the algorithm by Lenstra and Kannan [88, 89]. The fractional variables of the MILP have to be rounded and we achieve this with a properly designed flow network utilizing flow integrality and causing only a small error. With an additional error, the obtained solution can be used to construct a schedule with makespan $(1 + \mathcal{O}(\varepsilon))T$. This procedure is described in detail in Section 3.2. Building upon the basic EPTAS we achieve the improved running time using techniques by Jansen [72] and by Jansen et al. [73]. The basic idea of these techniques is to make use of results concerning the existence of simple structured solutions of integer linear programs (ILPs). In particular, these results can be used to guess the non-zero variables of the MILP because they sufficiently limit the search space. We show how these techniques can be applied in our case in Section 3.3. Furthermore, we present efficient approximation schemes for several other problem variants thereby demonstrating the flexibility of our approach. In particular, we can adapt all our techniques to the Santa Claus problem yielding the result stated above. This is covered in Section 3.4 and in Section 3.5 we show:

THEOREM 3.2: There is an EPTAS for scheduling on unrelated parallel machines with a constant number K of different uniform machine types with running time $2^{\mathcal{O}(K \log(K) 1/\varepsilon^3 \log^5 1/\varepsilon)} + \text{poly}(|I|)$.

We achieve this with a non-trivial combination of the ideas of Section 3.2 with techniques for scheduling on uniformly related machines by Jansen [72]. Finally, in Section 3.6, we revisit the unrelated vector scheduling problem that was studied by Bonifaci and Wiese [20]. We show that an additional rounding step—similar to the one in [26]—together with a slight modification of the MILP and the rounding procedure yield an EPTAS for this problem as well.

THEOREM 3.3: There is an EPTAS for vector scheduling on unrelated parallel machines with constant dimension D and a constant number K of different machine types.

Note that our results may also be seen as fixed parameter tractable algorithms (see Section 2.1) for the parameters $1/\varepsilon$ and K (and D). In the last section, we elaborate on possible directions for future research.

FURTHER RELATED WORK. It is well known that the unrelated scheduling problem admits an FPTAS in the case that the number of machines is considered constant [67], and we already mentioned the seminal work by Lenstra et al. [108]. Furthermore, the problem of unrelated scheduling with a constant number of machine types is

strongly NP-hard because it is a generalization of the strongly NP-hard problem of machine scheduling, where the execution times of the jobs do not depend on the machines. Therefore, an FPTAS cannot be hoped for, but there is a classical PTAS result due to Hochbaum and Shmoys [65] for this case. The same authors also provided the first PTAS for scheduling on uniform parallel machines [66], where each job i has a size p_j, each machine i has a speed s_i, and we have $p_{ij} = p_j/s_i$. For both of these problems, there are EPTAS results due to Alon et al. [5] for the identical and due to Jansen [72] for the uniform case. The EPTAS for scheduling on identical parallel machines with the best asymptotic running time so far was presented by Jansen, Klein and Verschae [73]. They achieve a running time of $2^{\mathcal{O}(1/\varepsilon \log^4 1/\varepsilon)} + \text{poly}(n)$. On the other hand, Chen, Ye and Zhang [30] showed that we cannot hope for an EPTAS with a sub-linear dependency in $1/\varepsilon$ in the exponent, unless the exponential time hypothesis (see [69]) fails. All the above EPTAS results employ some version of the classical *configuration ILP (integer linear program)*, which was originally introduced by Gilmore and Gomory [55] in the context of the closely related bin packing problem.

For unrelated scheduling with a constant number of machine types, the case $K = 2$ has been studied: Imreh [70] designed heuristic algorithms with rates $2 + (m_1 - 1)/m_2$ and $4 - 2/m_1$, and Bleuse et al. [17] presented an algorithm with rate $4/3 + 3/m_2$ and moreover a (faster) $3/2$-approximation for the case that for each job the processing time on the second machine type is at most the one on the first. Moreover, Raravi and Nélis [128] designed a PTAS for the case with two machine types. In 2018, Kones and Levin [103] presented an EPTAS for a problem that generalizes many of the problems considered in this chapter and the setting with uniform types in particular.

Interestingly, unrelated scheduling is in P if both the number of machine types and the number of job types is bounded by a constant. This is implied by a result due to Chen et al. [31] building upon a result by Goemans and Rothvoss [56]. Job types are defined analogously to machine types, i.e., two jobs j, j' have the same type, if $p_{ij} = p_{ij'}$ for each machine i. In this case the matrix (p_{ij}) has only a constant number of distinct rows and columns. Note that both the number of machine types and uniform machine types bounds the rank of this matrix. However, the case of unrelated scheduling where the matrix (p_{ij}) has constant rank turns out to be much harder: Already for the case with rank 3, the problem is APX-hard [31], and for rank 4 an approximation algorithm with rate smaller than $3/2$ can be ruled out unless P=NP [30]. In a rather recent work, Knop and Koutecký [100] considered the number of machine types as a parameter from the perspective of fixed parameter tractability. They showed that unrelated scheduling is fixed parameter tractable for the parameters K and $\max p_{i,j}$. Chen et al. [31] extended this, showing that unrelated

scheduling is fixed parameter tractable for the parameters $\max p_{i,j}$ and the rank of the processing time matrix.

For the case that the number of machines is constant, the Santa Claus problem behaves similar to the unrelated scheduling problem: there is an FPTAS that is implied by a result due to Woeginger [147]. In the general case however, so far no approximation algorithm with a constant approximation guarantee has been found. The results by Lenstra et al. [108] can be adapted to show that there is no approximation algorithm with a rate smaller than 2, unless P=NP, and to get an algorithm that finds a solution with value at least $\text{OPT}(I) - \max p_{i,j}$, as was done by Bezáková and Dani [15]. Since $\max p_{i,j}$ could be bigger than $\text{OPT}(I)$, this does not provide a (multiplicative) approximation guarantee. Bezáková and Dani also presented a simple $(n - m + 1)$-approximation, and an improved approximation guarantee of $\mathcal{O}(\sqrt{n}\log^3 n)$ was achieved by Asadpour and Saberi [9].

3.2 BASIC EPTAS

In this section, we describe a basic EPTAS for unrelated scheduling with a constant number of machine types with a running time doubly exponential in $1/\varepsilon$. W.l.o.g. we assume $\varepsilon < 1$. Furthermore, $\log(\cdot)$ denotes the logarithm to the base 2 and for $k \in \mathbb{Z}_{\geqslant 0}$ we write $[k]$ for $\{1, \ldots, k\}$.

First, we simplify the problem via the classical dual approximation concept by Hochbaum and Shmoys [65] (see Section 2.2), and therefore assume that a target makespan T is given. Note that a constant approximation for unrelated scheduling is known and hence the additional effort due to the binary search is only a constant factor. Next, we present a brief overview of the algorithm for the simplified problem followed by a detailed description and analysis.

ALGORITHM 3.4.

1. Simplify the input via geometric rounding with an error of εT.
2. Build the mixed integer linear program MILP($\bar{T}$), and solve it with the algorithm by Lenstra and Kannan ($\bar{T} = (1 + \varepsilon)T$).
3. If there is no solution, report that there is no solution with makespan T.
4. Generate an integral solution for MILP($\bar{T} + \varepsilon T + \varepsilon^2 T$) via a flow network utilizing flow integrality.
5. The integral solution is turned into a schedule with an additional error of $\varepsilon^2 T$ due to the small jobs.

SIMPLIFICATION OF THE INPUT. We construct a simplified instance $\bar{I}$ with modified processing times $\bar{p}_{tj}$. If a job j has a processing

time bigger than T for a machine type $t \in [K]$, we set $\bar{p}_{tj} = \infty$. We call a job *big* (for machine type t) if $p_{tj} > \varepsilon^2 T$ and *small* otherwise. We perform a geometric rounding step for each job j with $p_{tj} < \infty$, that is, we set $\bar{p}_{tj} = (1+\varepsilon)^x \varepsilon^2 T$ with $x = \lceil \log_{1+\varepsilon}(p_{tj}/(\varepsilon^2 T)) \rceil$.

LEMMA 3.5. *If there is a schedule with makespan at most T for I, then the same schedule has makespan at most $(1+\varepsilon)T$ for instance $\bar{I}$; and any schedule for instance $\bar{I}$ can be turned into a schedule for I without increase in the makespan.*

We will search for a schedule with makespan $\bar{T} = (1+\varepsilon)T$ for the rounded instance $\bar{I}$.

We establish some notation for the rounded instance. For any rounded processing time p, we denote the set of jobs j with $\bar{p}_{tj} = p$ by $J_t(p)$. Moreover, for each machine type t, let S_t and B_t be the sets of small and big rounded processing times. Obviously, we have $|S_t| + |B_t| \leqslant n$. Furthermore, $|B_t|$ is bounded by a constant: Let N be such that $(1+\varepsilon)^N \varepsilon^2 T$ is the biggest rounded processing time for all machine types. Then we have $(1+\varepsilon)^{N-1} \varepsilon^2 T \leqslant T$ and therefore $|B_t| \leqslant N \leqslant \log(1/\varepsilon^2)/\log(1+\varepsilon) + 1 \leqslant 1/\varepsilon \log(1/\varepsilon^2) + 1$ (using $\varepsilon \leqslant 1$).

MILP. For any set of processing times P, we call the P-indexed vectors of non-negative integers $\mathbb{Z}^P_{\geqslant 0}$ *configurations* (for P). The *size* $\Lambda(C)$ of configuration C is given by $\sum_{p \in P} C_p p$. For each $t \in [K]$, we consider the set $\mathcal{C}_t(\bar{T})$ of configurations C for the big processing times B_t and with $\Lambda(C) \leqslant \bar{T}$. Given a schedule σ, we say that a machine i of type t obeys a configuration C if the number of big jobs with processing time p that σ assigns to i is exactly C_p for each $p \in B_t$. Since the processing times in B_t are bigger than $\varepsilon^2 T$, we have $\sum_{p \in B_t} C_p \leqslant 1/\varepsilon^2$ for each $C \in \mathcal{C}_t(\bar{T})$. Therefore, the number of distinct configurations in $\mathcal{C}_t(\bar{T})$ can be bounded by:

$$\begin{aligned}(1/\varepsilon^2+1)^N &< (1/\varepsilon^2+1)^{1/\varepsilon \log(1/\varepsilon^2)+1} \\ &= 2^{\log(1/\varepsilon^2+1)\,1/\varepsilon \log(1/\varepsilon^2)+1} \in 2^{\mathcal{O}(1/\varepsilon \log^2 1/\varepsilon)}\end{aligned}$$

We define a mixed integer linear program MILP($\bar{T}$) in which configurations are chosen integrally and jobs are assigned fractionally to machine types. Note that we will call a solution of a MILP integral if both the integral and fractional variables have integral values. We introduce variables $z_{C,t} \in \mathbb{Z}_{\geqslant 0}$ for each machine type $t \in [K]$ and configuration $C \in \mathcal{C}_t(\bar{T})$, and $x_{j,t} \geqslant 0$ for each machine type $t \in [K]$ and job $j \in \mathcal{J}$. For $\bar{p}_{tj} = \infty$, we set $x_{j,t} = 0$. Besides this, the MILP has the following constraints:

$$\sum_{C \in \mathcal{C}_t(\bar{T})} z_{C,t} = m_t \qquad \forall t \in [K] \qquad (3.1)$$

$$\sum_{t \in [K]} x_{j,t} = 1 \qquad \forall j \in \mathcal{J} \qquad (3.2)$$

$$\sum_{j \in J_t(p)} x_{j,t} \leqslant \sum_{C \in \mathcal{C}_t(\bar{T})} C_p z_{C,t} \qquad \forall t \in [K], p \in B_t \qquad (3.3)$$

$$\sum_{C \in \mathcal{C}_t(\bar{T})} \Lambda(C) z_{C,t} + \sum_{p \in S_t} p \sum_{j \in J_t(p)} x_{j,t} \leqslant m_t \bar{T} \qquad \forall t \in [K] \qquad (3.4)$$

Because of Constraint (3.1), the number of chosen configurations for each machine type equals the number of machines of this type. Due to Constraint (3.2), the variables $x_{j,t}$ encode the fractional assignment of jobs to machine types. Moreover, for each machine type, it is ensured with Constraint (3.3) that the summed up number of big jobs of each size is at most the number of big jobs that are used in the chosen configurations for the respective machine type. Lastly, (3.4) guarantees that the overall processing time of the configurations and small jobs assigned to a machine type does not exceed the area $m_t \bar{T}$. It is easy to see that the MILP models a relaxed version of the problem:

LEMMA 3.6. *If there is schedule with makespan $\bar{T}$, then there is a feasible (integral) solution of* MILP($\bar{T}$)*; and if there is a feasible integral solution for* MILP($\bar{T}$)*, then there is a schedule with makespan at most $\bar{T} + \varepsilon^2 T$.*

Proof. Let σ be a schedule with makespan $\bar{T}$. Each machine of type t obeys exactly one configuration from $\mathcal{C}_t(\bar{T})$, and we set $z_{C,t}$ to be the number of machines of type t that obey C with respect to σ. Furthermore, for a job j^* let t^* be the type of machine $\sigma(j^*)$. We set $x_{j^*,t^*} = 1$ and $x_{j^*,t} = 0$ for $t \neq t^*$. It is easy to check that all conditions are fulfilled.

Now, let $(z_{C,t}, x_{j,t})$ be an integral solution of MILP($\bar{T}$). Using (3.2), we can assign the jobs to distinct machine types based on the x variables. The z variables can be used to assign configurations to machines such that each machine receives exactly one configuration (utilizing (3.1)). Based on these configurations, we can create slots for the big jobs, and, for each type t, we can successively assign all of the big jobs assigned to this type to slots of the size of their processing time because of (3.3). Now, for each type, we can iterate through the machines and greedily assign small jobs. When the makespan $\bar{T}$ is exceeded due to some job, we stop assigning to the current machine and continue with the next. Because of (3.4), all small jobs can be assigned in this fashion. Since the small jobs have size at most $\varepsilon^2 T$, we get a schedule with makespan at most $\bar{T} + \varepsilon^2 T$. □

We have $K 2^{\mathcal{O}(1/\varepsilon \log^2 1/\varepsilon)}$ integral variables, i.e., a constant number. Therefore, MILP($\bar{T}$) can be solved in polynomial time with the following classical result due to Lenstra [88] and Kannan [89]:

THEOREM 3.7: A mixed integer linear program with d integral variables and encoding size s can be solved in time $d^{\mathcal{O}(d)} \mathrm{poly}(s)$.

ROUNDING. In this paragraph, we describe how a feasible solution $(z_{C,t}, x_{j,t})$ for MILP($\bar{T}$) can be transformed into an integral feasible

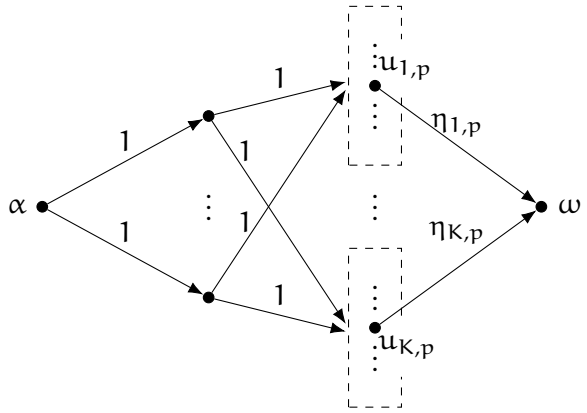


Figure 3.1: The flow network used for the rounding of the fractional variables.

solution $(\bar{z}_{C,t}, \bar{x}_{j,t})$ for MILP$(\bar{T} + \varepsilon T + \varepsilon^2 T)$, where the second MILP is defined using the same configurations but properly changed right hand side. This is achieved via a flow network utilizing flow integrality.

For any (small or big) processing time p, let $\eta_{t,p} = \lceil \sum_{j \in J_t(p)} x_{j,t} \rceil$ be the rounded up (fractional) number of jobs with processing time p that are assigned to machine type t. Note that for big job sizes $p \in B_t$, we have $\eta_{t,p} \leqslant \sum_{C \in \mathcal{C}_t(\bar{T})} C_p z_{C,t}$ because of (3.3) and because the right hand side is an integer.

Now, we describe the flow network $G = (V, E)$ with source α and sink ω. For each job $j \in \mathcal{J}$, there is a job node v_j and an edge (α, v_j) with capacity 1 connecting the source and the job node. Moreover, for each machine type t and processing time $p \in B_t \cup S_t$, we have a processing time node $u_{t,p}$. The processing time nodes are connected to the sink via edges $(u_{t,p}, \omega)$ with capacity $\eta_{t,p}$. Lastly, for each job j and machine type t with $\bar{p}_{t,j} < \infty$, we have an edge $(v_j, u_{t,\bar{p}_{t,j}})$ with capacity 1 connecting the job node with the corresponding processing time nodes. We outline the construction in Figure 3.1. Obviously we have $|V| \leqslant (K+1)n + 2$ and $|E| \leqslant (2K+1)n$.

LEMMA 3.8. *G has a maximum flow f with value $|f| = n$.*

Proof. Since the outgoing edges from α have summed up capacity n, n is a trivial upper bound for the maximum flow. The solution $(z_{C,t}, x_{j,t})$ for MILP$(\bar{T})$ can be used to design a flow f with value n by setting $f((\alpha, v_j)) = 1$, $f((v_j, u_{t,\bar{p}_{t,j}})) = x_{j,t}$ and $f((u_{t,y}, \omega)) = \sum_{j \in J_t(y)} x_{j,t}$. It is easy to check that f is indeed a feasible flow with value n. □

Using the Ford-Fulkerson algorithm, an integral maximum flow f^* can be found in time $\mathcal{O}(|E||f^*|) = \mathcal{O}(Kn^2)$. Due to flow conservation, for each job j, there is exactly one machine type t^* such that $f((v_j, u_{t^*,\bar{p}_{t^*,j}})) = 1$, and we set $\bar{x}_{j,t^*} = 1$ and $\bar{x}_{j,t} = 0$ for $t \neq t^*$. Moreover, we set $\bar{z}_{C,t} = z_{C,t}$. Obviously, $(\bar{z}_{C,t}, \bar{x}_{j,t})$ satisfies (3.1) and (3.2). Furthermore, (3.3) is fulfilled because of the capacities and because

$\eta_{t,p} \leqslant \sum_{C \in \mathcal{C}_t(\bar{T})} C_p z_{C,t}$ for big job sizes p. Due to the geometric rounding and the convergence of the geometric series, we have:

$$\sum_{p \in S_t} p \leqslant \varepsilon^2 T \sum_{i=0}^{\infty} (1+\varepsilon)^{-i} = \varepsilon^2 T \frac{1+\varepsilon}{\varepsilon}$$

This together with $\sum_{j \in J_t(p)} \bar{x}_{j,t} \leqslant \eta_{t,p} < \sum_{j \in J_t(p)} x_{j,t} + 1$ yields:

$$\begin{aligned}\sum_{p \in S_t} p \sum_{j \in J_t(p)} \bar{x}_{j,t} &< \sum_{p \in S_t} p \sum_{j \in J_t(p)} x_{j,t} + \sum_{p \in S_t} p \\ &< \sum_{p \in S_t} p \sum_{j \in J_t(p)} x_{j,t} + \varepsilon^2 T \frac{1+\varepsilon}{\varepsilon}\end{aligned}$$

Hence:

$$\sum_{C \in \mathcal{C}_t(\bar{T})} \Lambda(C) \bar{z}_{C,t} + \sum_{p \in S_t} p \sum_{j \in J_t(p)} \bar{x}_{j,t} < m_t(\bar{T} + \varepsilon T + \varepsilon^2 T)$$

Therefore (3.4) is fulfilled as well.

ANALYSIS. The solution found for MILP($\bar{T}$) can be turned into an integral solution for MILP($\bar{T} + \varepsilon T + \varepsilon^2 T$). Like described in the proof of Lemma 3.6 this can easily be turned into a schedule with makespan $\bar{T} + \varepsilon T + \varepsilon^2 T + \varepsilon^2 T \leqslant (1+4\varepsilon)T$. It is easy to see that the running time of the algorithm by Lenstra and Kannan (see Theorem 3.7) dominates the overall running time. Since MILP($\bar{T}$) has $\mathcal{O}(K/\varepsilon \log 1/\varepsilon + n)$ many constraints, Kn fractional and $K2^{\mathcal{O}(1/\varepsilon \log^2 1/\varepsilon)}$ integral variables, the running time of the algorithm can be bounded by:

$$\begin{aligned}&(K2^{\mathcal{O}(1/\varepsilon \log^2 1/\varepsilon)})^{\mathcal{O}(K2^{\mathcal{O}(1/\varepsilon \log^2 1/\varepsilon)})} \operatorname{poly}((K/\varepsilon \log 1/\varepsilon)|I|) \\ = \; & 2^{K2^{\mathcal{O}(1/\varepsilon \log^2 1/\varepsilon)}} \operatorname{poly}(|I|)\end{aligned}$$

3.3 BETTER RUNNING TIME

We improve the running time of the algorithm using techniques that utilize results concerning the existence of solutions for integer linear programs (ILPs) with a certain simple structure. In a first step, we can reduce the running time to be only singly exponential in $1/\varepsilon$ with a technique by Jansen [72]. Then we further improve the running time to the one claimed in Theorem 3.1 with a result by Jansen et al.[73]. Both techniques rely upon the following result about integer cones by Eisenbrandt and Shmonin [43]:

THEOREM 3.9: Let $X \subset \mathbb{Z}^d$ be a finite set of integer vectors and let $b \in \text{int-cone}(X) = \{\sum_{x \in X} \lambda_x x \mid \lambda_x \in \mathbb{Z}_{\geqslant 0}\}$. Then there is a subset $\tilde{X} \subseteq X$ such that $b \in \text{int-cone}(\tilde{X})$ and $|\tilde{X}| \leqslant 2d \log(4dM)$, with $M = \max_{x \in X} \|x\|_\infty$.

FIRST IMPROVEMENT. For the first improvement of the running time, this theorem is used to show:

COROLLARY 3.10. MILP($\bar{T}$) *has a feasible solution where for each machine type as few as* $\mathcal{O}(1/\varepsilon \log^2 1/\varepsilon)$ *of the corresponding integer variables are non-zero.*

We get the better running time by guessing the non-zero variables and removing all the others from the MILP. The number of possibilities of choosing $\mathcal{O}(1/\varepsilon \log^2 1/\varepsilon)$ elements out of a set of $2^{\mathcal{O}(1/\varepsilon \log^2 1/\varepsilon)}$ elements can be bounded by $2^{\mathcal{O}(1/\varepsilon^2 \log^4 1/\varepsilon)}$. Considering all the machine types, we can bound the number of guesses by $2^{\mathcal{O}(K/\varepsilon^2 \log^4 1/\varepsilon)}$. The running time of the algorithm by Lenstra and Kannan (see Theorem 3.7) with $\mathcal{O}(K/\varepsilon \log^2 1/\varepsilon)$ integer variables can be bounded by:

$$\mathcal{O}(K/\varepsilon \log^2 1/\varepsilon)^{\mathcal{O}(K/\varepsilon \log^2 1/\varepsilon)} \operatorname{poly}(|I|) = 2^{\mathcal{O}(K \log(K) 1/\varepsilon \log^3 1/\varepsilon)} \operatorname{poly}(|I|)$$

This yields a running time of:

$$2^{\mathcal{O}(K \log(K) 1/\varepsilon^2 \log^4 1/\varepsilon)} \operatorname{poly}(|I|)$$

Proof of Corollary 3.10. We consider the configuration ILP for scheduling on identical machines. Let m' be a given number of machines, P be a set of processing times with multiplicities $k_p \in \mathbb{Z}_{>0}$ for each $p \in P$, and let $\mathcal{C} \subseteq \mathbb{Z}^P_{\geqslant 0}$ be some finite set of configurations for P. The configuration ILP for m', P, $k = (k_p)_{p \in P}$, and $\mathcal{C}$ is given by:

$$\sum_{C \in \mathcal{C}} C_p y_C = k_p \qquad \forall p \in P \tag{3.5}$$

$$\sum_{C \in \mathcal{C}} y_C = m' \tag{3.6}$$

$$y_C \in \mathbb{Z}_{\geqslant 0} \qquad \forall C \in \mathcal{C} \tag{3.7}$$

The default case that we will consider most of the time is that $\mathcal{C}$ is given by a target makespan T that upper bounds the size of the configurations.

Let us assume we had a feasible solution $(\tilde{z}_{C,t}, \tilde{x}_{j,t})$ for MILP($\bar{T}$). For each $t \in [K]$ and $p \in B_t$, we set $\tilde{k}_{t,p} = \sum_{C \in \mathcal{C}_t(\bar{T})} C_p \tilde{z}_{C,t}$. We fix a machine type t. By setting $y_C = \tilde{z}_{C,t}$, we get a feasible solution for the configuration ILP given by m_t, B_t, $\tilde{k}_t$ and $\mathcal{C}_t(\bar{T})$. Theorem 3.9 can be used to show the existence of a solution for the ILP with only a few non-zero variables: Let X be the set of column vectors corresponding to the left hand side of the ILP and b be the vector corresponding to the right hand side. Then $b \in \text{int-cone}(X)$ holds and Theorem 3.9 yields that there is a subset $\tilde{X}$ of X with cardinality at most $2(|B_t| + 1) \log(4(|B_t| + 1)1/\varepsilon^2) \in \mathcal{O}(1/\varepsilon \log^2 1/\varepsilon)$ and $b \in \text{int-cone}(\tilde{X})$. Therefore, there is a solution $(\check{y}_C)$ for the ILP with $\mathcal{O}(1/\varepsilon \log^2 1/\varepsilon)$ many non-zero variables. If we set $\check{z}_{C,t} = \check{y}_C$ and $\check{x}_{j,t} = \tilde{x}_{j,t}$ and perform corresponding steps for each machine type, we get a solution

$(\check{z}_{C,t}, \check{x}_{j,t})$ that obviously satisfies Constraints (3.1),(3.2) and (3.3) of MILP($\check{T}$). The last constraint is also satisfied because the number of covered big jobs of each size does not change and therefore the overall size of the configurations does not change either for each machine type. □

FURTHER IMPROVEMENT. We show how the running time can be further improved using the techniques by Jansen et al. presented in [73]. In the following, we first state a slightly generalized version of the result. Although the proof is essentially the same as in the original work [73], we do state it—mainly for the sake of self-containment, but also because it contains minor changes. Next, we show how the results can be used to get the improved running time.

THIN SOLUTIONS. Again, let m' be a given number of machines and P be a set of processing times with multiplicities $k_p \in \mathbb{Z}_{>0}$ for each $p \in P$. Furthermore, let $\check{T}$ and $\hat{T}$ be some load bounds and $\mathcal{C}(\check{T}, \hat{T})$ the set of configurations (for P) with size at least $\check{T}$ and at most $\hat{T}$. Additionally, we set $\mathcal{C}(\hat{T}) = \mathcal{C}(0, \hat{T})$. The *support* of any vector of numbers ν is the set of indices with non-zero entries, i.e., $\operatorname{supp}(\nu) = \{i \mid \nu_i \neq 0\}$. A configuration $C \in \mathcal{C}(\check{T}, \hat{T})$ is called *simple* if the size of its support is at most $\log(\hat{T} + 1)$ and *complex* otherwise. The set of simple and complex configurations is denoted as $\mathcal{C}^s(\check{T}, \hat{T})$ and $\mathcal{C}^c(\check{T}, \hat{T})$, respectively. It is easy to see that there are only few simple configurations:

Remark 3.11. $|\mathcal{C}^s(\check{T}, \hat{T})| \leqslant |\mathcal{C}^s(\hat{T})| = 2^{\mathcal{O}(\log^2(\hat{T}) + \log^2(|P|))}$

The result by Jansen et al. [73] with the slight generalization that we consider $\mathcal{C}(\check{T}, \hat{T})$ instead of $\mathcal{C}(\hat{T})$ is the following. Note that the generalization is not needed in this section but in the next one when we consider the Santa Claus problem.

THEOREM 3.12: Let the configuration ILP for m', P, k, and $\mathcal{C}(\check{T}, \hat{T})$ have a feasible solution, and let both the makespan bounds $\check{T}$ and $\hat{T}$, as well as the processing times from P be integral. Then there is a solution (y_C) for the ILP that satisfies the following conditions:

1. $\sum_{\mathcal{C}^c(\check{T}, \hat{T})} y_C \leqslant 2(|P| + 1) \log(4(|P| + 1)\hat{T})$ and $y_C \leqslant 1$ for $C \in \mathcal{C}^c(\check{T}, \hat{T})$.
2. $|\operatorname{supp}(y)| \leqslant 4(|P| + 1) \log(4(|P| + 1)\hat{T})$.

We will call solutions with the above properties *thin*. The theorem can be proved using two lemmata:

LEMMA 3.13. *Let $X \subset \mathbb{Z}^d$ be a finite set of integer vectors with $|X| > d \log(2|X|M + 1)$ where $M = \max_{x \in X} \|x\|_\infty$. Then there exist two disjoint subsets $A, B \subseteq X$ with $\sum_{x \in A} x = \sum_{x \in B} x$.*

Proof. For each $X' \subseteq X$, we have $\| \sum_{x \in X'} x \|_\infty \leqslant |X|M$ and therefore at most $(2|X|M+1)^d$ distinct vectors that can be constructed in this fashion. On the other hand, there are $2^{|X|} > (2|X|M+1)^d$ sets $X' \subseteq X$. Hence, there are two subsets $A, B \subseteq X$ with $\sum_{x \in A} x = \sum_{x \in B} x$. By excluding their intersection from both sets, we get disjoint sets with the same property concluding the proof. □

This lemma is due to Eisenbrand and Shmonin [43] and one of the main tools in the proof of Theorem 3.9. The second lemma is due to Jansen et al. [73]:

LEMMA 3.14. *Let* $C \in \mathcal{C}^c(\check{T}, \hat{T})$ *be a complex configuration. There exist two configurations* $C', C'' \in \mathcal{C}(\check{T}, \hat{T})$ *such that* $C' + C'' = 2C$, $\Lambda(C') = \Lambda(C'') = \Lambda(C)$, $\operatorname{supp}(C') \subset \operatorname{supp}(C)$, *and* $\operatorname{supp}(C'') \subset \operatorname{supp}(C)$.

Proof. For each $S \subseteq \operatorname{supp}(C)$, let C^S be the configuration given by $C^S_p = C_p$ if $p \in S$ and $C^S_p = 0$ otherwise. Note that $C^S \in \mathcal{C}(0, \hat{T})$. There are exactly $2^{|\operatorname{supp}(C)|}$ such configurations, $2^{|\operatorname{supp}(C)|} > \hat{T} + 1$ since C is complex, and we have $\Lambda(C^S) \leqslant \Lambda(C) \leqslant \hat{T}$, as well as $\Lambda(C^S) \in \{0, \dots, \hat{T}\}$. Hence, there are two distinct sets $S_1, S_2 \subseteq \operatorname{supp}(C)$ such that $\Lambda(C^{S_1}) = \Lambda(C^{S_2}) > 0$, and we may assume that these sets are disjoint because otherwise we can simply remove the intersection. Let $C' = C - C^{S_1} + C^{S_2}$ and $C'' = C + C^{S_1} - C^{S_2}$. Then we have:

- $C', C'' \in \mathcal{C}(\check{T}, \hat{T})$
- $\Lambda(C') = \Lambda(C) - \Lambda(C^{S_1}) + \Lambda(C^{S_2}) = \Lambda(C) = \Lambda(C'')$
- $2C = 2C + C^{S_1} - C^{S_1} + C^{S_2} - C^{S_2} = C' + C''$
- $\operatorname{supp}(C') = \operatorname{supp}(C) \setminus \operatorname{supp}(C^{S_1}) \subset \operatorname{supp}(C)$
- $\operatorname{supp}(C'') = \operatorname{supp}(C) \setminus \operatorname{supp}(C^{S_2}) \subset \operatorname{supp}(C)$

This completes the proof. □

Proof of Theorem 3.12. For each feasible solution y of the configuration ILP for m', P, k, and $\mathcal{C}(\check{T}, \hat{T})$, let $\Phi(y) = \sum_{C \in \mathcal{C}^c(\check{T}, \hat{T})} y_C |\operatorname{supp}(C)|$ be a potential function and let y be a feasible solution such that $\Phi(y)$ is minimal.

Assume that $y_C > 1$ for some $C \in \mathcal{C}^c(\check{T}, \hat{T})$. Let $C', C'' \in \mathcal{C}(\check{T}, \hat{T})$ be the configurations implied by Lemma 3.14 and y' be a solution for the configuration ILP with $y'_C = y_C - 2$, $y'_{C'} = y_{C'} + 1$, $y'_{C''} = y_{C''} + 1$ and $y'_{C^*} = y_{C^*}$ for any other configuration $C^* \in \mathcal{C}(\check{T}, \hat{T})$. The properties guaranteed by Lemma 3.14 yield that y' is feasible and $\Phi(y') < \Phi(y)$ contradicting the choice of y. Hence, we have $y_C \leqslant 1$ for each $C \in \mathcal{C}^c(\check{T}, \hat{T})$.

Now assume that $\sum_{\mathcal{C}^c(\check{T}, \hat{T})} y_C > 2(|P|+1)\log(4(|P|+1)\hat{T})$. Let X be the set of column vectors given by the configuration ILP and corresponding to configurations $C \in \mathcal{C}^c(\check{T}, \hat{T})$ such that $y_C > 0$ (and hence $y_C = 1$). The assumption translates to $|X| > 2(|P|+1)\log(4(|P|+$

$1)\hat{T})$ and $|P|+1$ is the dimension of the vectors contained in X. We now want to employ Lemma 3.13 and therefore have to show that $|X| > (|P|+1)\log(2|X|M+1)$. First note that $M \leqslant \hat{T}$ and $2^{|X|} > (4(|P|+1)\hat{T})^{2(|P|+1)}$ due to the assumption. Furthermore, we have:

$$\begin{aligned}(|P|+1)\log(2|X|M+1) &< (|P|+1)\log\left(2|X|2^{|X|/2(|P|+1)}/(4(|P|+1))+1\right)\\ &\leqslant |X|/2 + (|P|+1)\log\left(|X|/(2(|P|+1))+1\right)\\ &\leqslant |X|\end{aligned}$$

Hence, due to Lemma 3.13, there exist two disjoint subsets $A, B \subseteq X$ with $\sum_{x\in A} x = \sum_{x\in B} x$. Furthermore, both subsets have to be non-empty because the vectors are strictly positive. Let y^A be the vector indexed by $\mathcal{C}(\check{T}, \hat{T})$ such that $y^A_C = 1$ if the column corresponding to C is contained in A and $y^A_C = 0$ otherwise. The vector y^B is defined analogously. Then $y' = y - y^A + y^B$ and $y'' = y + y^A - y^B$ are feasible solutions for the ILP as well, and we have $\Phi(y') = \Phi(y) - \Phi(y^A) + \Phi(y^B)$ and $\Phi(y'') = \Phi(y) + \Phi(y^A) - \Phi(y^B)$. Hence, if $\Phi(y^A) \neq \Phi(y^B)$, we constructed a feasible solution with smaller potential yielding a contradiction to the choice of y. If, on the other hand, $\Phi(y^A) = \Phi(y^B)$, the solution y' has the same potential, but $y_C > 1$ for each complex configuration corresponding to a vector included in B. This is a contradiction to the considerations above. Hence, $\sum_{\mathcal{C}^c(\check{T},\hat{T})} y_C \leqslant 2(|P|+1)\log(4(|P|+1)\hat{T})$.

Lastly, we can apply Theorem 3.9 to the configuration ILP given by the simple solution as we did in the proof of Corollary 3.10. This yields $|\{C | C \in \mathcal{C}^s(\check{T}, \hat{T}), y_C > 0\}| \leqslant 2(|P|+1)\log(4(|P|+1)T')$ and hence $|\mathrm{supp}(y)| \leqslant 4(|P|+1)\log(4(|P|+1)T')$. □

BETTER RUNNING TIME. The improved running time can be obtained by determining configurations that are equivalent to the complex configurations (via guessing and dynamic programming), guessing the support of the simple configurations, and solving the MILP with few integral variables. The approach is a direct adaptation of the one in [73] for our case. In the following, we explain the additional steps of the modified algorithm in more detail, analyze the running time and present an outline of the complete algorithm.

To utilize Theorem 3.12, we have to ensure that the makespan and the processing times are integral and that the makespan is small. After the geometric rounding step, we scale the makespan and the processing times such that $T = 1/\varepsilon^3$ and $\bar{T} = (1+\varepsilon)/\varepsilon^3$ holds and the processing times have the form $(1+\varepsilon)^x\varepsilon^2 T = (1+\varepsilon)^x/\varepsilon$. Next, we apply a second rounding step for the big processing times setting $\check{p}_{t,j} = \lceil \bar{p}_{t,j} \rceil$ for $\bar{p}_{t,j} \in B_t$ and denote the set of these processing times by $\check{B}_t$. Obviously we have $|\check{B}_t| \leqslant |B_t| \leqslant 1/\varepsilon \log(1/\varepsilon^2) + 1$. We denote the corresponding instance by $\check{I}$. Since for a schedule with makespan T for instance I there are at most $1/\varepsilon^2$ big jobs on any machine, we get:

LEMMA 3.15. *If there is a schedule with makespan at most* T *for* I*, then the same schedule has makespan at most* $(1+2\varepsilon)T$ *for instance* $\check{I}$*; and any schedule for instance* $\check{I}$ *can be turned into a schedule for* I *without increase in the makespan.*

We set $\check{T} = (1+2\varepsilon)T$, and, for each machine type t, we consider the set of configurations $\mathcal{C}_t(\lfloor\check{T}\rfloor)$ for $\check{B}_t$ with size at most $\lfloor\check{T}\rfloor$ (we do not need an additional lower bound for the sizes of configurations in this section). Rounding down $\check{T}$ ensures integrality and causes no problems because all big processing times are integral. Furthermore, let $\mathcal{C}^c_t(\lfloor\check{T}\rfloor)$ and $\mathcal{C}^s_t(\lfloor\check{T}\rfloor)$ be the subsets of complex and simple configurations. Due to Remark 3.11, we have:

$$|\mathcal{C}^s_t(\lfloor\check{T}\rfloor)| \in 2^{\mathcal{O}(\log^2\lfloor\check{T}\rfloor+\log^2|\check{B}_t|)} = 2^{\mathcal{O}(\log^2 1/\varepsilon))} \tag{3.8}$$

Due to Theorem 3.12 (using the same considerations concerning configuration ILPs like in the last paragraph), there is a solution $(\check{z}_C, \check{x}_{j,t})$ for MILP($\check{T}$) (adjusted to this case) that uses for each machine type t at most $4(|\check{B}_t|+1)\log(4(|\check{B}_t|+1)\lfloor\check{T}\rfloor) \in \mathcal{O}(1/\varepsilon\log^2 1/\varepsilon)$ many configurations from $\mathcal{C}_t(\lfloor\check{T}\rfloor)$. Moreover, at most $2(|\check{B}_t|+1)\log(4(|\check{B}_t|+1)\lfloor\check{T}\rfloor) \in \mathcal{O}(1/\varepsilon\log^2 1/\varepsilon)$ complex configurations are used and each of them is used only once. Since each configuration corresponds to at most $1/\varepsilon^2$ many jobs, there are at most $\mathcal{O}(1/\varepsilon^3\log^2 1/\varepsilon)$ many jobs for each type corresponding to complex configurations. Hence, we can determine the number of complex configurations m^c_t for machine type t along with the number of jobs $k^c_{t,p}$ with processing time $p \in \check{B}_t$ that are covered by a complex configuration in $(1/\varepsilon^3\log^2 1/\varepsilon)^{\mathcal{O}(K/\varepsilon\log 1/\varepsilon)} = 2^{\mathcal{O}(K/\varepsilon\log^2 1/\varepsilon)}$ many steps via guessing. Now, we can use a dynamic program to determine configurations (with multiplicities) that are equivalent to the complex configurations in the sense that their size is bounded by $\lfloor\check{T}\rfloor$, their summed up number is m^c_t, and they cover exactly $k^c_{t,p}$ jobs with processing time p.

The dynamic program can be defined as follows. For each $i \in \{0,1,\dots,m^c_t\}$, there is a layer, and, in each layer, we compute nodes labeled with $\check{B}_t$-indexed vectors y of non-negative integers with $y_p \leqslant k^c_{t,p}$. A node with label y in layer i is supposed to encode that y_p jobs of size p can be covered by i configurations from $\mathcal{C}^c_t(\lfloor\check{T}\rfloor)$. In layer 0, there is only one node with label $(0,\dots,0)$, and for each possible y, there is a node v with this label present in layer $i > 0$ if there is a predecessor node u with label y' in layer $i-1$ and a configuration $C \in \mathcal{C}_t(\lfloor\check{T}\rfloor)$ such that $y = y' + C$. In this case, the nodes v and u are connected with an edge labeled with C. Now, a path from the node with label $(0,\dots,0)$ in layer 0 to the node with label $(k^c_{t,p})_{p\in\check{B}}$ corresponds to a proper selection of configurations. We denote the set of configurations the program computes with $\tilde{\mathcal{C}}_t$ and the multiplicities with $\tilde{z}_C$ for $C \in \tilde{\mathcal{C}}_t$. It is easy to see that the running time of such a program can be bounded by $\mathcal{O}(m^c_t(\prod_{p\in\check{B}_t}(k^c_{t,p}+1))^2)$. Using $m^c_t \in \mathcal{O}(1/\varepsilon\log^2 1/\varepsilon)$ and $k^c_{t,p} \in \mathcal{O}(1/\varepsilon^3\log^2 1/\varepsilon)$, this yields

a running time of $K2^{\mathcal{O}(1/\varepsilon \log^2 1/\varepsilon)}$ when considering all the machine types.

Having determined configurations equivalent to the complex ones, we may just guess the simple configurations. For each machine type, there are at most $2^{\mathcal{O}(\log^2 1/\varepsilon)}$ simple configurations and the number of configurations we need is bounded by $\mathcal{O}(1/\varepsilon \log^2 1/\varepsilon)$. Therefore, the number of needed guesses is bounded by $2^{\mathcal{O}(K/\varepsilon \log^4 1/\varepsilon)}$. Now we can solve a modified version of MILP($\check{T}$) in which z_C is fixed to $\tilde{z}_C$ for $C \in \tilde{\mathcal{C}}_t$ and only variables $z_{C'}$ corresponding to the guessed simple configurations are used. The running time for the algorithm by Lenstra and Kannan can again be bounded by $2^{\mathcal{O}(K \log K 1/\varepsilon \log^3 1/\varepsilon)} \text{poly}(|I|)$. Thus we get an overall running time of $2^{\mathcal{O}(K \log K 1/\varepsilon \log^4 1/\varepsilon)} \text{poly}(|I|)$. Considering the two cases $2^{\mathcal{O}(K \log K 1/\varepsilon \log^4 1/\varepsilon)} < \text{poly}(|I|)$ and $2^{\mathcal{O}(K \log K 1/\varepsilon \log^4 1/\varepsilon)} \geqslant \text{poly}(|I|)$ yields the claimed running time of:

$$2^{\mathcal{O}(K \log(K) 1/\varepsilon \log^4 1/\varepsilon)} + \text{poly}(|I|)$$

Hence, the proof of the part of Theorem 3.1 concerning unrelated scheduling is complete. We conclude this section with a summary of the complete algorithm.

ALGORITHM 3.16.

1. Simplify the input via scaling, geometric rounding and a second rounding step for the big jobs with an error of $2\varepsilon T$. We now have $T = 1/\varepsilon^3$.

2. Guess the number of machines m_t^c with a complex configuration for each machine type t along with the number $k_{t,p}^c$ of jobs with processing time p covered by complex configurations for each big processing time $p \in \check{B}_t$.

3. For each machine type t, determine via dynamic programming configurations that are equivalent to the complex configurations.

4. Guess the simple configurations used in a thin solution.

5. Build the simplified mixed integer linear program MILP($\check{T}$) in which the variables for configurations from step 3 are fixed and only integral variables for configurations guessed in step 4 are used. Solve it with the algorithm by Lenstra and Kannan.

6. If there is no solution for each of the guesses, report that there is no solution with makespan T.

7. Generate an integral solution for MILP($\check{T} + \varepsilon T + \varepsilon^2 T$) via a flow network utilizing flow integrality.

8. With an additional error of $\varepsilon^2 T$ due to the small jobs, the integral solution is turned into a schedule.

3.4 THE SANTA CLAUS PROBLEM

Adapting the result for unrelated scheduling, we achieve an EPTAS for the Santa Claus problem. It is based on the basic EPTAS together with the second running time improvement. In the following, we show the needed adjustments.

PRELIMINARIES. W.l.o.g. we present a $(1-\varepsilon)^{-1}$-approximation instead of a $(1+\varepsilon)$-approximation. Moreover, we assume $\varepsilon < 1$ and that $m \leqslant n$ because otherwise the problem is trivial.

The dual approximation method can be applied in this case as well. However, since we have no approximation algorithm with a constant rate, the binary search is slightly more expensive. Still, we can use for example the algorithm by Bezáková and Dani [15] to find a bound B for the optimal makespan with $B \leqslant \mathrm{OPT} \leqslant (n-m+1)B$. In $\mathcal{O}(\log((n-m)/\varepsilon))$ many steps we can find a guess for the optimal minimum machine load T^* such that $T^* \leqslant \mathrm{OPT} < T^* + \varepsilon B$ and therefore $T^* > (1-\varepsilon)\,\mathrm{OPT}$. It suffices to find a procedure that given an instance and a guess T outputs a solution with objective value at least $(1-\alpha\varepsilon)T$ for some constant α.

Concerning the simplification of the input, we first scale the running times and the makespan such that $T = 1/\varepsilon^3$. Then we set the processing times that are bigger than T equal to T. Next, we round the processing times down via geometric rounding: We set $\bar{p}_{t,j} = (1-\varepsilon)^x \varepsilon^2 T$ with $x = \lceil \log_{1-\varepsilon} p_{tj}/(\varepsilon^2 T) \rceil$. The number of big jobs for any machine type is again bounded by $1/\varepsilon \log(1/\varepsilon^2) \in \mathcal{O}(1/\varepsilon \log 1/\varepsilon)$. For the big jobs, we apply the second rounding step setting $\breve{p}_{t,j} = \lfloor \bar{p}_{t,j} \rfloor$ and denote the resulting big processing times with $\breve{B}_t$, the corresponding instance by $\breve{I}$, and the occurring small processing times by S_t. The analogue of Lemma 3.15 holds, i.e. at the cost of $2\varepsilon T$ we may search for a solution for the rounded instance $\breve{I}$. We set $\breve{T} = (1-2\varepsilon)T$.

MILP. In the Santa Claus problem, it makes sense to use configurations of size bigger than $\breve{T}$. On the other hand, it also makes sense to consider smaller configurations, due to the small jobs. Let $\breve{T} = \lfloor \breve{T} \rfloor$ and $\hat{T} = \breve{T} + \max\{\breve{p}_{t,j} \mid t \in [K], j \in \breve{B}_t\}$. We will show that it suffices to consider configurations with size at most $\hat{T}$, and, for each machine type t, we denote the corresponding set of configurations by $\mathcal{C}_t(\hat{T})$. Again we can bound $\mathcal{C}_t(\hat{T})$ by $2^{\mathcal{O}(1/\varepsilon \log^2 1/\varepsilon)}$. The MILP has integral variables $z_{C,t}$ for each such configuration and fractional ones like before. Constraints (3.1), (3.2), and (3.3) can be adapted directly, but Constraint (3.4) has to be changed more. For this we partition $\mathcal{C}_t(\hat{T})$ into the set $\mathcal{C}_t(\breve{T}, \hat{T})$ of big configurations with size at least $\breve{T}$ and the set $\mathcal{C}_t(\breve{T}-1)$ of small configurations with size at most $\breve{T}-1$. The big configurations correspond to machines that are already properly covered, while the machines corresponding to small configurations

need additional load due to small jobs. The MILP has the following constraints:

$$\sum_{C \in \mathcal{C}_t(\hat{T})} z_{C,t} = m_t \qquad \forall t \in [K] \quad (3.9)$$

$$\sum_{t \in [K]} x_{j,t} = 1 \qquad \forall j \in \mathcal{J} \quad (3.10)$$

$$\sum_{j \in J_t(p)} x_{j,t} \geqslant \sum_{C \in \mathcal{C}_t(\hat{T})} C_p z_{C,t} \qquad \forall t \in [K], p \in \check{B}_t \quad (3.11)$$

$$\sum_{C \in \mathcal{C}_t(\check{T}-1)} \Lambda(C) z_{C,t} + \sum_{p \in S_t} p \sum_{j \in J_t(p)} x_{j,t} \geqslant (m_t - \sum_{C \in \mathcal{C}_t(\check{T},\hat{T})} z_{C,t})\check{T} \quad \forall t \in [K] \quad (3.12)$$

We denote the resulting MILP by MILP$(\check{T}, \hat{T})$ and get the analogue of Lemma 3.6:

LEMMA 3.17. *If there is schedule with minimum machine load $\check{T}$, then there is a feasible (integral) solution of* MILP$(\check{T}, \hat{T})$*; and if there is a feasible integral solution for* MILP$(\check{T}, \hat{T})$*, then there is a schedule with minimum machine load at least $\check{T} - \varepsilon^2 T$.*

Proof. Let σ be a schedule with minimum machine load $\check{T}$. We first consider only the machines for which the received load due to big jobs is at most $\hat{T}$. These machines obey exactly one configuration from $\mathcal{C}_t(\hat{T})$, and we set the corresponding integral variables like before. The rest of the integral variables we initially set to 0. Now, consider a machine of type t that receives more than $\hat{T}$ load due to big jobs. We can successively remove a biggest job from the set of big jobs assigned to the machine until we reach a subset with summed up processing time at most $\hat{T}$ and bigger than $\check{T}$. This set corresponds to a big configuration C' and we increment the variable $z_{C',t}$. The fractional variables are set like in the unrelated scheduling case and it is easy to verify that all constraints are satisfied.

Now, let $(z_{C,t}, x_{j,t})$ be an integral solution of MILP$(\check{T}, \hat{T})$. Again we can assign the jobs to distinct machine types based on the $x_{j,t}$ variables and the configurations to machines based on the $z_{C,t}$ variables such that each machine receives at most one configuration. Based on these configurations, we can create slots for the big jobs, and, for each type t, we can successively assign big jobs until all slots are filled (utilizing Constraint (3.11)). Now we can, for each type, iterate through the machines that received small configurations and greedily assign small jobs. When the load $\check{T}$ would be exceeded due to some job, we stop assigning to the current machine (not adding the current job) and continue with the next machine. Because of (3.12), we can cover all of the machines by this. Since the small jobs have size at most $\varepsilon^2 T$, we get a schedule with makespan at least $\check{T} - \varepsilon^2 T$. There may be some remaining jobs that can be assigned arbitrarily. □

SOLVING THE MILP. To solve the MILP, we again adapt the techniques by Jansen et al. [73] which is slightly more complicated for the modified MILP. Unlike in the previous section, in order to get a thin solution that still fulfills Constraint (3.12), we have to consider big and small configurations separately for each machine type. Note that if the solution of the MILP is changed, Constraint (3.12) remains satisfied if the summed up size of the small and the summed up number of the big configurations is not changed. Given a solution $(\tilde{z}_{C,t}, \tilde{x}_{j,t})$ for the MILP and a machine type t, we set:

$$\check{m}_t = \sum_{C \in \mathcal{C}_t(\check{T}-1)} \tilde{z}_{C,t} \qquad \hat{m}_t = \sum_{C \in \mathcal{C}_t(\check{T},\hat{T})} \tilde{z}_{C,t}$$
$$\check{k}_{t,p} = \sum_{C \in \mathcal{C}_t(\check{T}-1)} C_p \tilde{z}_{C,t} \qquad \hat{k}_{t,p} = \sum_{C \in \mathcal{C}_t(\check{T},\hat{T})} C_p \tilde{z}_{C,t} \quad \forall p \in \check{B}_t$$

We get two configuration ILPs: The first is given by $\check{m}_t$, $\check{B}_t$, $\check{k}_t$ and $\mathcal{C}_t(\check{T}-1)$; and the second is given by $\hat{m}_t$, $\check{B}_t$, $\hat{k}_t$ and $\mathcal{C}_t(\check{T},\hat{T})$. We can apply Theorem 3.12 to these ILPs as we have done before changing neither the summed up size of the small configurations nor the summed up number of the big configurations. Note that, at this point, the slight generalization of Theorem 3.12 is utilized in order to deal with the second ILP. We denote the subsets of simple and complex configurations contained in $\mathcal{C}_t(\check{T}-1)$ as $\mathcal{C}_t^s(\check{T}-1)$ and $\mathcal{C}_t^c(\check{T}-1)$; and use the analogue notation for $\mathcal{C}_t(\check{T},\hat{T})$. We have:

COROLLARY 3.18. *If MILP*$(\check{T},\hat{T})$ *has a solution, then there is also a solution* $(z_{C,t}, x_{j,t})$ *such that for each machine type* t*:*

1. $\sum_{C \in \mathcal{C}_t^c(\check{T}-1)} z_{C,t} \leqslant 2(|\check{B}_t|+1)\log(4(|\check{B}_t|+1)(\check{T}-1))$
2. $\sum_{C \in \mathcal{C}_t^c(\check{T},\hat{T})} z_{C,t} \leqslant 2(|\check{B}_t|+1)\log(4(|\check{B}_t|+1)\hat{T})$
3. $z_{C,t} \leqslant 1$ *for each* $C \in \mathcal{C}_t^c(\check{T}-1) \cup \mathcal{C}_t^c(\check{T},\hat{T})$
4. $|\mathrm{supp}(z_t)| \leqslant 4(|\check{B}_t|+1)(\log(4(|\check{B}_t|+1)(\check{T}-1)) + \log(4(|\check{B}_t|+1)\hat{T}))$.

Like before, the terms above can be bounded by $\mathcal{O}(1/\varepsilon \log^2 1/\varepsilon)$. Utilizing this corollary, we can again solve the MILP rather efficiently. For this we have to guess the numbers $\check{m}_t^c$ and $\hat{m}_t^c$ of machines that are covered by small and big complex configurations, respectively. In addition, we guess the numbers of big jobs corresponding to the complex configurations. With this, we can determine suitable configurations via dynamic programming using the same approach as we did before. In the MILP, we fix the big configurations we have determined and guess the non-zero variables corresponding to the simple configurations. Although this procedure is a little bit more complicated than in the unrelated machine case, the bound for the running time remains the same.

ROUNDING. To get an integral solution of the MILP, we build a similar flow network. However, in this case $\eta_{t,p}$ is set to be the rounded *down* (fractional) number of jobs with processing time p that are assigned to machine type t, i.e., $\eta_{t,p} = \lfloor \sum_{j \in J_t(p)} x_{j,t} \rfloor$. We get $\eta_{t,p} \geqslant \sum_{C \in \mathcal{C}_t(\hat{T})} C_\ell z_{C,t}$ for big processing times p. The flow network looks basically the same with one important difference: The $(u_{t,p}, \omega)$ have a *lower bound* of $\eta_{t,p}$ and an capacity of ∞. We may introduce lower bounds of 0 for all the other edges. The analogue of Lemma 3.8 holds, that is, the flow network has a (feasible) maximum flow with value n. Given such a flow, we can build a new solution for the MILP changing the $x_{j,t}$ variables based on the flow decreasing the load due to small jobs by at most $\varepsilon T + \varepsilon^2 T$.

Flow networks with lower bounds can be solved with a two-phase approach that first finds a feasible flow and then augments the flow until a max flow is reached. The first problem can be reduced to a max flow problem without lower bounds in a flow network that is rather similar to the original one with at most two additional nodes and $\mathcal{O}(|V|)$ additional edges. Flow integrality still can be used. For details we refer to [1]. The running time again can be bounded by $\mathcal{O}(Kn^2)$. Hence, the overall running time of the algorithm is $2^{\mathcal{O}(K \log(K) 1/\varepsilon \log^4 1/\varepsilon)} + \text{poly}(|I|)$ which concludes the proof of Theorem 3.1.

3.5 UNIFORM MACHINE TYPES

We consider the problem of unrelated scheduling with a constant number K of uniform machine types. In this version of the problem, the input is as follows: Each job has a size p_{tj} for each uniform machine type t, and each machine i has a speed value s_i and a type t_i. The processing time of job j on machine i is given by $p_{ij} = p_{t_i j}/s_i$.

We present an EPTAS and it has the same basic structure as the ones presented so far. However, both the MILP and its rounding are considerably more complicated. We employ ideas developed for the EPTAS for scheduling on uniform machines by Jansen [72] and carefully combine them with the techniques presented in the previous sections. Note that, in this section, we have taken less effort to get a small running time in order to keep the presentation of the result less technical.

We set $\mathcal{M}_t = \{i \in \mathcal{M} \mid t_i = t\}$ for each $t \in [K]$ and $s_{\max}^{(t)} = \max\{s_i \mid i \in \mathcal{M}_t\}$. For the sake of simplicity, we refer to uniform machine types as machine types or just types.

PRELIMINARIES. Again, we may assume that a target makespan T for instance I is given, and we employ geometric rounding to both the job sizes and machine speeds. More precisely, if a job j has a size bigger than $Ts_{\max}^{(t)}$ for a machine type $t \in [K]$, we set $\bar{p}_{tj} = \infty$.

For each job j with $p_{tj} < \infty$, we set $\bar{p}_{tj} = (1+\varepsilon)^x \varepsilon^2 T s^{(t)}_{max}$ with $x = \lceil \log_{1+\varepsilon}(p_{tj}/(\varepsilon^2 T s^{(t)}_{max})) \rceil$. Moreover, we set $\bar{s}_i = s^{(t)}_{max}/(1+\varepsilon)^y$ with $y = \lceil \log_{1+\varepsilon}(s^{(t)}_{max}/s_i) \rceil$ and call the rounded instance $\bar{I}$.

LEMMA 3.19. *If there is a schedule with makespan at most T for I, then the same schedule has makespan at most $(1+\varepsilon)^2 T$ for instance $\bar{I}$; and any schedule for instance $\bar{I}$ can be turned into a schedule for I without increase in the makespan.*

Therefore, it suffices to search for a schedule for instance $\bar{I}$ with makespan $\bar{T} := (1+\varepsilon)^2 T$. For the sake of simplicity, we do not use the $(\bar{\cdot})$-notation in the following, i.e., we assume that the instance is already rounded and the makespan properly increased.

We fix some notation: A job size p is called *huge* for a speed s if $p > Ts$, *big* if $p \leqslant Ts$ and $p > \varepsilon^2 Ts$, and *small* otherwise. We will not consider assigning a job j on a machine i of type t if p_{tj} is huge for ν_i. For each machine type t, we denote the set of occurring speeds $\{s_i \mid i \in \mathcal{M}_t\}$ by V_t, the set of machines of type t and speed s by $\mathcal{M}_{t,s}$, and set $m_{t,s} = |\mathcal{M}_{t,s}|$. For each machine type t and speed s, let $S_{t,s}$ and $B_{t,s}$ be the sets of occurring small and big job sizes. Furthermore, let P_t be the set of all occurring job sizes for type t. Like before, we have $|B_{t,s}| \in \mathcal{O}(1/\varepsilon \log 1/\varepsilon)$. For any processing time p, we denote the set of jobs j with $p_{tj} = p$ by $J_t(p)$.

SEPARATION OF MACHINES. We will consider configurations for each machine type t and speed value $s \in V_t$. However, the number of distinct speed values could be dependent in m and we cannot afford to introduce integral variables in the MILP for each of them. Instead, we will introduce integral variables only for the fastest speeds of each type and round the fractional variables. For the rounding approach, we will need a constant number of machines that receive some load from the slow speeds, and, furthermore, the speeds of these machines have to be faster than the slow speeds by some constant factor. This leads to a separation of the machines into three groups $G_{t,i}$ for $i \in [3]$ for each machine type t. This is done in a way such that for $j > i$ the machines in group $G_{t,i}$ are faster than the ones in group $G_{t,j}$. For $i \in [3]$ and opt $\in \{\min, \max\}$, we set $s^{(t)}_{i,\mathrm{opt}} := \mathrm{opt}\{s_i \mid i \in G_{t,i}\}$. The partition is defined by two parameters. The first parameter

$$\kappa := \max\{|B_{t,s}| + 1 \mid \forall t \in [K], s \in V_t\} \in \mathcal{O}(1/\varepsilon \log 1/\varepsilon)$$

controls the number of machines in the first group and the second $\gamma := 1/\varepsilon^2$ the speed-gap between the first and the third group. More precisely:

- $G_{t,1}$ contains the κ fastest machines of type t.
- $G_{t,2}$ contains all machines of type t that are not contained in $G_{t,1}$ and whose speed is bigger than $\gamma s^{(t)}_{1,\min}$.

- $G_{t,3}$ contains the rest of the machines of type t.

Note that $G_{t,2}$ and $G_{t,3}$ might be empty. We denote the occurring speeds in group $G_{t,i}$ by $V_{t,i}$ and call the speeds from $V_{t,1} \cup V_{t,2}$ *fast* and the rest *slow*. With these definitions, we have $(V_{t,1} \cup V_{t,2}) \cap V_{t,3} = \emptyset$ and $|V_{t,1}|, |V_{t,2}| \in \mathcal{O}(1/\varepsilon \log(1/\varepsilon))$, i.e., the fast and slow speed values are distinct and we have only a constant number of fast speed values. A similar separation step is also used by Jansen [72]. In that work, the fastest group is handled differently to get a slightly improved running time, and we remark that the same approach is applicable for the present result as well.

MILP. For each machine type t and speed $s \in V_t$, we consider the set $\mathcal{C}_t(sT)$ of configurations C for the big processing times $B_{t,s}$ and with $\Lambda(C) \leqslant sT$. Note that $|\mathcal{C}_t(sT)| \in 2^{\mathcal{O}(1/\varepsilon \log^2 1/\varepsilon)}$ and therefore $|\bigcup_{s \in V_{t,1} \cup V_{t,2}} \mathcal{C}_t(sT)| \in 2^{\mathcal{O}(1/\varepsilon \log^2 1/\varepsilon)}$.

The MILP formulation in this scenario follows the same basic ideas but is more complicated than before. We assign jobs fractionally to machines types and count the number of jobs of each job size that are assigned to each machine type. If the job size is big on some fast machine of the machine type, we require an integral number of jobs. The main difference to the prior approaches is that we have to deal with different speeds for each machine type, and jobs may be handled either as big or small jobs. To deal with the former, we choose configurations—integrally for fast machine speeds and fractionally otherwise. Lastly, we fractionally assign job sizes to machine speeds for which they are small. More precisely, we introduce the following variables:

- Configuration variables $z_C^{(t,s)}$ for each machine type $t \in [K]$, occurring speed $s \in V_t$, and configuration $C \in \mathcal{C}_t(sT)$. If s is fast, we require $z_C^{(t,s)} \in \mathbb{Z}_{\geqslant 0}$ and otherwise $z_C^{(t,s)} \geqslant 0$.
- Job assignment variables $x_{j,t} \geqslant 0$ for each machine type $t \in [K]$ and job $j \in \mathcal{J}$.
- Job size assignment variables $y_{p,s}^{(t)} \geqslant 0$ for each machine type $t \in [K]$, speed $s \in V_t$, and job size $p \in S_{t,s}$.
- Counting variables $u_p^{(t)}$ for each machine type $t \in [K]$ and job size $p \in P_t$. If there is a fast speed $s \in V_{t,1} \cup V_{t,2}$ such that $p \in B_{t,s}$, we require $u_p^{(t)} \in \mathbb{Z}_{\geqslant 0}$ and otherwise $u_p^{(t)} \geqslant 0$.

We require $x_{j,t} = 0$ if $p_{tj} = \infty$, and $y_{p,s}^{(t)} = 0$ if p is big or huge for s. Besides this, the MILP is is given by the following constraints:

$$\sum_{C \in \mathcal{C}_t(sT)} z_C^{(t,s)} = m_{t,s} \qquad \forall t \in [K], s \in V_t \quad (3.13)$$

$$\sum_{t\in[K]} x_{j,t} = 1 \qquad \forall j \in \mathcal{J} \quad (3.14)$$

$$\sum_{j\in J_t(p)} x_{j,t} \leqslant u_p^{(t)} \qquad \forall t \in [K], p \in P_t \quad (3.15)$$

$$\sum_{s:p\in B_{t,s}} \sum_{C\in\mathcal{C}_t(sT)} C_p z_C^{(t,s)} + \sum_{s:p\in S_{t,s}} y_{p,s}^{(t)} \geqslant u_p^{(t)} \qquad \forall t \in [K], p \in P_t \quad (3.16)$$

$$\sum_{C\in\mathcal{C}_t(sT)} \Lambda(C) z_C^{(t,s)} + \sum_{p\in S_{t,s}} p y_{p,s}^{(t)} \leqslant m_{t,s} sT \quad \forall t \in [K], s \in V_t \quad (3.17)$$

The Constraints (3.13) and (3.14) are very similar to constraints for the other MILPs that we consider. For each machine type it is ensured with the Constraints (3.15) and (3.16) that the summed up number of jobs of each size is covered by the the chosen configurations and the small job assignments. Furthermore, (3.17) guarantees that the overall processing time of the configurations and small jobs assigned to a machine speed for each type does not exceed the available area.

LEMMA 3.20. *If there is schedule with makespan* T*, then there is a feasible (integral) solution of* MILP(T)*; and if there is a feasible integral solution for* MILP(T)*, then there is a schedule with makespan at most* $(1+\varepsilon^2)T$.

Proof. Given a schedule σ with makespan T, each machine of type t with speed s obeys exactly one configuration from $\mathcal{C}_t(sT)$, and we can set the variables $z_{C,t}$ accordingly. Furthermore, we set $x_{j,t_{\sigma(j)}} = 1$, $x_{j^*,t} = 0$ for $t \neq t_{\sigma(j)}$, $u_p^{(t)} := |\{j \mid t_{\sigma(j)} = t, p_{t,j} = p\}|$, and $y_{p,s}^{(t)} := |\{j \mid t_{\sigma(j)} = t, s_{\sigma(j)} = s, p_{t,j} = p\}|$. It is easy to check that all conditions are fulfilled.

Like we did in the proof of Lemma 3.6, given an integral solution (z, x, y, u), we can assign the jobs to machine types and configurations to machines. Moreover, based one the $y_{p,s}^{(t)}$ variables we can assign jobs of size p that are assigned to type t to machines of speed s on which they are small. Because of (3.15) and (3.16), this can be done such that the remaining jobs of size p can be scheduled into slots provided by configurations. At this point, each unscheduled job is assigned to a type and a speed. Utilizing (3.17), these jobs can be scheduled greedily with an additive error of $\varepsilon^2 T$. □

LEMMA 3.21. *The MILP has* $\mathcal{O}(K(n+m))$ *many constraints,* $\mathcal{O}(Knm) + m2^{\mathcal{O}(1/\varepsilon \log^2 1/\varepsilon)}$ *many variables and* $K2^{\mathcal{O}(1/\varepsilon \log^2 1/\varepsilon)}$ *integral variables. It can be solved in time:*

$$2^{\mathcal{O}(K \log(K) 1/\varepsilon^3 \log^5 1/\varepsilon)} \mathrm{poly}(|I|)$$

Proof. The bounds for the number of constraints and variables are easy to verify using the above considerations as well as $|V_t| \leqslant m$ and $|P_t| \leqslant n$. The running time can be achieved with the first approach presented in Section 3.3: Using Theorem 3.9, we can argue that $\mathcal{O}(1/\varepsilon \log^2(1/\varepsilon))$ many integral variables for each machine type

t and speed s suffice. Therefore, the number of needed guesses is $2^{\mathcal{O}(K/\varepsilon^3 \log^5 1/\varepsilon)}$. Running the algorithm of Lenstra and Kannan (see Theorem 3.7) with $\mathcal{O}(K/\varepsilon^2 \log^3(1/\varepsilon))$ many integral variables takes $2^{\mathcal{O}(K \log(K) 1/\varepsilon^2 \log^4 1/\varepsilon)} \text{poly}(|I|)$ time. Together we get the stated running time. □

ROUNDING. We present rounding approaches for all fractional variables and start with the configuration variables. Again, many of the rounding steps can be found in a similar form in the EPTAS result by Jansen [72].

CONFIGURATION VARIABLES. We fix a type t, a slow speed $s \in V_t$ and set $k_p := \sum_{C \in \mathcal{C}_t(sT)} C_p z_C^{(t,s)}$ for each $p \in B_{t,s}$. We have:

$$\sum_{C \in \mathcal{C}_t(sT)} z_C^{(t,s)} = m_{t,s} \tag{3.18}$$

$$\sum_{C \in \mathcal{C}_t(sT)} C_p z_C^{(t,s)} = k_p \qquad \forall p \in B_{t,s} \tag{3.19}$$

It is easy to check that, if we replace the solution of this LP with any other solution and change the MILP solution accordingly, the resulting MILP solution will still be feasible. We transform the solution into a basic feasible solution. This can be done in polynomial time with respect to $1/\varepsilon$ and $|I|$. The LP has $|B_{t,s}| + 1$ many constraints and therefore the solution has at most $|B_{t,s}| + 1$ many variables greater than 0. Now, the idea is to round down the fractional values and to assign the respective job sizes that lost covering by the configurations to the fastest group $G_{t,1}$. More precisely, we choose some injective mapping ξ between the configurations C with fractional variables $z_C^{(t,s)}$ and the machines from $G_{t,1}$. This can be done due to the choice of the parameter κ that regulates the number of machines in $G_{t,1}$. Now, we round down $z_C^{(t,s)}$ to the next integral value and increase $y^{(t)}_{p,s_{\xi(C)}}$ by $(\lceil z_C^{(t,s)} \rceil - z_C^{(t,s)}) C_p \leqslant C_p$ for each $p \in B_{t,s}$. We perform these steps for all types t and slow speeds $s \in V_t$. Note that any particular variable $y^{(t)}_{p,s}$ might be increased several times for each speed value for which p is big. Let $\tilde{y}^{(t)}_{p,s}$ denote the resulting increased $y^{(t)}_{p,s}$ variables and $\bar{z}_C^{(t,s)}$ the resulting configuration variables. For $(\bar{z}, x, \tilde{y}, u)$ the Constraints (3.13)-(3.15) obviously still hold, and it is easy to see that this is also the case for (3.16), while (3.17) might be violated for speeds associated with the fastest machine groups. We show that a modified version of (3.17) still holds.

Consider a machine $i \in G_{t,1}$. For each slow speed value $s \in V_{t,3}$, there may be one configuration C that is mapped to i. The summed up job sizes that are reassigned to s_i because of this are bounded by sT. Summing up over all speed values $s \in V_{t,3}$ and utilizing the

convergence of the geometric series, the rounding of the speed values, and the fact that $s^{(t)}_{3,max} \leqslant \gamma s^{(t)}_{1,max} = 1/\varepsilon^2 s^{(t)}_{1,max}$, we get:

$$\sum_{s \in V_{t,3}} sT = Ts^{(t)}_{3,max} \sum_{s \in V_{t,3}} \frac{s}{s^{(t)}_{3,max}}$$
$$< Ts^{(t)}_{3,max} \sum_{i=0}^{\infty} \frac{1}{(1+\varepsilon)^i} \leqslant Ts^{(t)}_{1,min}(\varepsilon + \varepsilon^2) \leqslant Ts_i(\varepsilon + \varepsilon^2)$$

Hence, (3.17) holds if we increase the makespan on the right hand side by $(\varepsilon + \varepsilon^2)T$.

COUNTING VARIABLES. The rounding step for the counting variables u is the easiest: We round them up and assign the extra job sizes to the fastest machine speed in the group, that is, for each $t \in [K]$ and $p \in P_t$, we set $\bar{u}^{(t)}_p = \lceil u^{(t)}_p \rceil$ and increase $\tilde{y}^{(t)}_{p,s^*}$ by $\bar{u}^{(t)}_p - u^{(t)}_p \leqslant 1$ where $s^* = s^{(t)}_{max}$. We denote the changed $\tilde{y}^{(t)}_{p,s^*}$ by $\breve{y}^{(t)}_{p,s^*}$. Again, it is easy to see that for $(\bar{z}, x, \breve{y}, \bar{u})$ the Constraints (3.13)-(3.16) still hold, while (3.17) is violated. However, we can bound the increase the fastest speed receives again utilizing the geometric series:

$$\sum_{p \in S_{t,s^*}} p(\breve{y}^{(t)}_{p,s^*} - \tilde{y}^{(t)}_{p,s^*}) \leqslant \sum_{p \in S_{t,s^*}} p \leqslant \varepsilon^2 s^* T \frac{1+\varepsilon}{\varepsilon} = (\varepsilon + \varepsilon^2) s^* T$$

Hence, (3.17) holds if we further increase the makespan by $(\varepsilon + \varepsilon^2)T$.

JOB SIZE ASSIGNMENT VARIABLES. Consider the Constraint (3.16) for the solution $(\bar{z}, x, \breve{y}, \bar{u})$. Note that the right hand side and the first sum on the left hand side are both integral. Therefore, we can scale the $\breve{y}^{(t)}_{p,s}$ variables down such that $\sum_{s:p \in S_{t,s}} \breve{y}^{(t)}_{p,s}$ is integral for each machine type t and job size $p \in P_t$, and (3.16) is still fulfilled. We fix a machine type t, set $k_p := \sum_{s:p \in S_{t,s}} \breve{y}^{(t)}_{p,s}$ for each $p \in P_t$, and assume $k_p \in \mathbb{Z}$ because of the argument above. Furthermore, we set $L_{t,s} := \sum_{p \in S_{t,s}} p\breve{y}^{(t)}_{p,s}$ for each $s \in V_t$. We have:

$$\sum_{s:p \in S_{t,s}} \breve{y}^{(t)}_{p,s} = k_p \qquad \forall p \in P_t \tag{3.20}$$

$$\sum_{p \in S_{t,s}} p\breve{y}^{(t)}_{p,s} = L_{t,s} \qquad \forall s \in V_t \tag{3.21}$$

If we replace the values $\breve{y}^{(t)}_{p,s}$ with any other solution for the above LP, we get an equivalent MILP solution. We can use a variation of the classical rounding approach by Lenstra et al. [108] to transform the solution $\breve{y}^{(t)}_{p,s}$.

For the sake of completeness, we summarize the main ideas of the rounding. The solution is transformed into a basic feasible one and the following bipartite graph is considered. There are two types of

nodes, some associated with sizes p and some with speeds s. For any p or s, there can be at most one node; there are such nodes if and only if there are fractional variables $\breve{y}^{(t)}_{p,s'}$ or $\breve{y}^{(t)}_{p',s}$ left; and they are connected with an edge, if there is a fractional variable $\breve{y}^{(t)}_{p,s}$. Using a counting argument and some further considerations it can be shown that this graph is a pseudoforest, i.e., all connected components are either trees or trees with one extra edge. Furthermore, because k_p is integral, the definition of the graph together with the Constraint (3.20) yield that all the leafs are associated to speeds. Using this structure, we can define an injective mapping ξ from the job sizes for which there is a fractional variable $\breve{y}^{(t)}_{p,s}$ to the speeds such that $\breve{y}^{(t)}_{p,\xi(p)}$ is one of the fractional variables. This can be done as follows: For each connected component there may be at most one cycle in the graph with alternating size and speed nodes, and a suitable injective mapping for the corresponding sizes and speeds can easily be found by going around the cycle and appropriately mapping consecutive nodes. After removing the corresponding nodes and edges, only trees remain in the graph. For each tree, we can choose an arbitrary leaf. The leaf corresponds to a speed and its neighbor to a size, and we can map the size to the speed and remove both corresponding nodes from the graph. Iterating this yields the mapping ξ. All the above steps can be performed in polynomial time in $1/\varepsilon$ and $|I|$.

We use the mapping ξ to round the variables $\breve{y}^{(t)}_{p,s}$, and, because ξ is injective, we can guarantee that each speed receives at most one extra small job. More precisely, for each $s \in V_t$, we set $\bar{y}^{(t)}_{p,s} = \lceil \breve{y}^{(t)}_{p,s} \rceil$ if $\xi(p) = s$ and $\bar{y}^{(t)}_{p,s} = \lfloor \breve{y}^{(t)}_{p,s} \rfloor$ otherwise. For the solution $(\bar{z}, x, \bar{y}, \bar{u})$, the Constraints (3.13)-(3.16) still hold, while for (3.17) the makespan has to be increased further by $\varepsilon^2 T$.

JOB ASSIGNMENT VARIABLES. The rounding of the job assignment variables is the same as in the regular machine types case. The only difference is that we can set $\eta_{t,p} = \bar{u}^{(t)}_p$ in this case. Since all the values $\bar{u}^{(t)}_p$ are integral in this case, there is no further rounding error in this step. Let $\bar{x}_{j,t}$ be the rounded version of $x_{j,t}$.

Summarizing the previous steps, the Constraints (3.13)-(3.16) hold for the solution $(\bar{z}, \bar{x}, \bar{y}, \bar{u})$. Moreover, we have for each $t \in [K]$ and $s \in V_t$:

$$\sum_{C \in \mathcal{C}_t(sT)} \Lambda(C) \bar{z}^{(t,s)}_C + \sum_{p \in S_{t,s}} p \bar{y}^{(t)}_{p,s} \leqslant m_{t,s} s (1 + 2\varepsilon + 3\varepsilon^2) T \tag{3.22}$$

ANALYSIS. Summarizing the above steps, we can construct a schedule with makespan at most $(1+\varepsilon)^2(1+2\varepsilon+4\varepsilon^2)T \leqslant (1+27\varepsilon)T$ (assuming a schedule with makespan T exists) by building and solving the MILP, then rounding it, and lastly transforming it into a schedule like in the proof of Lemma 3.20. Solving the MILP is again the most

expensive step, and with a simple case analysis we get a running time of:

$$2^{\mathcal{O}(K \log(K) 1/\varepsilon^3 \log^5 1/\varepsilon)} + \text{poly}(|I|)$$

3.6 VECTOR SCHEDULING

We present an EPTAS for D-dimensional unrelated vector scheduling, where both the dimension D and the number K of machine types are constant. In this problem variant, for each job j, a D-dimensional processing time vector $p_{tj} = (p_{tj}^{(1)}, \dots, p_{tj}^{(D)})$ is given, and the makespan is defined as the maximum load any machine receives in any dimension, i.e., $C_{\max}(\sigma) = \max_{i \in \mathcal{M}} \| \sum_{j \in \sigma^{-1}(i)} p_{ij} \|_\infty$. We define $P = \{p_{tj}^{(d)} \mid d \in [D], t \in [K], j \in \mathcal{J}\}$ and $\mathcal{P} = \{p_{tj} \mid t \in [K], j \in \mathcal{J}\}$.

The EPTAS is a direct adaptation of the one for the one dimensional case. In the following, we briefly describe the needed extra steps and modification. Note that we consider this result to be a proof of concept and took little effort to optimize the running time.

PRELIMINARIES. We again use the dual approximation approach to get a guess T of the makespan. As an upper bound for this we can use the schedule that we get by assigning each job j to a machine i where $\sum_{d=1}^{D} p_{ij}^{(d)}$ is minimal. It is easy to see that this approach yields a Dm-approximation and we can use this result for the dual approximation like described in Section 3.4.

First, we perform rounding steps similar to those for the other results. For each $p_{tj} \in \mathcal{P}$ with $p_{tj}^{(d)} > T$ in at least one dimension d, we set $\bar{p}_{tj} = (\infty, \dots, \infty)$, and, for all other processing time vectors p_{tj}, we apply geometric rounding. Let $\theta = (\varepsilon^2/D)^D$ be some threshold parameter. We set $\bar{p}_{tj}^{(d)} = (1+\varepsilon)^x \theta T$ with $x = \lceil \log_{1+\varepsilon}(p_{tj}^{(d)}/(\theta T)) \rceil$ yielding a rounded vector $\bar{p}_{tj}$ and a corresponding rounded instance $\bar{I}$.

For a given processing time vector, the numbers that can occur in the different dimensions may still differ strongly. This complicates the problem, but we can reduce the extra complexity to some degree via a second rounding step: For each $\bar{p}_{tj}$, we set $\tilde{p}_{tj}^{(d)} = \max\{\bar{p}_{tj}^{(d)}, \|\bar{p}_{tj}\|_\infty \varepsilon/D\}$ yielding a rounded vector $\tilde{p}_{tj}$ and a corresponding rounded instance $\tilde{I}$.

LEMMA 3.22. *If there is a schedule with makespan at most T for I, then the same schedule has makespan at most $(1+\varepsilon)^2 T$ for instance $\tilde{I}$; and any schedule for instance $\tilde{I}$ can be turned into a schedule for I without increase in the makespan.*

Proof. Consider a schedule σ with makespan T for I. The first rounding step may increase the makespan by a factor of $(1+\varepsilon)$. We fix a machine i and a dimension d and bound the increase in load on machine i

in dimension d for instance $\tilde{I}$. Let j be a job with $\sigma(j) = i$. If $\bar{p}_{ij}^{(d)} = \|\bar{p}_{ij}\|_\infty$, then job j causes no extra load on i in dimension d, and, if $\bar{p}_{ij}^{(d')} = \|\bar{p}_{ij}\|_\infty$ for some dimension $d' \neq d$, there might be an increase of at most $\|\bar{p}_{ij}\|_\infty \varepsilon/D$. In fact, the summed up load machine i receives in dimension d' might increase the load in d by an ε/D-factor in this fashion. Because the load in dimension d' is bounded by $(1+\varepsilon)T$ and there are $D-1$ dimensions $d' \neq d$, the overall load increase in dimension d on i can be up to $(D-1)(1+\varepsilon)T\varepsilon/D \leqslant \varepsilon(1+\varepsilon)T$. □

Hence, we may search for a schedule for instance $\tilde{I}$ with makespan $\tilde{T} := (1+\varepsilon)^2 T$. For the sake of simplicity, we do not use the $(\tilde{\cdot})$-notation in the following, i.e., we assume that the instance is already rounded and the makespan properly increased.

In this context, we call a size $q \in P$ *big* if $q > \theta T$ and *small* otherwise. Furthermore, we call a processing time vector $p \in \mathcal{P}$ *big* if there is a dimension $d \in [D]$ such that $p^{(d)}$ is big and *small* otherwise. Because of the second rounding step, we have $p^{(d)} > \theta T\varepsilon/D$ for each big vector p and dimension d. Let B_t and S_t be the sets of big and small processing time vectors occurring on machine type t. Note that $|B_t| \leqslant (\lceil \log_{1+\varepsilon}(D/(\theta\varepsilon))\rceil)^D \leqslant (3D/\varepsilon \log(D/\varepsilon))^D$. Using these definition, the bound on the number of big jobs is much bigger than in the other cases. We chose this definition because in the rounding of the MILP solution each machine may receive a big (but constant) number of jobs for each small job size, and to bound the overall load the small jobs have to be appropriately small.

For each processing time vector $p \in \mathcal{P}$, we denote the set of jobs j with $p_{tj} = p$ with $J_t(p)$.

MILP. Similar to the one dimensional case, for any set V of processing time vectors, we call the V-indexed vectors of non-negative integers $\mathbb{Z}_{\geqslant 0}^V$ configurations (for V), set the size $\Lambda(C)$ of a configuration C to be the corresponding vector of sizes, i.e., $\Lambda(C) = \sum_{p \in \mathcal{P}} C_p p$, and set $\mathcal{C}_t(T)$ to be the set of configurations C for B_t with $\Lambda(C) \leqslant T^D$. Note that:

$$\begin{aligned}|\mathcal{C}_t(T)| &\leqslant \left(\frac{D}{\theta\varepsilon}+1\right)^{(3D/\varepsilon \log(D/\varepsilon))^D} \\ &\leqslant \left(\frac{D}{\varepsilon}\right)^{3D(3D/\varepsilon \log D/\varepsilon)^D} \leqslant 2^{(3D/\varepsilon \log D/\varepsilon)^{D+1}}\end{aligned}$$

The MILP is a straight-forward adaptation of the one for the one-dimensional case with one important difference: The jobs are fractionally assigned to configurations belonging to a type instead of just being assigned to machine types. More precisely, we introduce integral variables $z_{C,t} \in \mathbb{Z}_{\geqslant 0}$ for each machine type $t \in [K]$ and configuration $C \in \mathcal{C}_t(T)$ and fractional variables $x_{j,t,C} \geqslant 0$ for each job $j \in \mathcal{J}$, ma-

chine type $t \in [K]$ and configuration $C \in \mathcal{C}_t(T)$. For $p_{tj} = \infty^D$ we set $x_{j,t,C} = 0$. MILP(T) is given by:

$$\sum_{C \in \mathcal{C}_t(T)} z_{C,t} = m_t \qquad \forall t \in [K] \tag{3.23}$$

$$\sum_{t \in [K]} \sum_{C \in \mathcal{C}_t(T)} x_{j,t,C} = 1 \qquad \forall j \in \mathcal{J} \tag{3.24}$$

$$\sum_{j \in J_t(p)} x_{j,t,C} \leqslant C_p z_{C,t} \qquad \forall t \in [K], p \in B_t, C \in \mathcal{C}_t(T) \tag{3.25}$$

$$\sum_{p \in S_t} p \sum_{j \in J_t(p)} x_{j,t,C} \leqslant (T^D - \Lambda(C)) z_{C,t} \qquad \forall t \in [K], C \in \mathcal{C}_t(T) \tag{3.26}$$

Note that the last constraint is D-dimensional. Unlike in the other cases, we cannot transform an integral solution for MILP(T) directly into a schedule with only a small increase in the makespan. However, we deal with this in the rounding step and still have:

LEMMA 3.23. *If there is schedule with makespan* T *there is a feasible (integral) solution of* MILP(T).

We can solve MILP(T) in time $f(1/\varepsilon, D, K)\text{poly}(|I|)$ for some computable function f using the algorithm by Lenstra and Kannan (see Theorem 3.7).

ROUNDING. Using a variation of the rounding approach for the one dimensional case, we can transform a solution (z, x) for MILP(T) into a schedule with a makespan of at most $(1 + \varepsilon + \varepsilon^2)T$. The main difference is that we create nodes for pairs of *machines* and processing time vectors instead of pairs of machine types and processing times.

For each type t, we assign configurations to machines of type t such that for each configuration $C \in \mathcal{C}_t(T)$ exactly $z_{C,t}$ configurations get assigned. Therefore, we can assume that for each machine i a configuration $C^{(i)}$ is given. Based on this, we can fractionally assign jobs to machines by setting $x_{j,i} = x_{j,t,C^{(i)}} / z_{C^{(i)},t}$ yielding:

$$\sum_{j \in \mathcal{J}} p_{ij} x_{j,i} \leqslant T^D \tag{3.27}$$

For each machine i, let $\mathcal{P}_i$ be the set of occurring processing time vectors for i, that is, for each $p \in \mathcal{P}$, we have $p \in \mathcal{P}_i$ if and only if there is a job j with $p_{ij} = p$ and $x_{j,i} > 0$. We set $\eta_{i,p} = \lceil \sum_{j \in J_t(p)} x_{j,i} \rceil$. If p is big, we have $\eta_{i,p} \leqslant C_p^{(i)}$ because of Constraint (3.25).

Like in the one dimensional case, the flow network $G = (V, E)$ has a source α and sink ω, and, for each job $j \in \mathcal{J}$, there is a job node v_j and an edge (α, v_j) with capacity 1 connecting the source and the job node. Moreover, for each machine i, we have processing time vector nodes $u_{i,p}$ for each $p \in \mathcal{P}_i$. The processing time nodes are connected to the sink via edges $(u_{i,p}, \omega)$ with capacity $\eta_{i,p}$. Lastly, for each job j and machine type i with $x_{j,i} > 0$, we have an edge $(v_j, u_{i,p_{i,j}})$ with

capacity 1 connecting the job node with the corresponding processing time vector nodes. The variables $x_{j,i}$ yield a flow with value n that is guaranteed to be feasible because of the constraints of the MILP.

LEMMA 3.24. *G has a maximum flow f with value* $|f| = n$.

Using the Ford-Fulkerson algorithm, an integral maximum flow f^* can be found in time $\mathcal{O}(|E||f^*|) = \mathcal{O}(n^2 m)$. Due to flow conservation, for each job j, there is exactly one machine i^* such that $f((v_j, u_{i^*,p_{i^*j}})) = 1$, and we set $\sigma(j) = i^*$. Analogously to the one dimensional case, for each big processing time vector p, the schedule σ assigns at most $C_p^{(i)}$ many jobs j with $p_{ij} = p$ to machine i, and, for each small processing time vector p', one additional job j with $p_{ij} = p'$ may be assigned to i. Because of the choice of the parameter θ and the second rounding step, we can bound the extra load i receives.

LEMMA 3.25. *Let* $q \in P$ *be small and* $d \in [D]$. *The number of vectors* $p \in \mathcal{P}$ *with* $p^{(d)} = q$ *is upper bounded by* $(2\lceil \log_{1+\varepsilon}(D/\varepsilon)\rceil)^{D-1}$.

Proof. Let p be such a vector, $d' \neq d$ and $q' = p^{(d')}$. Because of the second rounding step, we have $q' \geqslant q\varepsilon/D$ and $q \geqslant q'\varepsilon/D\ p^{(d')}$. Now, because of the first rounding step, there are only few such processing times q, more precisely, at most $2\lceil \log_{1+\varepsilon}(D/\varepsilon)\rceil$. Hence, there can be at most $(2\lceil \log_{1+\varepsilon}(D/\varepsilon)\rceil)^{D-1}$ many such processing time vectors p. □

Using this lemma and the same argumentation as in the one dimensional case, we can bound the extra load machine i receives in dimension d by:

$$\begin{aligned}
&(2\lceil \log_{1+\varepsilon}(D/\varepsilon)\rceil)^{D-1} \times \theta T \times \sum_{i=0}^{\infty} 1/(1+\varepsilon)^i \\
\leqslant\ & 2(1/\varepsilon \log((D-1)/\varepsilon) + 1)^{D-1} \times (\varepsilon^2/D)^D T \times (1+\varepsilon)/\varepsilon \\
\leqslant\ & 2(D/\varepsilon^2)^{D-1} \times (\varepsilon^2/D)^D T \times (1+\varepsilon)/\varepsilon \\
\leqslant\ & \varepsilon^2 T \times (1+\varepsilon)/\varepsilon \leqslant (\varepsilon + \varepsilon^2) T
\end{aligned}$$

Summarizing we have:

LEMMA 3.26. *A solution* (z, x) *for* MILP(T) *can be transformed into a schedule with makespan at most* $(1 + \varepsilon + \varepsilon^2)T$ *in time polynomial in* $|I|$ *and* $1/\varepsilon$.

Therefore, there is an EPTAS for this case as well.

3.7 OPEN PROBLEMS

In the following, we briefly discuss some possible directions for further studies.

BETTER RUNNING TIMES. The presented approximation schemes have running times of the form $f(1/\varepsilon, K) + \text{poly}(|I|)$ (or $f(1/\varepsilon, K, D) + \text{poly}(|I|)$ in the vector scheduling case). While we took some effort to optimize f at least for the first two schemes, we did not optimize the $\text{poly}(|I|)$ part in any of the results. Furthermore, for the case with a constant number of uniform types, one could study whether a quadratic or linear dependence in $1/\varepsilon$ (ignoring polylogarithmic dependencies) in the exponent of the $f(1/\varepsilon, K)$ part can be achieved, e.g., by utilizing techniques from [72] and [73]. Lastly, the EPTAS for the vector scheduling variant is basically just a proof of concept and we did not optimize the running time at all.

LOWER BOUND. Chen et al. [30] showed that we cannot hope for an EPTAS with a sub-linear dependency in $1/\varepsilon$ in the exponent unless the exponential time hypothesis fails. It is unclear what can be ruled out in terms of the parameter K.

JOB TYPES. In the introduction we mentioned the concept of job types for scheduling on unrelated parallel machines: Two jobs j, j' are of the same type, if they behave the same on every machine i, i.e., $p_{ij} = p_{ij'}$. It is unknown, whether there is a PTAS for scheduling on unrelated parallel machines with a constant number of job types. Furthermore, it is unknown, whether this problem is NP-hard. Indeed, the problem is in P for important special cases: For scheduling on identical parallel machines the number of job types is equal to the number of distinct processing times and for the case of the restricted assignment problem—where each job j has a size p_j and its processing time p_{ij} on machine i is either p_j or ∞—the number of distinct processing times is bounded by the number of job types and a constant number of job types implies a constant number of machine types. Both problems can be solved in polynomial time, if the number of distinct processing times is constant.

4

INTERVAL AND RESOURCE RESTRICTIONS

4.1 INTRODUCTION

Consider the restricted assignment problem: Given a set of machines $\mathcal{M}$ and a set of jobs $\mathcal{J}$ each with a processing time or size p_j and a subset of eligible machines $\mathcal{M}(j) \subseteq \mathcal{M}$, the goal is to find a schedule $\sigma : \mathcal{J} \to \mathcal{M}$ with $\sigma(j) \in \mathcal{M}(j)$ for each job j and minimizing the makespan $C_{\max}(\sigma) = \max_{i \in \mathcal{M}} \sum_{j \in \sigma^{-1}(i)} p_j$.

In a seminal work, Lenstra, Shmoys and Tardos [108] presented a 2-approximation for restricted assignment and also showed that there is no polynomial time approximation algorithm with rate smaller than 1.5 for the problem, unless P=NP. If there are no restrictions, i.e., $\mathcal{M}(j) = \mathcal{M}$ for each job j, we have the classical problem of machine scheduling which is already strongly NP-hard. On the other hand, machine scheduling is well-known to admit PTAS due to a classical result by Hochbaum and Shmoys [65]. In recent years, the approximability of special cases of restricted assignment has been intensively studied (see, e.g., [25, 40, 68, 85]) with one line of research focusing on the existence of approximation schemes (see, e.g., [46, 82, 119, 122]). In this and the following chapter, we present contributions to this research direction.

INTERVAL RESTRICTIONS. Arguably one of the most natural variants of the restricted assignment problem is the case of scheduling with interval restrictions (RAI). In this variant, the machines are totally ordered and each job is eligible on consecutive machines. More precisely, we have $\mathcal{M} = \{M_1, \ldots, M_m\}$, and for each job j we have $\mathcal{M}(j) = \{M_\ell, \ldots, M_r\}$ for some indices $\ell, r \in [m]$. Several special cases of RAI are known to admit a PTAS: the hierarchical case [122], where for each job the interval of eligible machines starts with the first machine; the nested case [46, 119], where $\mathcal{M}(j) \subseteq \mathcal{M}(j')$, $\mathcal{M}(j') \subseteq \mathcal{M}(j)$ or $\mathcal{M}(j) \cap \mathcal{M}(j') = \emptyset$ for each pair of jobs (j, j'); and the inclusion-free case [97, 135], where $\mathcal{M}(j) \subseteq \mathcal{M}(j')$ implies that j and j' share either their first or last eligible machine. Furthermore, for general RAI, a $2 - 2/(\max_{j \in \mathcal{J}} p_j)$-approximation due to Schwarz [135] is known (assuming integral processing times); and the special case with two distinct processing times is even polynomial time solvable [143]. Note that the problem has also been studied in the context of online algorithms (see [107, 110]).

The question of whether there is a PTAS for RAI has been posed by several authors [95, 135, 143]. As the main result of this chapter, we resolve the question in the negative:

THEOREM 4.1: There is no PTAS for scheduling with interval restrictions unless P=NP. [1]

RESOURCE RESTRICTIONS. The second variant considered in this chapter, is the problem of scheduling with resource restrictions with R resources (RAR(R)). Herein, a set $\mathcal{R}$ of R (renewable) resources is given, each machine i has a resource capacity $c_r(i)$ and each job j has a resource demand $d_r(j)$ for each $r \in \mathcal{R}$. Job j is eligible on machine i if $d_r(j) \leqslant c_r(i)$ for each resource r. For $R = 1$, the problem is equivalent to the mentioned hierarchical case and has been studied intensively [109, 110]. Furthermore, it is not hard to see that RAI is properly placed between RAR(1) and RAR(2) (see Section 4.3) and hence there is a close relationship between the two problems. For arbitrary R, the problem was mentioned in a work by Bhaskara et el. [16] under the name of geometrically restricted scheduling[2] but to the best of our knowledge it has not been further studied up to now. There is, however, a close relationship to the low rank version of unrelated scheduling introduced in [16]. In the problem of unrelated scheduling, the processing time of each job is dependent on the machine it is scheduled on, that is, a processing time matrix $(p_{ij})_{j\in\mathcal{J},i\in\mathcal{M}}$ is given in the input. Restricted assignment can be seen as a special case of unrelated scheduling by setting $p_{ij} = p_j$ for $i \in \mathcal{M}(j)$ and $p_{ij} = \infty$ otherwise. In the rank D version of unrelated scheduling (LRS(D)), the processing time matrix has a rank of at most D, or, equivalently [31], we may assume that there are D-dimensional size vectors $s(j)$ for each job j and speed vectors $v(i)$ for each machine i such that $p_{ij} = \sum_{k=1}^{D} s_k(j) \cdot v_k(i)$. Considering the latter definition, scheduling with resource restrictions may intuitively be seen as the restricted assignment equivalent of low rank unrelated scheduling. It is not hard to see that formally already for RAR(1) instances the processing time matrix can have rank $|\mathcal{M}|$. However, LRS(D) includes approximations of any RAR($D-1$) instance with arbitrary precision (see Section 4.3 for details). The case with $D = 1$ of LRS(D) is equivalent to uniform scheduling and well known to admit a PTAS [66]. Bhaskara et el. [16] gave a quasi-polynomial time approximation scheme (QPTAS) for $D = 2$, and showed that there is no PTAS or approximation algorithm with rate smaller than 1.5 for $D \geqslant 4$ or $D \geqslant 7$ respectively. The latter two results have been improved from $D = 4$ to $D = 3$ by Chen et al. [31] and from $D = 7$ to $D = 4$ by Chen, Ye and Zhang [30]. We

1 There is a paper [96] claiming to have found a PTAS for RAI. However, according to [136], the result is not correct and the authors published a revised version of the paper [97] claiming a less general result, namely, a PTAS for the inclusion-free case.

2 The demands $d(j)$ and capacities $c(i)$ may be interpreted as points in R-dimensional space.

present similar inapproximability results for scheduling with resource restrictions:

THEOREM 4.2: There is no approximation algorithm with rate less than $48/47 \approx 1.02$ or 1.5 for scheduling with resource restrictions with 2 or 4 resources, respectively, unless P=NP.

SANTA CLAUS. The problems of restricted assignment and unrelated scheduling are also studied with the reverse objective of maximizing the minimal machine load $\min_{i \in \mathcal{M}} \sum_{j \in \sigma^{-1}(i)} p_{ij}$. Usually these variants are described in a more game theoretical context with players instead of machines, goods instead of jobs, and values instead of processing times, and sometimes unrelated scheduling with the reverse objective is called the Santa Claus problem. In this work, we mostly stick to the scheduling notation but denote the variants of the considered problems with reverse objective as the Santa Claus version of the respective problem.

For the Santa Claus version of the restricted assignment problem a 13-approximation due to Annamalai, Kalaitzis and Svensson [6] is known, which has been improved to a rate of $6 + \varepsilon$ by both Cheng and Mao [32] and Davies, Rothvoss and Zhang [38]. PTAS results are known for the case without restrictions [146] and the inclusion-free interval case [97].

Our results can be directly transferred to the Santa Claus versions of the respective problems:

THEOREM 4.3: Unless P=NP, there is no PTAS for the Santa Claus version of scheduling with interval restrictions and no approximation algorithm with rate less than $47/46$ or 2 for the Santa Claus version of scheduling with resource restrictions with 2 or 4 resources, respectively.

ORGANIZATION. In the remainder of this section, we discuss further related literature and present preliminary considerations needed throughout the chapter. In Section 4.2, we present our results for RAI; in Section 4.3, we study the problem of RAR(R); and in Section 4.4, we present some open problems and possible future research directions.

FURTHER RELATED WORK. First note that if the number of machines is constant, there is a FPTAS already for unrelated scheduling [67]. Furthermore, for some broad overview concerning parallel machine scheduling with different kinds of restrictions in the context of online and approximation algorithms, we refer to the surveys by Lee et al. [107] and Leung and Li [109, 110].

We already discussed many variants of restricted assignment that admit a PTAS. In particular, Ou, Leung and Li [122] presented a PTAS for the hierarchical case; Epstein and Levin [46] and Muratore, Schwarz and Woeginger [119] for the nested case; and Schwarz [135] and

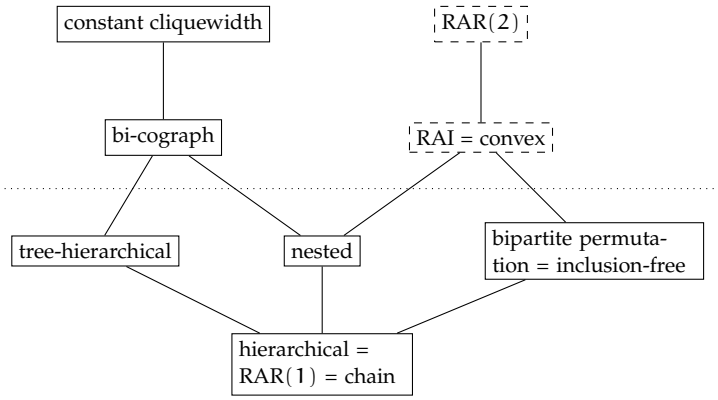


Figure 4.1: An overview of the inclusion structure of several of the discussed variants of restricted assignment. If two problems are connected, the upper includes the lower one. The dashed problems do not admit a PTAS, the remaining ones do. For the problems below the dotted line this was known before, and for the problems above the dotted line this is shown in the present work.

Khodamoradi et al. [97] for the inclusion-free case. Another case that has been studied in the literature is the tree-hierarchical case, where the machines can be arranged in a rooted tree such that for each job the set of eligible machines corresponds to a path starting at the root. It was shown to admit a PTAS by Epstein and Levin [46] and Schwarz [134]. It is not hard to see that all of the above cases contain the hierarchical case as a subcase, and that the tree-hierarchical, nested and inclusion-free case are distinct. There is, however, a variant admitting a PTAS that covers both the nested and the tree-hierarchical case: For each instance of the restricted assignment problem the corresponding incidence graph is a bipartite graph whose nodes are given by the jobs and machines and a job j is adjacent to a machine i if j is eligible on i. In the next chapter (and [82]), it is shown that there is PTAS for restricted assignment for the case that the clique- or rank-width of the incidence graph is constant. Furthermore, if the incidence graph is a bi-cograph the clique-width is well-known to be small and this case covers the nested and tree-hierarchical case. The inclusion-free case, on the other hand, is equivalent to the case that the incidence graph is a bipartite permutation graph [97] which does not have a bounded clique-width [22]. Note that RAR(1) or RAI are equivalent to the cases that the incidence graph is a chain [62] or convex graph [96], respectively. For an overview of the discussed cases, we refer to Figure 4.1.

Lastly, there has been a series of promising results in recent years concerning restricted assignment and variants thereof, and we highlight a few of them. In a breakthrough result, Svensson [137] showed that a certain integer linear program modeling the problem has an integrality gap of at most $33/17$, which implies an algorithm approximating the optimal objective value with rate $33/17 + \varepsilon$ for any

$\varepsilon > 0$ without producing a corresponding schedule. This has been improved by Jansen and Rohwedder [84] to a rate of $11/6$, and in [83] the same authors provide a quasi-polynomial approximation algorithm with rate $11/6 + \varepsilon$ that also outputs a corresponding schedule. For the special case of restricted assignment with only two distinct processing times (not counting ∞) an approximation algorithm due to Chakrabarty, Khanna and Li [24] with a rate slightly below 2 is known. Furthermore, the case in which the set of eligible machines for each job has cardinality at most 2 has been studied under the name of graph balancing. Ebenlendr, Krcál and Sgall [40] presented a 1.75-approximation for this case. Note that even if both of the above cases apply, there is no approximation algorithm with rate smaller than 1.5 [40] (unless P=NP). However, for this special case multiple authors found a fitting 1.5-approximation algorithm [25, 68, 124]. In a recent result, Jansen and Rohwedder [85] showed that in the graph balancing case the optimal objective value can be approximated with a rate slightly below 1.75.

PRELIMINARIES. In the following, we deal with satisfiability problems like the classical 3-SAT problem where a logical formula over variables $x_1, \ldots, x_n$ is given. The formula is a conjunction of clauses, each clause is a disjunction of three literals, and a literal is either a variable or its negation. The goal is to decide whether there is a satisfying truth assignment, that is, an assignment of the variables to the truth values "true" and "false", denoted by $\top$ and $\bot$, respectively, such that the formula evaluates to "true".

Nearly all the reductions in this work follow the same pattern: Given an instance I of the starting problem, we construct an instance I' of the variant of the restricted assignment problem considered in the respective case. For I', all job sizes are integral and upper bounded by some constant T such that the overall size of the jobs equals $|\mathcal{M}|T$. Obviously, if for such an instance a machine receives jobs with overall size more or *less* than T, the makespan of the schedule is greater than T. Then we show that that there exists a schedule with makespan T for I', if and only if I is a yes-instance. This rules out the existence of an approximation algorithm with rate smaller than $(T+1)/T$ and a PTAS in particular. Furthermore, for the Santa Claus version, approximation algorithms with rate smaller than $T/(T-1)$ are ruled out.

4.2 INTERVAL RESTRICTIONS

The sole goal of this section is to prove Theorem 4.1, that is, the non-existence of a PTAS for RAI (given P≠NP). Our starting point for the reduction is a satisfiability problem 3-SAT* that we tailor to our needs. We show that 3-SAT* is NP-hard via a straight forward reduction from the 1-in-3-SAT problem, which is well-know to be NP-

complete [130] and discussed in more detail below. Next, we provide a reduction from 3-SAT* to the classical restricted assignment problem (with arbitrary sets of eligible machines). This reduction introduces some of the needed gadgets and ideas for the main result. Lastly, we show how the reduction can be refined for RAI, and this is the most elaborate step.

STARTING POINT. An instance of 1-in-3-SAT is a conjunction of clauses with 3 literals each. Each clause is a formula depending on 3 literals that is satisfied if and only if exactly one of its literals takes the value $\top$. We call such formulas 1-in-3-clauses in the following and define 2-in-3-clauses correspondingly.

An instance of the problem 3-SAT* also is a conjunction of clauses with exactly 3 literals each. However, each of the clauses is either a 1-in-3-clause or a 2-in-3-clause and there are as many clauses of the first as of the second type. Furthermore, we require that each literal occurs *exactly twice*. In the following, we denote a 1-in-3-clause or 2-in-3-clause with literals z_1, z_2 and z_3 by $(z_1, z_2, z_3)_1$ or $(z_1, z_2, z_3)_2$, respectively.

To see that 3-SAT* is NP-hard, consider an instance of 1-in-3-SAT with n variables $x_1, \ldots, x_n$ and m clauses. We now construct an equivalent 3-SAT* instance. Let d_i be the number of times the variable x_i occurs in the given 1-in-3-SAT formula. For each variable x_i, we introduce new variables $x_{i,1}, \ldots, x_{i,d_i}$ and $y_{i,1}, \ldots, y_{i,d_i}$ along with clauses $(x_{i,1}, \neg x_{i,2}, y_{i,1})_2$, $\ldots$, $(x_{i,d_i-1}, \neg x_{i,d_i}, y_{i,d_i-1})_2$, $(x_{i,d_i}, \neg x_{i,1}, y_{i,d_i})_2$ and clauses $(y_{i,j}, \neg y_{i,j}, \neg y_{i,j})_1$ for each $j \in [d_i]$. Note that each variable $y_{i,j}$ has to take the value $\top$ in a satisfying assignment, due to the clause $(y_{i,j}, \neg y_{i,j}, \neg y_{i,j})_1$. The remaining clauses ensure, that for each i the variables $x_{i,1}, \ldots x_{i,d_i}$ have the same value in a satisfying assignment. Furthermore, for each of the clauses of the original problem, we introduce one 1-in-3-clause and one 2-in-3-clause. The 1-in-3-clauses are obtained by exchanging the j-th occurrence of each variable x_i with $x_{i,j}$. Moreover, the 2-in-3-clauses are obtained by copying the new 1-in-3-clauses, negating all the literals and turning them into a 2-in-3-clause. Hence, each 2-in-3-clauses evaluates to $\top$, if and only if its corresponding 1-in-3-clause does. It is not hard to verify the correctness of the reduction. Similar constructions are widely used, see, e.g., [141] or [31]. The remarkable aspect of the present construction lies in its symmetrical structure which helps to avoid additional dummy gadgets in the following reductions.

SIMPLE REDUCTION. In the following, we assume that an instance of 3-SAT* with n variables $x_1, \ldots, x_n$, m 1-in-3-clauses $C_1, \ldots, C_m$, and m 2-in-3-clauses $C_{m+1}, \ldots, C_{2m}$ is given. Note that we have $2m$ clauses with 3 literals each, and $4n$ occurring literals in total, hence $3m = 2n$. In addition to the ordering of the variables and clauses, we

Job	Size	Eligible Machines
$\texttt{CMach}_{i,s}$	111	$\texttt{CMach}_{i,s}$
$\texttt{CJob}^{\top}_{i,s'}$	100	$\texttt{CMach}_{i,1}$, $\texttt{CMach}_{i,2}$, $\texttt{CMach}_{i,3}$
$\texttt{CJob}^{\bot}_{i,s'}$	101	$\texttt{CMach}_{i,1}$, $\texttt{CMach}_{i,2}$, $\texttt{CMach}_{i,3}$
$\texttt{TJob}^{\top}_{j}$	100	$\texttt{TMach}_{j,1}$, $\texttt{TMach}_{j,2}$
$\texttt{TJob}^{\bot}_{j}$	102	$\texttt{TMach}_{j,1}$, $\texttt{TMach}_{j,2}$
$\texttt{VJob}^{\top}_{j,t}$	111	$\texttt{TMach}_{j,\lceil t/2\rceil}$, $\texttt{CMach}_{\kappa(j,t)}$
$\texttt{VJob}^{\bot}_{j,t}$	110	$\texttt{TMach}_{j,\lceil t/2\rceil}$, $\texttt{CMach}_{\kappa(j,t)}$

Table 4.1: The sizes and sets of eligible machines of the jobs in the simple reduction. The entry for $\texttt{CMach}_{i,s}$ marks the private load of the machine. The target makespan is given by $T = 322$.

fix an ordering of the literals belonging to each clause, and an ordering of the occurrences of each variable by assigning an index $t \in [4]$ to each of them. In particular, for each variable x_j, $t = 1, 2$ correspond to the first and second positive and $t = 3, 4$ to the first and second negative occurrence of x_j. Furthermore, let $\kappa : [n] \times [4] \to [2m] \times [3]$ be the bijection defined as follows: $\kappa(j, t) = (i, s)$ implies that the t-th occurrence of x_j is positioned in clause C_i on position s.

We now define the restricted assignment instance. For some of the machines, we introduce *private loads* which is a synonym for jobs of the corresponding size that have to be scheduled on the respective machine because its the only eligible one. The sizes and sets of eligible machines of the introduced jobs are presented in Table 4.1 and the target makespan is given by $T = 322$.

- For each clause C_i, there are three *clause machines* $\texttt{CMach}_{i,s}$ with $s \in [3]$ corresponding to its three literals, as well as three *clause jobs* $\texttt{CJob}^{\circ_{s'}}_{i,s'}$, with $s' \in [3]$ and $\circ_{s'} \in \{\top, \bot\}$. We have $\circ_1 = \top$ and $\circ_3 = \bot$, as well as $\circ_2 = \bot$ if C_i is a 1-in-3 clause, and $\circ_2 = \top$ otherwise. Furthermore, each clause machine has a private load of 111.

- For each variable x_j, there are two *truth assignment machines* $\texttt{TMach}_{j,q}$ with $q \in [2]$ corresponding to the positive ($q = 1$) and negative ($q = 2$) literal of x_j, as well as 2 *truth assignment jobs* $\texttt{TJob}^{\circ}_{j}$ with $\circ \in \{\top, \bot\}$.

- For each variable x_j, there are eight *variable jobs* $\texttt{VJob}^{\circ}_{j,t}$ with $t \in [4]$ and $\circ \in \{\top, \bot\}$ corresponding to the two occurrences of the positive ($t \in \{1, 2\}$) and negative ($t \in \{3, 4\}$) literal of x_j.

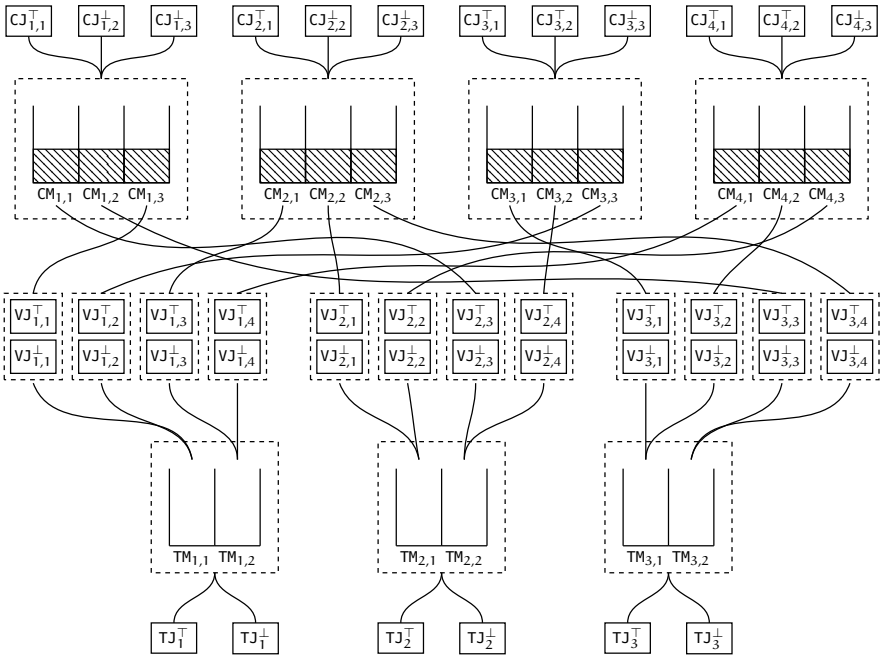


Figure 4.2: The restricted assignment instance constructed for a minimal example instance. The hatched rectangles represent private loads, and the connecting lines indicate eligibility. If these lines end at a dashed rectangle, the eligibility information concerns everything within the rectangle. We chose a short notation for the jobs and machines writing, e.g., $\mathtt{VJ}^{\circ}_{j,t}$ instead of $\mathtt{VJob}^{\circ}_{j,t}$.

EXAMPLE SIMPLE REDUCTION. The following formula is an instance of 3-SAT* with minimal size:

$$(\neg x_2, \neg x_3, x_1)_1 \wedge (\neg x_1, x_2, \neg x_3)_1 \wedge (x_3, \neg x_2, x_1)_2 \wedge (\neg x_1, x_3, x_2)_2$$

The formula is satisfied if all the variables take the value $\top$ and the corresponding restricted assignment instance is depicted in Figure 4.2.

ANALYSIS SIMPLE REDUCTION. First note:

Claim 4.4. The overall size of all the jobs is exactly $|\mathcal{M}|T$.

Proof. Since we have as many 1-in-3 as 2-in-3 clauses, the overall job size equals:

$$P = 6m \cdot 111 + m(3 \cdot 100 + 3 \cdot 101) + n(100 + 102 + 4 \cdot 111 + 4 \cdot 110)$$

Moreover, since $3m = 2n$, we have $P = 1932n = 322 \cdot 6n = |\mathcal{M}|T$. □

We will show that there is a satisfying truth assignment for the 3-SAT* instance if and only if there is a schedule in which each machine receives jobs with load exactly T.

For any job $\mathtt{Job}^{\circ}$ with $\circ \in \{\top, \bot\}$, we refer to $\circ$ as its *truth configuration* and say that $\mathtt{Job}^{\circ}$ has $\circ$-configuration. The rationale of the reduction is as follows: Each clause machine $\mathtt{CMach}_{i,s}$ should receive

Machine	Possible Schedules
$\mathtt{TMach}_{j,1}$	$\{\mathtt{TJob}_j^\top, \mathtt{VJob}_{j,1}^\top, \mathtt{VJob}_{j,2}^\top\}, \{\mathtt{TJob}_j^\bot, \mathtt{VJob}_{j,1}^\bot, \mathtt{VJob}_{j,2}^\bot\}$
$\mathtt{TMach}_{j,2}$	$\{\mathtt{TJob}_j^\top, \mathtt{VJob}_{j,3}^\top, \mathtt{VJob}_{j,4}^\top\}, \{\mathtt{TJob}_j^\bot, \mathtt{VJob}_{j,3}^\bot, \mathtt{VJob}_{j,4}^\bot\}$
$\mathtt{CMach}_{i,s}$ (1-in-3)	$\{\mathtt{VJob}_{\kappa^{-1}(i,s)}^\top, \mathtt{CJob}_{i,1}^\top\}, \{\mathtt{VJob}_{\kappa^{-1}(i,s)}^\bot, \mathtt{CJob}_{i,2/3}^\bot\}$
$\mathtt{CMach}_{i,s}$ (2-in-3)	$\{\mathtt{VJob}_{\kappa^{-1}(i,s)}^\top, \mathtt{CJob}_{i,1/2}^\top\}, \{\mathtt{VJob}_{\kappa^{-1}(i,s)}^\bot, \mathtt{CJob}_{i,3}^\bot\}$

Table 4.2: Each set indicates one of the possible job assignments for each machine in a schedule with makespan T. Note that for each clause job there are three possibilities.

exactly one variable job corresponding to the literal placed in position s in the clause. The truth configuration of this variable job should correspond to the truth value the variable contributes to the clause. To ensure that the jobs $\mathtt{VJob}_{j,t}^\top$ belonging to variable x_j contribute consistent truth values, the truth assignment jobs and machines are introduced. In the following, we sometimes talk about the truth assignment gadget and thus refer to these jobs and machines. Similarly, the clause machines and jobs are sometimes called the clause gadget.

Next, we present a sequence of easy claims concerning the properties of a schedule for the above instance with makespan T.

Claim 4.5. Each machine receives exactly 3 jobs (including private loads).

Proof. Since the overall size of the jobs is $|\mathcal{M}|T$, we know that each machine has to receive jobs with overall size $T = 322$. Each job or private load has a size of at least 100 and at most 111. □

Since each digit of each occurring size is upper bounded by 2, the above claim implies that there can be no carryover when adding up job sizes of jobs scheduled on each machine. Hence the digits of the numbers involved can be considered independently, e.g., there can be at most two jobs with a 1 in the third (or second) digit of its size scheduled on any machine. This together with the given job restrictions already implies:

Claim 4.6. Each truth assignment machine receives exactly one truth assignment and two variable jobs; and each clause machine receives exactly one clause and one variable job.

Claim 4.7. The jobs scheduled on a truth assignment or clause machine all have the same truth configuration (excluding private loads).

Claim 4.8. Let $j \in [n]$. The truth configurations of jobs scheduled on $\mathtt{TMach}_{j,1}$ and $\mathtt{TMach}_{j,2}$ are distinct.

The resulting possible schedules for each machine are summed up in Table 4.2, and Figure 4.3 depicts the resulting two possible schedules for each pair of truth assignment machines. Lastly, we have:

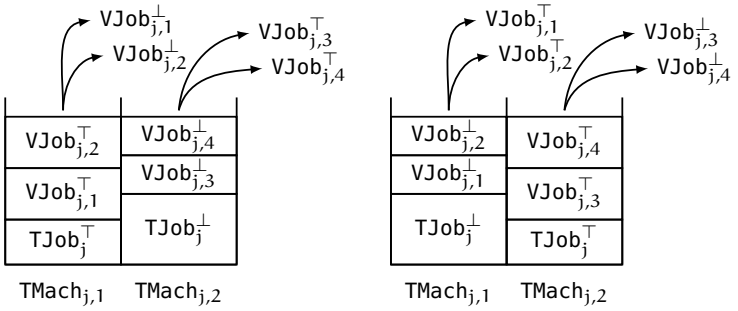


Figure 4.3: The truth assignment gadget: There are two possible schedules of the truth assignment machines $\mathtt{TMach}_{j,1}$ and $\mathtt{TMach}_{j,2}$ that already determine the schedule of the variable jobs.

Claim 4.9. For each $i \in [2m]$, the three clause machines corresponding to i receive exactly one variable job with $\top$-configuration if C_i is a 1-in-3-clause and exactly two such jobs if C_i is a 2-in-3-clause.

Proof. The overall load on each triplet of clause machines has to be $3T = 966$ and the private loads and clause jobs that have to be scheduled on the triplet have summed up load 635, in case of a 1-in-3-clause, and 634, in case of a 2-in-3-clause. The only other jobs eligible on the clause machines are variable jobs with size 111 in $\top$-configuration and 110 otherwise. This implies the claim. □

PROPOSITION 4.10. *There is a satisfying truth assignment for the given* 3-SAT* *instance if and only if there is a schedule with makespan* T *for the constructed restricted assignment instance.*

Proof. Given a schedule with makespan T, let $\mathtt{VJob}^{\circ_{j,t}}_{j,1}$ be the variable job scheduled on $\mathtt{CMach}_{\kappa(j,t)}$ (see Table 4.2) for each variable x_j and occurrence $t \in [4]$. We choose the truth value of x_j to be $\circ_{j,1}$. The variable x_j occurs exactly four times in the formulas, namely as a positive literal on the positions $\kappa(j,1)$ and $\kappa(j,2)$ and as a negative literal at position $\kappa(j,3)$ and $\kappa(j,4)$. Because of the above observations (see Figure 4.3), we know that $\circ_{j,2} = \circ_{j,1}$ and $\circ_{j,3} = \circ_{j,4} \neq \circ_{j,1}$. Hence, for each variable x_j and occurrence $t \in [4]$, the truth configuration $\mathtt{VJob}^{\circ_{j,t}}_{j,1}$ corresponds exactly to the truth value x_j contributes to the clause given by $\kappa(j,t)$. Lastly, for each clause C_i, there are exactly three variable jobs scheduled on the corresponding clause machines, and exactly one or two of these has $\top$-configuration, if C_i is a 1-in-3-clause or 2-in-3-clause respectively (Claim 4.9). Hence, C_i is satisfied.

Next, we consider the case that a satisfying truth assignment is given. For each variable x_j, let $\triangleleft_j$ be the corresponding truth value and $\triangleright_j$ its negation. We set $\circ_{j,t} = \triangleleft_j$ for $t \in \{1,2\}$ and $\circ_{j,t} = \triangleright_j$ for $t \in \{3,4\}$ and assign $\mathtt{VJob}^{\circ_{j,t}}_{j,1}$ to $\mathtt{CMach}_{\kappa(j,t)}$. All the other jobs are assigned as indicated by Table 4.2 and Figure 4.3. It is easy to verify, that all jobs are assigned and each machine has a load of T. □

The basic approach of using some kind of truth assignment and clause gadget for reductions in the context of restricted assignment and unrelated scheduling has been used before, see, e.g., [31, 40].

REFINED REDUCTION. When trying to adapt the above reduction to the more restricted problem of RAI, we obviously have less latitude when defining the restrictions. To deal with this, we introduce additional gadgets and encode much more information into the job sizes. The idea of the reduction can be described as follows. We arrange the truth assignment gadgets on the left and the clause gadgets on the right. Consider the case that a truth assignment decision is made in the left most truth assignment gadget. Information about this decision—called signal in the following—has to be passed on to the proper clause gadgets passing multiple other truth assignment and clause gadgets on the way. This signal in the simple reduction simply corresponds to a variable job that is to be scheduled on its corresponding clause machine, and in order to prevent interaction with other gadgets, we could encode information about the corresponding variable into the size of the variable job. However, this would lead to a super constant number of job sizes. To avoid this, we introduce a new gadget called the bridge and highway gadget. Very roughly speaking, the signal is passed on to the *highway* via *gateways*; the highway passes each following truth assignment gadget using *bridges* and carries the signal to the proper clauses. Next, we give a detailed description and analysis of the refined reduction.

We adopt all the machines and jobs introduced in the simple reduction, but change the sizes and sets of eligible machines and introduce additional jobs and machines as well as private loads for *every* machine. We introduce the following jobs and machines:

- For each $j \in [n]$ and $t \in [4]$, we introduce one *gateway machine* $\mathtt{GMach}_{j,t}$.
- For each $j \in [n]$, $t \in [4]$ and $j' \in \{j+1, \dots n\}$, we introduce two *bridge machines* $\mathtt{BMachIn}_{j,t,j'}$ and $\mathtt{BMachOut}_{j,t,j'}$. Furthermore, we introduce two *bridge jobs* $\mathtt{BJob}^{\top}_{j,t,j'}$ and $\mathtt{BJob}^{\bot}_{j,t,j'}$.
- For each $j \in [n]$, $t \in [4]$ and $j' \in \{j, \dots n\}$, we introduce two *highway jobs* $\mathtt{HJob}^{\top}_{j,t,j'}$ and $\mathtt{HJob}^{\bot}_{j,t,j'}$.

In order to define the intervals of eligible machines, we first need a total order of the machines. We partition the machines into blocks, define an internal order for each block, and then define an order of the blocks. Remember that $\kappa : [n] \times [4] \to [2m] \times [3]$ is a bijection indicating the positions of the occurrences of variables in the clauses. In particular, $\kappa(j,1) = (i,s)$ indicates that the first positive occurrence of variable x_j is in clause C_i on position s, and $\kappa(j,2)$, $\kappa(j,3)$, and $\kappa(j,4)$ indicate analogue information for the second positive, first negative, and second negative occurrence of x_j.

Job	First machine	Last machine
$\mathtt{CJob}^{\circ_s}_{i,s}$	$\mathtt{CMach}_{i,1}$	$\mathtt{CMach}_{i,3}$
$\mathtt{TJob}^{\circ}_{j}$	$\mathtt{TMach}_{j,1}$	$\mathtt{TMach}_{j,2}$
$\mathtt{VJob}^{\circ}_{j,t}$	$\mathtt{TMach}_{j,\lceil t/2 \rceil}$	$\mathtt{GMach}_{j,t}$
$\mathtt{BJob}^{\circ}_{j,t,j'}$	$\mathtt{BMachIn}_{j,t,j'}$	$\mathtt{BMachOut}_{j,t,j'}$
$\mathtt{HJob}^{\circ}_{j,t,j'}$	$\mathtt{BMachOut}_{j,t,j'}$, if $j' > j$, $\mathtt{GMach}_{j,t}$, if $j' = j$	$\mathtt{BMachIn}_{j,t,j'+1}$ if $j' < n$, $\mathtt{CMach}_{\kappa(j,t)}$, if $j' = n$

Table 4.3: The sets of eligible machines for each job or job type, defined by the first and last eligible machine in the ordering. Note that in case of the highway jobs all four combinations of first and last machine are possible.

- For each $j \in [n]$, we have a truth assignment block $\mathcal{T}_j$ containing the truth assignment machines $\mathtt{TMach}_{j,1}$ and $\mathtt{TMach}_{j,2}$ in this order.
- For each $i \in [2m]$, we have a clause block $\mathcal{C}_i$ containing the clause machines $\mathtt{CMach}_{i,s}$ for each $s \in [3]$ and ordered increasingly by s.
- For each $j \in [n]$, we have a successor block $\mathcal{S}_j$ containing the gateway machines $\mathtt{GMach}_{j,t}$ for each $t \in [4]$ and the bridge machines $\mathtt{BMachOut}_{j',t,j}$ for each $t \in [4]$ and $j' < j$. For each machine, we define an index, namely $\kappa(j,t)$ for $\mathtt{GMach}_{j,t}$ and $\kappa(j',t)$ for $\mathtt{BMachOut}_{j',t,j}$, and order the machines by the *decreasing* lexicographical ordering of their indices. For example, if $\mathtt{BMachOut}_{j_1,t_1,j}, \mathtt{BMachOut}_{j_2,t_2,j}, \mathtt{GMach}_{j,t_3} \in \mathcal{S}_j$ and $\kappa(j_1,t_1) = (1,2)$, $\kappa(j_2,t_2) = (1,1)$ and $\kappa(j,t_3) = (2,3)$, then $\mathtt{GMach}_{j,t_3}$ precedes $\mathtt{BMachOut}_{j_1,t_1,j}$ which in turn precedes $\mathtt{BMachOut}_{j_2,t_2,j}$.
- For each $j \in [n]$ with $j > 1$, we have a predecessor block $\mathcal{P}_j$ containing the bridge machines $\mathtt{BMachIn}_{j',t,j}$ for each $t \in [4]$ and $j' < j$. Machine $\mathtt{BMachIn}_{j',t,j}$ has index $\kappa(j',t)$ and the machines are ordered by the *increasing* lexicographical ordering of their indices.

The blocks are ordered as follows:

$$(\mathcal{T}_1, \mathcal{S}_1, \mathcal{P}_2, \mathcal{T}_2, \mathcal{S}_2, \ldots, \mathcal{P}_n, \mathcal{T}_n, \mathcal{S}_n, \mathcal{C}_1, \ldots, \mathcal{C}_{2m})$$

The sets of eligible machines are specified in Table 4.3 and the job sizes in Table 4.4. Note that we have prioritized comprehensibility over small sizes. For instance, it is not hard to see that the columns in Table 4.4 corresponding to the highway and clause jobs could be deleted and the reduction would still work.

	#		B	H	C	T	V	V	V	V	
$\mathtt{CJob}^{\top}_{i,s}$	$3m = 2n$	1	0	0	1	0	0	0	0	0	0
$\mathtt{CJob}^{\bot}_{i,s}$	$3m = 2n$	1	0	0	1	0	0	0	0	0	1
$\mathtt{TJob}^{\top}_{j}$	n	1	0	0	0	1	0	0	0	0	2
$\mathtt{TJob}^{\bot}_{j}$	n	1	0	0	0	1	0	0	0	0	0
$\mathtt{VJob}^{\top}_{j,1}$	n	1	0	0	0	0	1	0	0	0	0
$\mathtt{VJob}^{\top}_{j,2}$	n	1	0	0	0	0	0	1	0	0	0
$\mathtt{VJob}^{\top}_{j,3}$	n	1	0	0	0	0	0	0	1	0	0
$\mathtt{VJob}^{\top}_{j,4}$	n	1	0	0	0	0	0	0	0	1	0
$\mathtt{VJob}^{\bot}_{j,1}$	n	1	0	0	0	0	1	0	0	0	1
$\mathtt{VJob}^{\bot}_{j,2}$	n	1	0	0	0	0	0	1	0	0	1
$\mathtt{VJob}^{\bot}_{j,3}$	n	1	0	0	0	0	0	0	1	0	1
$\mathtt{VJob}^{\bot}_{j,4}$	n	1	0	0	0	0	0	0	0	1	1
$\mathtt{BJob}^{\top}_{j,t,j'}$	$2n(n-1)$	1	1	0	0	0	0	0	0	0	0
$\mathtt{BJob}^{\bot}_{j,t,j'}$	$2n(n-1)$	1	1	0	0	0	0	0	0	0	1
$\mathtt{HJob}^{\top}_{j,t,j'}$	$2n(n+1)$	1	0	1	0	0	0	0	0	0	1
$\mathtt{HJob}^{\bot}_{j,t,j'}$	$2n(n+1)$	1	0	1	0	0	0	0	0	0	0
$\mathtt{CMach}_{i,s}$	$6m = 4n$	1	1	0	0	1	1	1	1	1	1
$\mathtt{TMach}_{j,1}$	n	0	1	1	1	0	0	0	1	1	0
$\mathtt{TMach}_{j,2}$	n	0	1	1	1	0	1	1	0	0	0
$\mathtt{GMach}_{j,1}$	n	1	1	0	1	1	0	1	1	1	1
$\mathtt{GMach}_{j,2}$	n	1	1	0	1	1	1	0	1	1	1
$\mathtt{GMach}_{j,3}$	n	1	1	0	1	1	1	1	0	1	1
$\mathtt{GMach}_{j,4}$	n	1	1	0	1	1	1	1	1	0	1
$\mathtt{BMachIn}_{j,t,j'}$	$2n(n-1)$	1	0	0	1	1	1	1	1	1	1
$\mathtt{BMachOut}_{j,t,j'}$	$2n(n-1)$	1	0	0	1	1	1	1	1	1	1
Makespan T		3	1	1	1	1	1	1	1	1	2

Table 4.4: Table of job and machine types with job sizes and private loads and the makespan. The second column states the number of jobs and machines of the respective types. Each horizontal sequence of numbers following the second column indicates the size of the respective job or private load. Each of the corresponding columns serves a function in the reduction: the first bounds the number of jobs on each machines; the following eight implement restrictions for the bridge, highway, clause, truth assignment and variable jobs; and the last encodes truth values.

EXAMPLE REFINED REDUCTION. We consider the same formula as we did for the simple reduction, that is:

$$(\neg x_2, \neg x_3, x_1)_1 \wedge (\neg x_1, x_2, \neg x_3)_1 \wedge (x_3, \neg x_2, x_1)_2 \wedge (\neg x_1, x_3, x_2)_2$$

The values of κ for the occurrences of the first two variables together with the resulting increasing lexicographical ordering is depicted in Table 4.5. Furthermore, in Figure 4.4 the truth assignment as well as the bridge and highway gadget for the first two variables are depicted.

(j,t)	(1,1)	(1,2)	(1,3)	(1,4)	(2,1)	(2,2)	(2,3)	(2,4)
$\kappa(j,t)$	(1,3)	(3,3)	(2,1)	(4,1)	(2,2)	(4,3)	(1,1)	(3,2)
Ordering	(2,3)	(1,1)	(1,3)	(2,1)	(2,4)	(1,2)	(1,4)	(2,2)

Table 4.5: The occurrences of the first two values in the clauses and the resulting increasing lexicographical ordering of the occurrences.

ANALYSIS REFINED REDUCTION. We have:

Claim 4.11. The overall size of the jobs is exactly $|\mathcal{M}|T$.

Proof. This can be verified by basic arithmetic using Table 4.4. For simplicity, this can also be done digit by digit. We look at the last digit as an example. Note that we have $4n^2 + 6n$ machines and the last digit of the makespan is 2. Summing up the last digits of the job sizes, on the other hand, yields $2n + 2n + n + n + n + n + 2n(n-1) + 2n(n+1) + 4n + n + n + n + n + 2n(n-1) + 2n(n-1) = 8n^2 + 12n$. □

Like for the simple reduction, we proof a sequence of easy claims concerning the properties of a schedule for the constructed instance with makespan T.

Claim 4.12. Each machine receives exactly 4 jobs if it is a truth assignment machine and exactly 3 jobs otherwise (including private loads).

Proof. This follows directly from Claim 4.11 and the job sizes defined in Table 4.4. □

Since each machine receives at most 4 jobs and each digit in the job sizes is bounded by 2, we may consider each digit of the involved numbers independently, e.g., if two jobs and the makespan have a 1 at the ℓ-th digit, we already know that these jobs cannot be scheduled on the same machine. This already implies a series of claims:

Claim 4.13. The jobs $\texttt{TJob}_j^\top$ and $\texttt{TJob}_j^\bot$ can exclusively be scheduled on $\texttt{TMach}_{j,1}$ and $\texttt{TMach}_{j,2}$, for each $j \in [n]$, and each of the two machines receives exactly one of the two jobs.

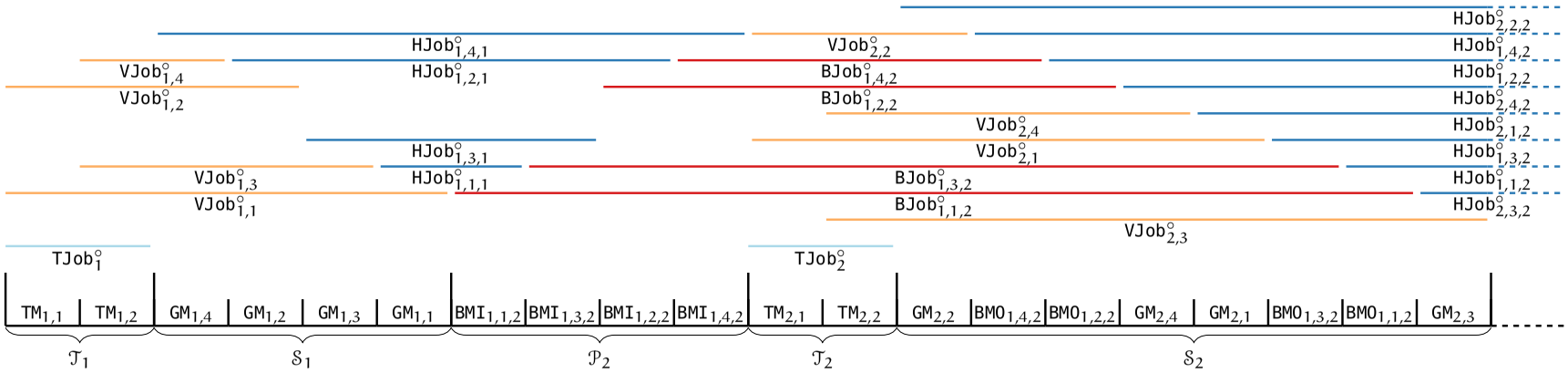


Figure 4.4: The truth assignment as well as the bridge and highway gadget for the first two variables of an example instance. The colored lines mark the intervals of eligible machines for the respective jobs. In the picture, we use a more compact notation for the machines, and write, e.g., $\texttt{TM}_{1,1}$ instead of $\texttt{TMach}_{1,1}$.

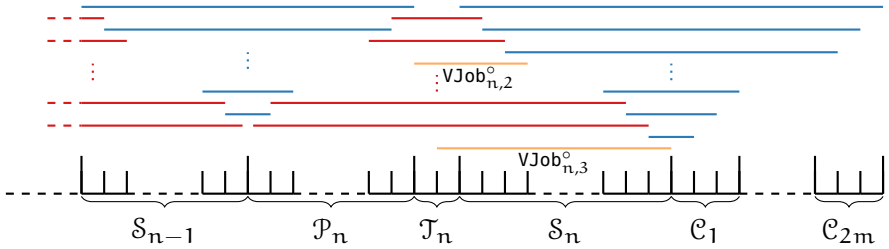


Figure 4.5: The bridge and highway gadget. The intervals of eligible machines of highway, bridge and variable jobs are depicted in blue, red and orange, respectively. In this example, variable x_n occurs for the second time in its positive form in the last clause at the first position, and for the first time in its negative form in the first clause at the first position.

Claim 4.14. The jobs $\texttt{VJob}^{\top}_{j,t}$ and $\texttt{VJob}^{\bot}_{j,t}$ can exclusively be scheduled on $\texttt{TMach}_{j,\lceil t/2\rceil}$ and $\texttt{GMach}_{j,t}$, for each $j \in [n]$ and $t \in [4]$, and each of the two machines receives exactly one of the two jobs.

Claim 4.15. Bridge jobs can exclusively be scheduled on bridge machines and each bridge machine receives exactly one bridge job.

Claim 4.16. Highway jobs can exclusively be scheduled on bridge, gateway and clause machines and each such machine receives exactly one highway job.

Claim 4.17. Each clause machine $\texttt{CMach}_{i,s}$ receives exactly one of the corresponding clause jobs $\texttt{CJob}^{\circ_{s'}}_{i,s}$, with $s' \in [3]$.

At this point, we already know that variable (and truth assignment) jobs can exclusively be scheduled on the first or last machine of their respective interval of eligible machines. The next step is to show that the same holds for highway and bridge jobs. To do so, the ordering of the bridge and highway machines is of critical importance.

Claim 4.18. The jobs $\texttt{BJob}^{\top}_{j,t,j'}$ and $\texttt{BJob}^{\bot}_{j,t,j'}$ can exclusively be scheduled on $\texttt{BMachIn}_{j,t,j'}$ and $\texttt{BMachOut}_{j,t,j'}$, for each $j \in [n]$, $j' \in \{j+1,\dots,n\}$ and $t \in [4]$, and each of the two machines receives exactly one of the two jobs.

Proof. The claim can be proved with a simple inductive argument: Let $j' \in \{2,\dots,n\}$ and, furthermore, $(j_\ell, t_\ell) \in [j'-1] \times [4]$ denote the ℓ-th element from $[j'-1] \times [4]$ when ordering the pairs $(j,t) \in [j'-1] \times [4]$ by the increasing lexicographical ordering of the pairs $\kappa(j,t)$. Considering the ordering of the machines and the job restrictions, $\texttt{BJob}^{\top}_{j_1,t_1,j'}$ and $\texttt{BJob}^{\bot}_{j_1,t_1,j'}$ are the only bridge jobs that can be scheduled on $\texttt{BMachIn}_{j_1,t_1,j'}$ and $\texttt{BMachOut}_{j_1,t_1,j'}$ (see Figure 4.5). Hence, the claim has to hold for (j_1,t_1). But then again $\texttt{BJob}^{\top}_{j_2,t_2,j'}$ and $\texttt{BJob}^{\bot}_{j_2,t_2,j'}$ are the only remaining bridge jobs that can be scheduled on $\texttt{BMachIn}_{j_2,t_2,j'}$ and $\texttt{BMachOut}_{j_2,t_2,j'}$, and so on. □

Machine	Possible Schedule
$\mathtt{TMach}_{j,1}$	$\{\mathtt{TJob}_j^\top, \mathtt{VJob}_{j,1}^\top, \mathtt{VJob}_{j,2}^\top\}, \{\mathtt{TJob}_j^\bot, \mathtt{VJob}_{j,1}^\bot, \mathtt{VJob}_{j,2}^\bot\}$
$\mathtt{TMach}_{j,2}$	$\{\mathtt{TJob}_j^\top, \mathtt{VJob}_{j,3}^\top, \mathtt{VJob}_{j,4}^\top\}, \{\mathtt{TJob}_j^\bot, \mathtt{VJob}_{j,3}^\bot, \mathtt{VJob}_{j,4}^\bot\}$
$\mathtt{GMach}_{j,t}$	$\{\mathtt{VJob}_{j,t}^\top, \mathtt{HJob}_{j,t,j}^\top\}, \{\mathtt{VJob}_{j,t}^\bot, \mathtt{HJob}_{j,t,j}^\bot\}$
$\mathtt{BMachIn}_{j,t,j'}$	$\{\mathtt{BJob}_{j,t,j'}^\top, \mathtt{HJob}_{j,t,j'-1}^\top\}, \{\mathtt{BJob}_{j,t,j'}^\bot, \mathtt{HJob}_{j,t,j'-1}^\bot\}$
$\mathtt{BMachOut}_{j,t,j'}$	$\{\mathtt{BJob}_{j,t,j'}^\top, \mathtt{HJob}_{j,t,j'}^\top\}, \{\mathtt{BJob}_{j,t,j'}^\bot, \mathtt{HJob}_{j,t,j'}^\bot\}$
$\mathtt{CMach}_{i,s}$ (1-in-3)	$\{\mathtt{CJob}_{i,1}^\top, \mathtt{HJob}_{\kappa^{-1}(i,s),n}^\top\}, \{\mathtt{CJob}_{i,2}^\bot, \mathtt{HJob}_{\kappa^{-1}(i,s),n}^\top\},$ $\{\mathtt{CJob}_{i,3}^\bot, \mathtt{HJob}_{\kappa^{-1}(i,s),n}^\bot\}$
$\mathtt{CMach}_{i,s}$ (2-in-3)	$\{\mathtt{CJob}_{i,1}^\top, \mathtt{HJob}_{\kappa^{-1}(i,s),n}^\top\}, \{\mathtt{CJob}_{i,2}^\top, \mathtt{HJob}_{\kappa^{-1}(i,s),n}^\top\},$ $\{\mathtt{CJob}_{i,3}^\bot, \mathtt{HJob}_{\kappa^{-1}(i,s),n}^\bot\}$

Table 4.6: For each machine there are only few possible jobs that may be assigned to it in a schedule with makespan T. Each set corresponds to one of the possible schedules.

Claim 4.19. The jobs $\mathtt{HJob}_{j,t,j'}^\top$ and $\mathtt{HJob}_{j,t,j'}^\bot$ can exclusively be scheduled on machine X and Y, for each $j \in [n]$, $j' \geqslant j$, and $t \in [4]$; where $X = \mathtt{BMachOut}_{j,t,j'}$ if $j' > j$, and $X = \mathtt{GMach}_{j,t}$ otherwise, and $Y = \mathtt{BMachIn}_{j,t,j'+1}$ if $j' < n$, and $Y = \mathtt{CMach}_{\kappa(j,t)}$ otherwise. Furthermore, each of the two machines receives exactly one of the two jobs.

Proof. We can use the same argument (with reversed orderings) as we did in the last claim. It is only slightly more complicated, because more machine types are involved. □

Summing up, each job except for clause jobs may only be scheduled on the first or last machine of their interval of eligible machines, and each of these machines receives either the respective job in $\top$- or $\bot$-configuration. Considering this distribution of the jobs and the last digit of the size vectors, we get the following two claims:

Claim 4.20. For any machine, the jobs assigned to this machine all have the same truth configuration (excluding private loads).

Claim 4.21. For each $i \in [2m]$, the three clause machines corresponding to i receive exactly one highway job with $\top$-configuration, if C_i is a 1-in-3-clause, and exactly two such jobs, if C_i is a 2-in-3-clause.

The former property together with the possible job distribution determined so far implies that there are only few possible schedules for each machine. We summarize these schedules in Table 4.6. Furthermore, we can infer that the truth assignment gadget works essentially the same as before (see Figure 4.3):

Claim 4.22. Let $j \in [n]$. The truth configuration of any job scheduled on $\mathtt{TMach}_{j,1}$ is distinct from the truth configuration of any job scheduled on $\mathtt{TMach}_{j,2}$.

Lastly, we can show that the bridge and highway gadget works as well:

Claim 4.23. Let $j \in [n]$ and $t \in [4]$. The variable job scheduled on $\mathtt{TMach}_{j,\lceil t/2 \rceil}$ and the highway job scheduled on $\mathtt{CMach}_{\kappa(j,t)}$ have the same truth configuration.

Proof. Note that the truth configuration of the variable job scheduled on $\mathtt{GMach}_{j,t}$ compared with the one of the variable job scheduled on $\mathtt{TMach}_{j,\lceil t/2 \rceil}$ is reversed. Hence, the highway job scheduled on $\mathtt{GMach}_{j,t}$ also has the reversed truth-configuration while the highway job that is passed on again has the original truth-configuration. This argument can be repeated with the bridge and highway jobs in the following, yielding the asserted claim. □

Using the above claims, we can conclude the proof of Theorem 4.1 via the following Lemma:

LEMMA 4.24. *There is a satisfying truth assignment for the given* 3-SAT* *instance, if and only if there is a schedule with makespan* T *for the constructed* RAI *instance.*

Proof. Given a schedule with makespan T, let $\mathtt{HJob}^{\circ_{j,t}}_{j,t,n}$ be the highway job scheduled on $\mathtt{CMach}_{\kappa(j,t)}$ for each variable x_j and occurrence $t \in [4]$ (see Table 4.6). We choose the truth value of x_j to be $\circ_{j,1}$. Considering the distribution of jobs on the truth assignment machines (see Table 4.6), as well as Claim 4.22 and 4.23, we know that for each variable x_j and occurrence $t \in [4]$, the truth configuration $\circ_{j,t}$ corresponds exactly to the truth value x_j contributes to the clause given by $\kappa(j,t)$. Furthermore, we know that for each clause C_i, there are exactly three variable jobs scheduled on the corresponding clause machines, and exactly one or two of these has $\top$-configuration, if C_i is a 1-in-3-clause or 2-in-3-clause, respectively (Claim 4.21). Hence, C_i is satisfied.

Next, we consider the case that there is a satisfying truth assignment. Let $\triangleleft_j$ be the corresponding truth value of variable x_j and $\triangleright_j$ its negation. We set $\triangle_{j,t} = \triangleleft_j$ for $t \in \{1,2\}$ and $\triangle_{j,t} = \triangleright_j$ for $t \in \{3,4\}$ and assign $\mathtt{HJob}^{\triangle_{j,t}}_{j,t,n}$ to $\mathtt{CMach}_{\kappa(j,t)}$. Let $\bigtriangledown_{jt}$ be the negation of $\triangle_{jt}$. All the other jobs are assigned as indicated by the claims and Table 4.6 in particular: Each machine receives its private load; $\mathtt{CMach}_{\kappa(j,t)}$ additionally receives one of the eligible remaining clause jobs with $\triangle_{j,t}$-configuration (this can be done because the truth assignment is satisfying); $\mathtt{BMachOut}_{j,t,j'}$ receives $\mathtt{HJob}^{\bigtriangledown_{j,t}}_{j,t,j'}$ and $\mathtt{BJob}^{\bigtriangledown_{j,t}}_{j,t,j'}$; $\mathtt{BMachIn}_{j,t,j'}$ receives $\mathtt{HJob}^{\triangle_{j,t}}_{j,t,j'-1}$ and $\mathtt{BJob}^{\triangle_{j,t}}_{j,t,j'}$; $\mathtt{GMach}_{j,t}$ receives $\mathtt{HJob}^{\bigtriangledown_{j,t}}_{j,t,j}$ and $\mathtt{VJob}^{\bigtriangledown_{j,t}}_{j,t}$; $\mathtt{TMach}_{j,1}$ receives $\mathtt{VJob}^{\triangle_{j,1}}_{j,1}$, $\mathtt{VJob}^{\triangle_{j,2}}_{j,2}$ and $\mathtt{TJob}^{\triangle_{j,1}}_{j}$; and $\mathtt{TMach}_{j,2}$ receives $\mathtt{VJob}^{\triangle_{j,3}}_{j,3}$, $\mathtt{VJob}^{\triangle_{j,4}}_{j,4}$ as well as $\mathtt{TJob}^{\triangle_{j,3}}_{j}$. It is easy to verify, that all jobs are assigned and each machine has a load of T. □

4.3 RESOURCE RESTRICTIONS

In this section, we first present some preliminary observations concerning RAR(R) and discuss the relationship of the problem with RAI and LRS(D). Next, we revisit established reductions for the restricted assignment problem and show that they can be modeled with only few resources. This already gives the result for 4 resources in Theorem 4.2. Lastly, we study the cases with 2 and 3 resources. We first give a reduction for $R = 3$ and then refine the result to work for $R = 2$ as well thereby concluding the proof of Theorem 4.2 and Theorem 4.3.

PRELIMINARIES. Recall that in the problem of scheduling with resource restrictions with R resources (RAR(R)), a set $\mathcal{R}$ of R (renewable) resources is given, each machine i has a resource capacity $c_r(i)$ and each job j has a resource demand $d_r(j)$ for each $r \in \mathcal{R}$. Job j is eligible on machine i, if $d_r(j) \leqslant c_r(i)$ for each resource r. We allow arbitrary real values for the capacities and demands but it is not hard to see that relatively small integer values suffice. Indeed, given an instance of RAR(R), we may perform the following two steps: First, we increase for each job j and each resource r the demand $d_r(j)$ to the smallest value included in $\{c_r(i) \mid i \in \mathcal{M}, c_r(i) \geqslant d_r(j)\}$ (if this set is empty the job cannot be processed anywhere). Afterwards, there are at most $m = |\mathcal{M}|$ distinct demand or capacity values for each resource, and we can change the smallest value to 1 the second to 2 and so on. This yields an instance with the same restrictions and the property that all the capacities and demands are included in $[m]$.

Technically, there are two versions of the problem RAR(R) depending on whether the resources, demands and capacities are explicitly given or not. In the second variant recognition is an issue that we do not address in this work since our results work for both versions of the problem. However, note that the proof of the following lemma gives some intuition concerning this:

LEMMA 4.25. *Each restricted assignment instance with m machines is also a* RAR(m) *instance; and for each $m \in \mathbb{N}$ there is a* RAR(m) *instance with m machines (and m jobs) that is not a* RAR(R) *instance for any $R < m$.*

Proof. Given a restricted assignment instance with m machines, we define resources, demands and capacities that model the given restrictions. First, we identify each machine with a resource, that is, we set $\mathcal{R} = \mathcal{M}$. Furthermore, we set the capacities of machine $i \in \mathcal{M}$ concerning resource $r \in \mathcal{R}$ to be $c_r(i) = 1$ if $r \neq i$ and $c_i(i) = 0$; and the demand of job $j \in \mathcal{J}$ concerning r to be $d_r(i) = 0$ if $r \in \mathcal{M}(j)$, and $d_r(i) = 1$ otherwise. It is easy to check that for each job j and machine i, we have $d_r(j) \leqslant c_r(i)$ for each resource r, if and only if $i \in \mathcal{M}(j)$.

The next goal is to construct a simple RAR(m) instance for each $m \in \mathbb{N}$. We first collect some simple observations. Given some instance of RAR(R) with $j \in \mathcal{J}$ and $i' \in \mathcal{M} \setminus \mathcal{M}(j)$, we know that there is

a resource $r(j,i') \in \mathcal{R}$ such that $c_{r(j,i')}(i') < d_{r(j,i')}(j)$ (otherwise $i' \in \mathcal{M}(j)$). One could say that $r(j,i')$ separates i' from j or $\mathcal{M}(j)$. On the other hand, we know that $d_r(j) \leqslant c_r(i)$ for each $i \in \mathcal{M}(j)$ and $r \in \mathcal{R}$. Let $i \in \mathcal{M}(j)$. Now, consider the case that we have another job j' with $i' \in \mathcal{M}(j')$ and $i \in \mathcal{M} \setminus \mathcal{M}(j')$. Then we have $r(j,i') \neq r(j',i)$, because otherwise:

$$\begin{aligned} &d_{r(j',i)}(j) \leqslant c_{r(j',i)}(i) < d_{r(j',i)}(j') \leqslant c_{r(j',i)}(i') \\ &= c_{r(j,i')}(i') < d_{r(j,i')}(j) = d_{r(j',i)}(j) \end{aligned}$$

Using this insight, we construct the instance as follows: We set $\mathcal{M} = [m]$ and $\mathcal{J} = \left\{ j \subseteq \mathcal{M} \,\middle|\, |j| = m-1 \right\}$ with $\mathcal{M}(j) = j$. We may assume unit processing times. This instance has exactly m jobs and machines and we know that it is a RAR(m) instance because of the first part of the proof. Now, given any resources $\mathcal{R}$ along with capacities and demands that model the restrictions for the above instance, we show that $|\mathcal{R}| \geqslant m$. For each $j \in \mathcal{J}$ let $i_j \in \mathcal{M}$ be the single machine that is restricted to process j, i.e., $i_j \in \mathcal{M} \setminus \mathcal{M}(j)$. This implies $\mathcal{M} = \{i_j \,|\, j \in \mathcal{J}\}$ and due to the above observation, we have $r(j,i_j) \neq r(j',i_{j'})$ for each pair of distinct jobs $j, j' \in \mathcal{J}$. Hence, $\{r(j,i_j) \,|\, j \in \mathcal{J}\} = \mathcal{R}' \subseteq \mathcal{R}$ and $|\mathcal{R}'| = m$ concluding the proof. □

The relationship between scheduling with resource and interval restrictions is discussed in the following lemma:

LEMMA 4.26. *Each* RAR(1) *instance is also a* RAI *instance and there is a* RAI *instance that is not a* RAR(1) *instance. Moreover, each* RAI *instance is also a* RAR(2) *instance and there is a* RAR(2) *instance that is not a* RAI *instance. With a slight abuse of notation, we may write:* RAR(1) ⊂ RAI ⊂ RAR(2).

Proof. Given an instance of RAR(1), we may sort the machines based on their capacity values decreasingly. Than each job j that can be processed on any machine, can also be processed on any predecessor of this machine. Hence, $\mathcal{M}(j)$ corresponds to an interval of machines starting with the first machine. On the other hand, consider an instance with two machines and two jobs. The first job is (exclusively) eligible on the first machine and the second one on the second. This instance is a RAI but not a RAR(2) instance.

Given an instance of RAI, we may assume w.l.o.g. $\mathcal{M} = [m]$ and that the ordering of the machines is the natural ordering. We set $\mathcal{R} = [2]$. Furthermore, for each machine i, we set $c(i) = (i, (m+1) - i)$; and for each job j with $\mathcal{M}(j) = \{\ell, \ldots, r\}$, we set $d(j) = (\ell, (m+1) - r)$ (see Figure 4.6). If $i \in \mathcal{M}(j)$, we have $c_1(i) = i \geqslant \ell = d_1(j)$ and $c_2(i) = (m+1) - i \geqslant (m+1) - r = d_2(j)$; and if $i \in \mathcal{M} \setminus \mathcal{M}(j)$, we have either $i < \ell$ or $i > r$ which implies $c_1(i) < d_1(j)$ or $c_2(i) < d_2(j)$ respectively.

Lastly, we construct an instance of RAR(2) that is not a RAI instance. Let $\mathcal{M} = [4]$, $\mathcal{R} = [2]$ and $\mathcal{J} = \big\{\{1,2,3,4\},\{1,2\},\{1,3\},\{1,4\}\big\}$. We may

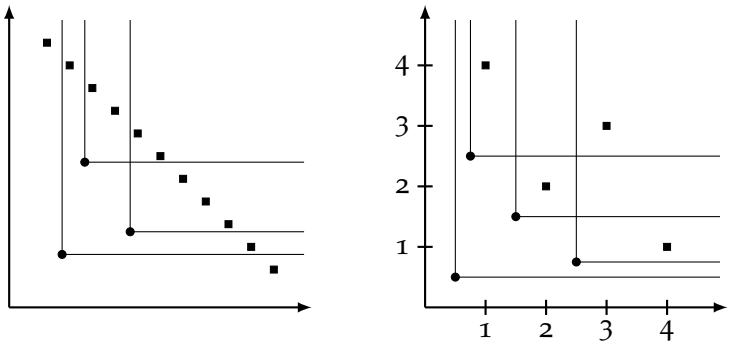


Figure 4.6: The left picture visualizes that each RAI instance can be seen as a RAR(2) instance and the right one depicts an RAR(2) instance that is not a RAI instance. In both pictures, each dimension corresponds to a resource, the squares mark the capacities of machines and the circles the demands of jobs. If the capacity of a machine is at least as big as the demand of a job in both dimension, the job is eligible on the machine.

Machine	Capacity	Job	Demand
1	$(3,3)$	$\{1,2,3,4\}$	$(0.5,0.5)$
2	$(4,1)$	$\{1,2\}$	$(2.5,0.75)$
3	$(2,2)$	$\{1,3\}$	$(1.5,1.5)$
4	$(1,4)$	$\{1,4\}$	$(0.75,2.5)$

Table 4.7: A simple example of a RAR(2) instance that is not a RAI instance. Note that all demands could be rounded up to the next integer value without changing the construction.

assume unit job sizes. The resource capacities and demands are given in Table 4.7 and the construction is illustrated in Figure 4.6. It is easy to see that $\mathcal{M}(j) = j$ for each $j \in \mathcal{J}$. In any total ordering of the machines in which each job is eligible on consecutive machine, the machine 1 has to be a direct neighbor of 2, 3 and 4. This is not possible. □

We already mentioned in the introduction that there is a close relationship between scheduling with resource restrictions and low rank unrelated scheduling. Remember that in the rank D unrelated scheduling problem (LRS(D)) the processing time matrix $(p_{ij})_{i\in\mathcal{M},j\in\mathcal{J}}$ has a rank of at most D, or, equivalently, there is a D dimensional size or speed vector $s(j)$ or $v(i)$ for each job j or machine i respectively, and the processing time p_{ij} is given by $\sum_{d\in[D]} s_d(j)v_d(i)$. It is easy to construct for any $m \in \mathbb{N}$ a RAR(1) instance that is a LRS(m) instance as well: The instance with $\mathcal{J} = \mathcal{M} = [m]$, $\mathcal{R} = \{1\}$, $d_1(k) = c_1(k) = k$ for each $k \in [m]$ and unit processing times suffices (assuming that the number ∞ is interpreted as some sufficiently big number). On the

other hand, any RAR(R) instance can be approximated with arbitrary precision by LRS(R + 1) instances in the following sense:

LEMMA 4.27. *Let* I *be a* RAR(R) *instance. For any* $\varepsilon, K > 0$*, there is a* LRS(R + 1) *instance* I′ *with the same jobs and machines and the following property: Let* p'_{ij} *be the processing time of job* j *on machine* i *in instance* I′*. We have* $p_j \leqslant p'_{ij} \leqslant p_j + \varepsilon$ *if* $i \in \mathcal{M}(j)$ *and* $p'_{ij} \geqslant p_j + K$ *otherwise.*

Proof. W.l.o.g. we assume $\mathcal{R} = [R]$. Let $\delta = \varepsilon/R$ and $N = \max\{K/\delta, 1\}$. We define the size and speed vectors of I′ as follows: For each job j we set $s'_r(j) = \delta N^{d_r(j)}$ for each $r \in [R]$, as well as $s'_{R+1}(j) = p_j$. Moreover, for each machine i we set $v'_r(j) = N^{-c_r(j)}$ for each $r \in [R]$, as well as $v'_{R+1}(j) = 1$. Then $p'_{ij} = \sum_{r\in[R+1]} s_r(j)v_r(i) = p_j + \sum_{r\in[R]} \delta N^{d_r(j)-c_r(j)}$ and therefore $p'_{ij} \geqslant p_j$ in any case. Furthermore, if $i \in \mathcal{M}(j)$, we have $d_r(j) \leqslant c_r(j)$ for each $r \in [R]$, and hence $p'_{ij} \leqslant p_j + R\delta = p_j + \varepsilon$. If, on the other hand, $i \notin \mathcal{M}(j)$, then there is an $r \in [R]$ such that $d_r(j) > c_r(j)$ yielding $p'_{ij} \geqslant p_j + \delta N \geqslant p_j + K$. □

The above lemma implies that from the perspective of approximation algorithms RAR(R) is essentially included in LRS(R + 1). We could use this lemma and Theorem 4.1 or Theorem 4.2 to show that there is no PTAS for LRS(3) unless P=NP. While this is already known [31], the resulting construction may be more accessible.

ESTABLISHED REDUCTIONS REVISITED. In the following, we first present the classical reduction by Lenstra et al. [108] showing 1.5-inapproximability for the restricted assignment problem. We show that the restricted assignment instances in this reduction can be modeled using 6 resources yielding the same hardness for RAR(6). Nearly the same argument was used by Bhaskara et al. [16] to show 1.5-inapproximability for LRS(7). Next, we take the same approach for the more recent reduction by Ebenlendr et al. [40] and show 1.5-inapproximability already for 4 resources.

In the 3-DM problem, the input consists of three disjoint sets A, B and C with $|A| = |B| = |C| = n \in \mathbb{N}$, as well as a set of triplets $E \subseteq \{\{a, b, c\} \mid a \in A, b \in B, c \in C\}$. The goal is to decide whether there is a subset $F \subseteq E$ that perfectly covers A, B and C, that is, for each $x \in A \cup B \cup C$ there is exactly one triplet $e \in F$ with $x \in e$. The set F is called a 3D-matching. We assume that the elements of A, B and C are indexed, that is, $A = \{a_1, a_2, \ldots, a_n\}$, $B = \{b_1, b_2, \ldots, b_n\}$ and $C = \{c_1, c_2, \ldots, c_n\}$. Furthermore, we assume that for each $x \in A \cup B \cup C$ there is at least one $e \in E$ with $x \in E$ (otherwise the problem is trivial). Via a reduction from 3-DM to the restricted assignment problem, Lenstra et al. [108] showed:

THEOREM 4.28 ([108]): There is no polynomial time approximation algorithm for restricted assignment with rate smaller than 1.5 unless P=NP.

Proof. Given an instance of 3-DM, we set $\mathcal{M} = E$ and $E(x) = \{e \in E \mid x \in e\}$ for each $x \in A \cup B \cup C$. For each $a \in A$, we introduce $|E(a)| - 1 \geqslant 0$ many dummy jobs with size 2 and eligible on machines $e \in E(a)$. Moreover, for each $x \in B \cup C$ we introduce an element job with size 1 eligible on machines $e \in E(x)$. Note that the overall size of the jobs is given by $2n + 2\sum_{a \in A}(|E(a)| - 1) = 2n + 2(|E| - n) = 2|\mathcal{M}|$.

If there is a schedule with makespan 2 for this instance, then each machine either processes two element or one dummy job. For each $x \in B \cup C$, we have $x \in e$ for the machine e processing the corresponding element job, and, furthermore, for each $a \in A$, there is exactly one machine e with $a \in e$ that does not process a dummy job (and therefore processes element jobs). Hence, we get a 3D-matching by selecting the machines that process element jobs.

If, on the other hand, there is a 3D-matching F for the 3-DM instance, then we can schedule the element job corresponding to $x \in B \cup C$ on the machine $e \in F$ with $x \in E$ and the dummy jobs corresponding to $a \in A$ on the $|E(a)| - 1$ machines $e \in E(a) \setminus F$. This yields a schedule with makespan 2. □

We reproduce the restrictions in the above reduction using six resources and get:

COROLLARY 4.29. *There is no polynomial time approximation algorithm for* RAR(6) *with rate smaller than* 1.5 *unless P=NP.*

Proof. We set $\mathcal{R} = \{(X, k) \mid X \in \{A, B, C\}, k \in [2]\}$. Let $e \in E$ and $X \in \{A, B, C\}$. We set the resource capacities $c_{(X,1)}(e) = i$ and $c_{(X,2)}(e) = (n + 1) - i$. Let x_j be the element with index j in X. We set the resource demand of a (element or dummy) job J corresponding to x_j as follows: $d_{(X,1)}(J) = j$, $d_{(X,2)}(J) = (n + 1) - j$, as well as $d_{(Y,k)}(J) = 0$ for each $Y \in \{A, B, C\} \setminus \{X\}$ and $k \in [2]$. It is easy to see that J can exclusively be scheduled on machines e with $x_j \in e$. □

In the classical 3-SAT problem, a conjunction of m clauses is given and each clause is a disjunction of at most three literals of variables $x_1, \ldots, x_n$. In the result due to Ebenlendr et al. [40], the modified 3-SAT problem, where each variable occurs exactly three and each literal at most two times in the formula, is reduced to the graph balancing problem, that is, restricted assignment with the additional property that each job is eligible on at most two machines. To show that the modified 3-SAT problem is NP-hard, we can use techniques already applied in Section 4.2: We may replace the d_j occurrences of variable x_j with new variables $z_{j1}, \ldots, z_{jd_j}$ and add new clauses $(z_{j1} \wedge \neg z_{j2})$, $\ldots (z_{jd_{j-1}} \wedge \neg z_{jd_j})$, $(z_{jd_j} \wedge \neg z_{j1})$.

THEOREM 4.30 ([40]): There is no polynomial time approximation algorithm with rate smaller than 1.5 for the graph balancing problem unless P=NP.

Proof. Given an instance of modified 3-SAT, we introduce clause machines v_i corresponding to the clauses C_i, and literal machines $u_{j,1}$ and $u_{j,0}$ corresponding to the literals x_j and $\neg x_j$. Furthermore, we introduce truth assignment jobs e_j for each variable x_j with size 2 and eligible on $u_{j,1}$ and $u_{j,0}$; and clause jobs $f_{i,j,\alpha}$ for each clause C_i and literal y_j occurring in C_i with $\alpha = 1$ if $y_j = x_j$ and $\alpha = 0$ if $y_j = \neg x_j$. The job $f_{i,j,\alpha}$ has size 1 and is eligible on v_i and $u_{j,\alpha}$. Lastly, we introduce a dummy job d_i for each clause C_i with less than three literals. Its size is 1 if C_i contains two literals, and 2 if C_i contains only one literal.

In a schedule with makespan 2, there is at least one clause job $f_{i,j,\alpha}$ for each v_i that is scheduled on $u_{j,\alpha}$ and not on v_i. Hence, the job e_j has to be scheduled on $u_{j,|\alpha-1|}$. Now, it is easy to see that there is a schedule with makespan 2, if and only if there is a satisfying assignment. The construction works as follows: Given a schedule with makespan 2, we set variable x_j to $\top$ if e_j is scheduled on $u_{j,0}$, and to $\bot$ otherwise. Moreover, given a satisfying truth assignment we assign the truth assignment jobs correspondingly, and the machines $u_{j,\alpha}$ that did not receive a truth assignment job receive all eligible clause jobs (at most two). □

We reproduce the restrictions in the above reduction using four resources and get:

COROLLARY 4.31. *There is no polynomial time approximation algorithm for* RAR(4) *with rate smaller than* 1.5 *unless P=NP.*

Proof. We set $\mathcal{R} = [4]$. The clause machine v_i has a resource capacity vector of $(2n+1, 2n+1, i, (m+1)-i)$, and the literal machine $u_{j,\alpha}$ has capacity vectors $(2j-\alpha, (2n+1)-(2j-\alpha), m+1, m+1)$. Furthermore, the truth assignment job e_j has a resource demand vector of $(2j-1, (2n+1)-2j, m+1, m+1)$; the clause job $f_{i,j,\alpha}$ has a demand vector of $(2j-\alpha, (2n+1)-(2j-\alpha), i, (m+1)-i)$; and the dummy job d_i has a demand vector of $(2n+1, 2n+1, i, (m+1)-i)$. It is easy to verify that the resulting sets of eligible machines are the same as described in Theorem 4.30. □

THREE RESOURCES. We present a reduction from 3-DM to RAR(3). The reduction is based on the classical result by Lenstra et al. [108] and very similar to a reduction by Bhaskara et al. [16] for LRS(4). However, there is a problem with the choice of processing times in the latter reduction, and the present result can be used to fix it. We discuss this problem at the end of this section.

Given an instance (A, B, C, E) of 3-DM, let $n = |A|$ and $E(x) = \{e \in E \mid x \in e\}$ for each $x \in A \cup B \cup C$. Furthermore, we set $\alpha_A = 12$, $\alpha_B = 13$, $\alpha_C = 22$, $\beta_A = 14$, $\beta_B = 15$ and $\beta_C = 18$. Let $\mathcal{R} = \{A, B, C\}$ and $\mathcal{M} = E$. For each machine e, we define the resource capacities as follows. Let $X \in \{A, B, C\}$ and $x_i \in X \cap e$ be the element of x with

index i. We set $c_X(e) = i$. Furthermore, for each element $x_i \in X$ with index i in $X \in \{A, B, C\}$, we introduce one element job with size α_X and $|E(x)| - 1$ dummy jobs with size β_X. The resource demand for each of these jobs is given by $d(i)$ with $d_X(i) = i$ and $d_Y(i) = 0$ for $Y \in \{A, B, C\} \setminus \{X\}$.

Claim 4.32. We have $\alpha_A + \alpha_B + \alpha_C = 47 = \beta_A + \beta_B + \beta_C$; any four numbers from $\Gamma = \{\alpha_A, \alpha_B, \alpha_C, \beta_A, \beta_B, \beta_C\} = \{12, 13, 22, 14, 15, 18\}$ sum up to a value bigger than 47; any selection of less than 3 numbers sums up to a value smaller than 47; and for any three numbers $\gamma_1, \gamma_2, \gamma_3 \in \Gamma$ with $\gamma_1 \leqslant \gamma_2 \leqslant \gamma_3$ and $\gamma_1 + \gamma_2 + \gamma_3 = 47$, we have either $(\gamma_1, \gamma_2, \gamma_3) = (\alpha_A, \alpha_B, \alpha_C)$ or $(\gamma_1, \gamma_2, \gamma_3) = (\beta_A, \beta_B, \beta_C)$.

Proof. The first three assertions are obvious, and the fourth holds due to a simple case analysis:

- If $\gamma_1 > 15$, we have $\gamma_1 \geqslant 18$, and hence $47 = \gamma_1 + \gamma_2 + \gamma_3 \geqslant 3 \cdot \gamma_1 = 54$: a contradiction.
- Note that $\gamma_3 \geqslant (\gamma_2 + \gamma_3)/2 = (47 - \gamma_1)/2$. Hence, $\gamma_1 \leqslant 15$ implies $\gamma_3 \geqslant 16$ and therefore $\gamma_3 \in \{18, 22\}$.
- If we have $\gamma_3 = 22 = \alpha_C$, then $\gamma_1 \leqslant (\gamma_1 + \gamma_2)/2 = (47 - \gamma_3)/2 = 12.5$. Hence, $\gamma_1 = 12 = \alpha_A$ and $\gamma_2 = 13 = \alpha_B$.
- If we have $\gamma_3 = 18 = \beta_C$, then $\gamma_2 \geqslant (\gamma_1 + \gamma_2)/2 = (47 - \gamma_3)/2 = 14.5$. Hence, $\gamma_2 \in \{15, 18\}$. If $\gamma_2 = 15 = \beta_B$, then $\gamma_1 = 14 = \beta_A$, and if $\gamma_2 = 18$, then $\gamma_1 = 11 \notin \Gamma$.

This concludes the proof of the claim. □

By brute force, it can be verified that 47 is the smallest value such that suitable numbers α_A, α_B, α_C, β_A, β_B and β_C exist and the above claim holds.

Claim 4.33. The summed up size of all the element and dummy jobs is $47|\mathcal{M}|$.

Proof. We have exactly n element jobs with size α_A, α_B and α_C, respectively, yielding an overall load of $47n$. The dummy jobs have an overall load of:

$$\begin{aligned}&\beta_A \sum_{a \in A} (|E(a)| - 1) + \beta_B \sum_{b \in B} (|E(b)| - 1) + \beta_C \sum_{c \in C} (|E(b)| - 1)\\ =&(\beta_A + \beta_B + \beta_C)(|E| - n) = 47(|\mathcal{M}| - n)\end{aligned}$$

In this equation, we used the simple fact that $\{E(x) \mid x \in X\}$ is a partition of E for each $X \in \{A, B, C\}$, and hence $|E| = \sum_{x \in X} |E(x)|$. □

These two claims imply:

Claim 4.34. In any schedule with makespan 47 for the constructed instance, each machine receives exactly three jobs with sizes $\gamma_1, \gamma_2, \gamma_3$ such that $(\gamma_1, \gamma_2, \gamma_3) = (\alpha_A, \alpha_B, \alpha_C)$ or $(\gamma_1, \gamma_2, \gamma_3) = (\beta_A, \beta_B, \beta_C)$.

Using these claims, we can show:

PROPOSITION 4.35. *There is a perfect matching for the given* 3-DM *instance, if and only if there is a schedule with makespan* 47 *for the constructed* RAR(3) *instance.*

Proof. Let F be a perfect matching for the 3-DM instance. For each $x \in A \cup B \cup C$ we assign the corresponding element job to the machine e with $x \in e$ and $e \in F$. Furthermore, the dummy jobs corresponding to $x \in X$ with $X \in \{A, B, C\}$, are distributed to the machines e with $x \in e$ and $e \notin F$ such that each machine receives exactly one job. Hence, each machine $e \in E$ receives exactly three eligible jobs either with sizes α_A, α_B and α_C (if $e \in F$) or β_A, β_B and β_C (otherwise).

Next, we assume that there is a schedule with makespan 47 for the scheduling instance. For each $X \in \{A, B, C\}$, there are exactly $|\mathcal{M}|$ many jobs with size α_X or β_X, and due to the above claims, we know that each machine receives exactly one of these jobs. For each $j \in [n]$, let $x_j \in X$ be the element with index j in $X \in \{A, B, C\}$. The machines $\bigcup_{j=i}^{n} E(x_j)$ are the only machines that may process jobs corresponding to $x_i, \dots, x_n$ for each $i \in [n]$ and we have exactly $\sum_{j=i}^{n} |E(x_j)|$ many such jobs. Hence, the machines from $E(x_i)$ receive exactly the jobs corresponding to x_i. Now, considering this and Claim 4.34, we get a perfect matching by selecting the machines that process three element jobs. □

TWO RESOURCES. We are able to refine the result for three resources to work for two resources as well by using another variant of 3-DM as the starting point of the reduction. The problem 3-DM* was introduced by Chen et al. [30] to get an improved lower bound for the approximation ratio of rank four unrelated scheduling. In this problem, a set of six disjoint sets $\mathcal{E} = \{A, A', B, B', C, C'\}$ is given. For each $X \in \mathcal{E}$, we have $|X| = 3n$ for some $n \in \mathbb{N}$ and the sets are indexed by $[3n]$, e.g., $A = \{a_1, a_2, \dots, a_{3n}\}$. Furthermore, there are two sets of triplets $E_1 \subseteq \left\{\{a_i, b_j, c_j\}, \{a'_i, b_j, c_j\} \,\middle|\, i \in [3n], j \in [3n]\right\}$ and $E_2 = \left\{\{a_i, b'_i, c'_i\}, \{a'_i, b'_i, c'_{\zeta(i)}\} \,\middle|\, i \in [3n]\right\}$ with $\zeta(3k+1) = 3k+2$, $\zeta(3k+2) = 3k+3$ and $\zeta(3k+3) = 3k+1$ for each $k \in \{0, \dots, n-1\}$. Note that the second set of triplets is already determined by the element sets in the input. Similar to the classical 3-DM problem, the goal is to decide whether there is a subset $F \subseteq E_1 \cup E_2$ that perfectly covers the element set, that is, for each $x \in \bigcup_{X \in \mathcal{E}} X$ there is exactly one triplet $e \in F$ with $x \in e$. Furthermore, we assume that for each $x \in \bigcup_{X \in \mathcal{E}} X$ there is at least one $e \in E$ with $x \in E$ (otherwise the problem is trivial).

Jobs	Resources	Machines	Resources
a_i	$(2i, 0)$	$\{a_i, b_j, c_j\}$	$(2i, 3n + j)$
a'_i	$(2i - 1, 0)$	$\{a'_i, b_j, c_j\}$	$(2i - 1, 3n + j)$
b_j	$(0, 3n + j)$	$\{a_i, b'_i, c'_i\}$	$(2i, i)$
c_j	$(0, 3n + j)$	$\{a'_i, b'_i, c'_{\zeta(i)}\}$	$(2i - 1, \zeta(i))$
b'_i	$(2i - 1, 0)$		
c'_i	$(0, i)$		

Table 4.8: The resource demands and capacities for the different job (types) and machines.

Let $\alpha_A = \alpha_{A'} = 12$, $\alpha_B = \alpha_{B'} = 13$, $\alpha_C = \alpha_{C'} = 22$, $\beta_A = \beta_{A'} = 14$, $\beta_B = \beta_{B'} = 15$ and $\beta_C = \beta_{C'} = 18$. We set $\mathcal{M} = E_1 \cup E_2$ and $\mathcal{R} = [2]$. The corresponding resource capacity vectors are presented in Table 4.8. Furthermore, for each element $x \in X$ in $X \in \mathcal{E}$, we introduce one element job with size α_X and $|E(x)| - 1$ dummy jobs with size β_X. The vector of resource demands for each such job is given in Table 4.8. Note that Claim 4.32, 4.33 and 4.34 hold for this reduction as well and with the same reasoning. Furthermore, a simple case analysis yields:

Claim 4.36. For each $x \in \bigcup_{X \in \mathcal{E}} X$, a (dummy or element) job corresponding to x is eligible on each machine e with $x \in e$.

Using these claims, we can conclude the proof of Theorem 4.2:

LEMMA 4.37. *There is a perfect matching for the given* 3-DM* *instance, if and only if there is a schedule with makespan* 47 *for the constructed* RAR(2) *instance.*

Proof. Let F be a perfect matching for the 3-DM* instance. For each $x \in \bigcup_{X \in \mathcal{E}} X$, we assign the corresponding element job to the machine e with $x \in e$ and $e \in F$. Furthermore, the dummy jobs corresponding to $x \in X$ with $X \in \mathcal{E}$, are distributed to the machines e with $x \in e$ and $e \notin F$ such that each such machine receives exactly one job. Hence, each machine $e \in E$ receives exactly three eligible jobs either with sizes α_A, α_B and α_C or β_A, β_B and β_C.

Next, we assume that there is a schedule with makespan 47 for the scheduling instance. There are exactly $|\mathcal{M}|$ many jobs with size $\alpha_A = \alpha_{A'}$ or $\beta_A = \beta_{A'}$ corresponding to elements of $A \cup A'$, and due to Claim 4.34 we know that each machine receives exactly one of these jobs. The machines corresponding to triplets from $E(a_{3n})$ are the only ones that can process the $|E(a_{3n})|$ jobs corresponding to a_{3n}, and hence each of these machines receives exactly one of these jobs. Now, the machines corresponding to triplets from $E(a'_{3n})$ are the only *remaining* ones that can process the $|E(a'_{3n})|$ jobs corresponding to a'_{3n}. Iterating this argument, we get that each machine e receives exactly one job corresponding to some $x \in A \cup A'$ with $x \in e$. Note that the

above argument was based on the first resource value. Considering the second resource value yields the same result for each $x \in C \cup C'$. For the elements $x \in B \cup B'$ both resource values have to be considered, namely the second for $b \in B$ and the first for $b' \in B'$, but the argument stays the same. Summing up, each machine $e = \{x, y, z\}$ receives exactly three jobs corresponding to x, y and z. Now, considering this and Claim 4.34, we get a perfect matching by selecting the triplets e that processes three element jobs. □

PROBLEM WITH KNOWN REDUCTION. We state the reduction by Bhaskara et al. [16] from 3-DM to LRS(4) and show that it is not sound. We remark that the construction can be repaired with not too much effort using processing times similar to the ones presented in the reduction for RAR(3) in Section 4.3.

Given an instance (A, B, C, E) of 3-DM and some $\varepsilon > 0$, let $n = |A|$, $N = n/\varepsilon$ and $E(x) = \{e \in E \mid x \in e\}$ for each $x \in A \cup B \cup C$. We identify the set of machines $\mathcal{M}$ with the set of triplets E, i.e., $\mathcal{M} = E$. The speed vector of $e = \{a_i, b_j, c_k\} \in \mathcal{M}$ is given by $(N^i, N^j, N^k, 1)$. Furthermore, for each $x \in A \cup B \cup C$, we introduce one element and $|E(x)| - 1$ dummy jobs. The size vectors of the jobs are presented in the following table:

	Element	Dummy
$a_i \in A$	$(\varepsilon N^{-i}, 0, 0, 1)$	$(\varepsilon N^{-i}, 0, 0, 0.8)$
$b_j \in B$	$(0, \varepsilon N^{-j}, 0, 1)$	$(0, \varepsilon N^{-j}, 0, 0.9)$
$c_k \in C$	$(0, 0, \varepsilon N^{-k}, 1)$	$(0, 0, \varepsilon N^{-k}, 1.3)$

In [16] the authors show that there is a schedule with makespan at most $3 + 3\varepsilon$ if there is a 3D-matching. Furthermore, two Lemmata (Lemma 2.1 and 2.2 in [16]) are used to show that the existence of a schedule with makespan at most $3.09 + 3\varepsilon$ implies the existence of a 3D-matching. We present a counter example. Let $n = 3$ and:

$$E = \{\{a_1, b_1, c_2\}, \{a_2, b_2, c_2\}, \{a_3, b_3, c_3\}, \{a_3, b_2, c_3\}, \{a_3, b_3, c_1\}\}$$

Hence, we have:

x	a_1	a_2	a_3	b_1	b_2	b_3	c_1	c_2	c_3
$\lvert E(x)\rvert$	1	1	3	1	2	2	1	2	2

In order to match a_1 and a_2, a 3D-matching would have to contain the first two triplets matching c_2 twice. Hence, there is no 3D-matching for this instance.

On the other hand, we define a schedule with makespan $3 + 3\varepsilon \leqslant 3.09 + 3\varepsilon$ in the following table.

Mach.	$\{a_1, b_1, c_2\}$	$\{a_2, b_2, c_2\}$	$\{a_3, b_3, c_3\}$	$\{a_3, b_2, c_3\}$	$\{a_3, b_3, c_1\}$
Jobs	a_1, a_2, b_1 (element)	a_3, b_2, c_2 (dummy)	a_3, b_3, c_3 (dummy)	a_3, b_2, c_3 (element)	b_3, c_1, c_2 (element)
Load	$3 + 2\varepsilon + \frac{\varepsilon}{N}$	$3 + 2\varepsilon + \frac{\varepsilon}{N}$	$3 + 3\varepsilon$	$3 + 3\varepsilon$	$3 + 2\varepsilon + \frac{\varepsilon}{N}$

4.4 OPEN PROBLEMS

We list some possible future research directions: From the perspective of complexity, tighter hardness results seem plausible. In particular, we have the same inapproximability results for RAR(2) and RAR(3) and it would be interesting to find a better result for RAR(3).

From the algorithmic perspective, it remains open whether any of the studied problems and RAI in particular admits an approximation algorithm with a rate smaller than 2. There have been some results [135, 143] for RAI using promising linear programming relaxations that may be useful in this context. Another possibility is the application of the local search techniques originally used by Svensson [137] for the restricted assignment problem. This approach recently yielded a breakthrough for the graph balancing problem [85].

Finally, while a PTAS for RAR(1) is known [122], it is unclear whether the problem admits a an EPTAS.

5

STRUCTURAL PARAMETER RESTRICTIONS

5.1 INTRODUCTION

Like in the last chapter, we study the relationship between the structure of restrictions and the existence of approximation schemes for the restricted assignment problem. To this end, we introduce a graph framework based on the restrictions and consider structural parameters of the resulting graphs in this context. This leads to PTAS results generalizing the hierarchical, tree-hierarchical and nested cases discussed in Section 4.1. We also use the framework to develop FPT results (see Section 2.1) which also work for the unrelated scheduling setting. These results can in turn be used to obtain FPTAS results.

Recall that in the unrelated scheduling problem we are given a set $\mathcal{J}$ of n jobs that has to be assigned to a set $\mathcal{M}$ of m machines via a schedule $\sigma : \mathcal{J} \to \mathcal{M}$. A job j has a processing time p_{ij} for every machine i and the goal is to minimize the makespan $C_{\max}(\sigma) = \max_i \sum_{j \in \sigma^{-1}(i)} p_{ij}$. On some machines a job might have a very high or even infinite processing time, hence it should never be processed on these machines. This amounts to assignment restrictions in which for every job j there is a subset $\mathcal{M}(j)$ of eligible machines on which it may be processed. In the restricted assignment problem, the machines are identical in the sense that each job j has the same processing time p_j on all its eligible machines, i.e., $p_{ij} \in \{p_j, \infty\}$.

GRAPH FRAMEWORK. We briefly describe the graph framework. In the *primal graph*, the vertices are the jobs and two vertices are connected by an edge if there is a machine on which both of the jobs can be processed. In the *dual graph*, on the other hand, the machines are vertices and two of them are adjacent if there is a job that can be processed by both machines. Lastly, we consider the *incidence graph*. This is a bipartite graph and both the jobs and machines are vertices. A job j is adjacent to a machine i if $i \in \mathcal{M}(j)$. In Figure 5.1, an example of each graph is given. These graphs have also been studied in the context of constraint satisfaction (see e.g. [139] or [129]) and we adopted the naming scheme.

We study so called width parameters for these graphs, namely, the *tree-* and *cliquewidth* which are probably the most famous width parameters. Concerning the cliquewidth, we will actually utilize the closely related *rankwidth*, but the results will translate directly back to the cliquewidth. Each of these parameters is associated with a graph

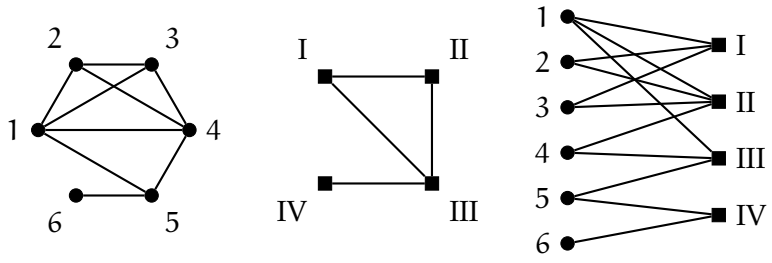


Figure 5.1: Primal, dual and incidence graph for an instance with 6 jobs and 4 machines.

decomposition which is used in the presented algorithms. For the definitions of these concepts, we refer to Section 5.2.

RESULTS AND ORGANIZATION. In the following, we denote the treewidth of the primal, dual and incidence graph with tw_p, tw_d and tw_i, respectively. Let $\mathcal{J}(i)$ be the set of jobs the machine i can process. In the context of parameterized algorithms, we show the following:

THEOREM 5.1: Unrelated scheduling is FPT for the parameter tw_p.

THEOREM 5.2: Unrelated Scheduling is FPT for the pair of parameters k_1, k_2 with $k_1 \in \{tw_d, tw_i\}$ and $k_2 \in \{OPT, \max_i |\mathcal{J}(i)|\}$.

Note that unrelated scheduling with constant k_2 remains NP-hard [40], and it is easy to see that k_1 is upper bounded by the number of machines m. Unrelated scheduling remains weakly NP-hard even if $m = 2$. The above results are achieved via dynamic programs that utilize tree decompositions of the respective graphs.

In the context of approximation we get:

THEOREM 5.3: There is an FPTAS for instances of unrelated scheduling with constant tw_d or tw_i.

This is achieved by combining the dynamic programs used for Theorem 5.2 with a suitable rounding. The result for the dual graph generalizes one by Lee, Leung and Pinedo [106]—who studied the case that the dual graph is a tree—and resolves cases that were marked as open in that paper. All results mentioned so far are discussed in Section 5.3.

In Section 5.4, we consider the clique- and rankwidth:

THEOREM 5.4: There is a PTAS for instances of the restricted assignment problem where the clique- or rankwidth of the incidence graph is bounded by a constant.

This results is achieved via dynamic programming utilizing a branch decomposition combined with a suitable rounding approach. It can be shown that instances of restricted assignment with hierarchical, tree-hierarchical or nested restrictions are special cases of the case when the incidence graph is a bi-cograph. Bicographs are known to have a cliquewidth of at most 4 (see [54]) and a suitable branch

decomposition can be found very easily. Therefore, we generalize and unify most known PTAS results for restricted assignment with structured job assignment restrictions.

Lastly, in Section 5.5, we discuss to what extent our results can be generalized to other objective functions. More precisely, we consider minimizing the ℓ_p-norm of machines loads, that is, $\|\lambda^\sigma\|_p = (\sum_{i\in\mathcal{M}}(\lambda_i^\sigma)^p)^{1/p}$ with $\lambda_i^\sigma = \sum_{j\in\sigma^{-1}(i)} p_{ij}$; and maximizing the minimum machine load $C_{\min}(\sigma) = \min_i \sum_{j\in\sigma^{-1}(i)} p_{ij}$. We argue that the results concerning the treewidth hold for these objective functions as well. Concerning the rank- and cliquewidth, we get a PTAS for the minimum machine load but only a QPTAS, that is, an approximation scheme with quasi-polynomial running time, for the ℓ_p-norm of machines loads. Quasi-polynomial running times are of the form $2^{\text{polylog}(|I|)}$.

FURTHER RELATED WORK. In the last chapter, namely in Section 4.1, we provided a detailed literature review that for the most part also applies to this setting. We highlight a few further results in the context of approximation and otherwise focus on the FPT perspective and other objective functions in the following.

We already mentioned that Lee, Leung and Pinedo [106] showed that there is an FPTAS for instances restricted assignment where the dual graph is a tree. Moreover, the special case of graph balancing where the graph is simple has been considered. For this problem, Asahiro et al. [10] presented, among other things, a pseudo-polynomial time algorithm for the case of graphs with bounded treewidth.

The first result for unrelated scheduling from the FPT perspective was given by Mnich and Wiese [116]. They showed that unrelated scheduling is FPT for the pair of parameters m and the number of distinct processing times. Knop and Koutecký [100] showed that the problem is also FPT for the parameter pair $\max p_{ij}$ and the number of machine types. Two machines have the same type, if each job has the same processing time on them. This was further generalized by Chen et al. [31] who showed that the problem is also FPT when considering $\max p_{ij}$ and the rank of the processing time matrix (p_{ij}). Furthermore, Szeider [140] showed that graph balancing on simple graphs with unary encoding of the processing times is not FPT for the parameter treewidth under usual complexity assumptions. For an overview of FPT results for scheduling, we refer to the survey by Mnich and van Bevern [115].

We briefly consider the other objective functions studied in this chapter. Concerning the maximization of the minimum machine load on unrelated machines, there is an FPTAS implied by a work by Woeginger [147] for the case that there is only a constant number of machines. In the general case, however, no approximation algorithm with a constant rate is known. Adapting the result by Lenstra et al.

[108], Bezáková and Dani [15] showed that there is no approximation ratio with a rate smaller than 2 (unless P=NP), and presented an algorithm that finds a solution with value at least $\text{OPT}(I) - \max p_{i,j}$ as well as a simple $(n - m + 1)$-approximation. The best known approximation ratio of $\mathcal{O}(n^{\varepsilon})$ is due to Bateni et al. [13] and Chakrabarty et al. [23] with a running time of $\mathcal{O}(n^{1/\varepsilon})$ for any $\varepsilon > 0$. Moreover, for maximization of the minimum machine load in the restricted assignment setting, Annamalai, Kalaitzis and Svensson [6] developed a 13-approximation. This was improved to a rate of $6 + \varepsilon$ by both Cheng and Mao [32] and Davies, Rothvoss and Zhang [38]. There are also PTAS results in this setting, namely, for the identical machine case [146] and the inclusion-free case [97].

Concerning the minimization of the ℓ_p-norm of machines loads, Azar et al. [12] showed for the restricted assignment case that there is no PTAS (for any $p > 1$) unless P=NP, and furthermore presented a 2-approximation that works for all norms simultaneously. Moreover, for unrelated scheduling, they presented an FPTAS for the case that the number of machines is constant. Lastly, there is a 2-approximation for the unrelated scheduling case due to Azar and Epstein [11].

5.2 PRELIMINARIES

In the following, I will always denote an instance of unrelated scheduling or restricted assignment and most of the time we will assume that it is feasible. We call an instance feasible if $\mathcal{M}(j) \neq \emptyset$ for every job $j \in \mathcal{J}$. A schedule is feasible if $\sigma(j) \in \mathcal{M}(j)$. For a subset $J \subseteq \mathcal{J}$ of jobs and a subset $M \subseteq \mathcal{M}$ of machines we denote the subinstance of I induced by J and M with $I[J, M]$. Furthermore, for a set S of schedules for I we define $\text{OPT}(S) = \min_{\sigma \in S} C_{\max}(\sigma)$ and $\text{OPT}(I) = \text{OPT}(S)$ if S is the set of all schedules for I. We will sometimes use $\text{OPT}(\emptyset) = \infty$. Note that there are no schedules for instances without machines. On the other hand, if I is an instance without jobs, we consider the empty function a feasible schedule (with makespan 0), and have therefore $\text{OPT}(I) = 0$ in that case.

SIMPLE DYNAMIC PROGRAMS. We sketch two basic dynamic programs for unrelated scheduling that will be needed as subprocedures in the following. The first one is based on iterating through the machines. Let $\text{OPT}(i, J) = \text{OPT}(I[\mathcal{J} \setminus J, [i]])$ for $J \subseteq \mathcal{J}$ and $i \in [m] := \{1, \dots, m\}$ assuming $\mathcal{M} = [m]$. Then it is easy to see that $\text{OPT}(i, J) = \min_{J \subseteq J' \subseteq \mathcal{J}} \max\{\text{OPT}(i-1, J'), \sum_{j \in J' \setminus J} p_{ij}\}$. Using this recurrence relation, a simple dynamic program can be formulated that computes the values $\text{OPT}(i, J)$. It holds that $\text{OPT}(I) = \text{OPT}(m, \emptyset)$, and, as usual for dynamic programs, an optimal schedule can be recovered via backtracking. The running time of such a program can be bounded by $2^{\mathcal{O}(n)} \times \mathcal{O}(m)$ yielding the following trivial result:

Remark 5.5. Unrelated scheduling is FPT for the parameter n.

The second dynamic program is based on iterating through the jobs. Let $\lambda \in \mathbb{Z}_{\geqslant 0}^{\mathcal{M}}$. We call λ a *load vector* and say that a schedule σ fulfills λ if $\lambda_i = \sum_{j \in \sigma^{-1}(i)} p_{ij}$. For $j \in [n]$, let $\Lambda(j)$ be the set of load vectors that are fulfilled by some schedule for the subinstance $I[[j], \mathcal{M}]$ assuming $\mathcal{J} = [n]$. Then $\Lambda(j)$ can also be defined recursively as the set of vectors λ with $\lambda_{i^*} = \lambda'_{i^*} + p_{i^*j}$ and $\lambda_i = \lambda'_i$ for $i \neq i^*$, where $i^* \in \mathcal{M}(j)$ and $\lambda' \in \Lambda(j-1)$. Using this, a simple dynamic program can be formulated that computes $\Lambda(j)$ for all $j \in [n]$. $\textsc{opt}(I)$ can be recovered from $\Lambda(n)$, and a corresponding schedule can be found via backtracking. Let there be a bound L for the number of distinct loads that can occur on each machine, i.e., $|\{\sum_{j \in \sigma^{-1}(i)} p_{ij} \mid \sigma \text{ schedule for } I\}| \leqslant L$ for each $i \in \mathcal{M}$. Then the running time can be bounded by $L^{\mathcal{O}(m)} \times \mathcal{O}(n)$ yielding:

Remark 5.6. Unrelated scheduling is FPT for the pair of parameters m and k with $k \in \{\textsc{opt}, \max_i |\mathcal{J}(i)|\}$.

For this, note that $2^{\max_i |\mathcal{J}(i)|}$ is a bound for the number of distinct loads that can occur on any machine. Furthermore, if a target objective value is given, we can easily modify the dynamic program to only consider loads that are upper bounded by this value.

This dynamic program can also be used to get a simple FPTAS for unrelated scheduling with a constant number of machines m. We briefly discuss such an FPTAS. Let B be an upper bound of $\textsc{opt}(I)$ with $B \leqslant 2\,\textsc{opt}$. Such a bound can be found with the 2-approximation by Lenstra et al. [108]. Moreover, let $\varepsilon > 0$ and $\delta = \varepsilon/2$. By rounding the processing time of every job up to the next integer multiple of $\delta B/n$, we get an instance I' whose optimum makespan is at most $\varepsilon\,\textsc{opt}(I)$ bigger than $\textsc{opt}(I)$. The dynamic program can easily be modified to only consider load vectors for I', where all loads are upper bounded by $(1+\delta/n)B$. Therefore, there can be at most $n/\delta + 2$ distinct load values for any machine and an optimal schedule for I' can be found in time $(n/\varepsilon)^{\mathcal{O}(m)}$. The schedule can trivially be transformed into a schedule for the original instance without an increase in the makespan.

TREE DECOMPOSTION AND TREEWIDTH. A *tree decomposition* of a graph G is a pair $(T, \{X_t \mid t \in V(T)\})$ where T is a tree, $X_t \subseteq V(G)$ for each $t \in V(t)$ is a set of vertices of G called a bag, and the following three conditions hold:

(T1) $\bigcup_{t \in V(T)} X_t = V(G)$

(T2) $\forall \{u, v\} \in E(G) \exists t \in V(T) : u, v \in X_t$

(T3) For every $u \in V(G)$ the set $T_u := \{t \in V(T) \mid u \in X_t\}$ induces a connected subtree of T.

The *width* of the decomposition is defined as $\max_{t \in V(T)}(|X_t| - 1)$, and the *treewidth* $\mathrm{tw}(G)$ of G is the minimum width of all tree decompositions of G. It is well known that forests are exactly the graphs

with treewidth one, and that the treewidth of G is at least as big as the biggest clique in G minus 1. More precisely, for each set of vertices $V' \subseteq V(G)$ inducing a clique in G, there is a node $t \in V(T)$ with $V' \subseteq X_t$ (see e.g. [19]). For a given graph and a value k it can be decided in FPT time (and linear in $|V(G)|$) whether the treewidth of G is at most k and in the affirmative case a corresponding tree decomposition can be computed [18]. However, deciding whether a graph has a treewidth of at most k, is NP-hard [7].

BRANCH DECOMPOSITION AND RANKWIDTH. It is easy to see that graphs with a small treewidth are sparse. Probably the most studied parameter for *dense* graphs is the cliquewidth $cw(G)$. In this work, however, we are going to utilize a closely related parameter called the rankwidth $rw(G)$. These two parameters are equivalent in the sense that $rw(G) \leqslant cw(G) \leqslant 2^{rw(G)+1} - 1$ [123]. Due to this, our result utilizing the rankwidth translate back to the cliquewidth. Furthermore, it is known that $cw(G) \leqslant 3 \times 2^{tw(G)-1}$ [34]. On the other hand, $tw(G)$ cannot be bounded by any function in $cw(G)$ or $rw(G)$ which can easily be seen by considering complete graphs.

A *cut* of G is a partition of $V(G)$ into two subsets. For $X, Y \subseteq V(G)$ let $A_G[X,Y] = (a_{xy})$ be the $|X| \times |Y|$ adjacency submatrix induced by X and Y, i.e., $a_{xy} = 1$ if $\{x,y\} \in E(G)$ and $a_{xy} = 0$ otherwise for $x \in X$ and $y \in Y$. The *cut rank* of (X,Y) is the rank of $A_G[X,Y]$ over the field with two elements $GF(2)$ and denoted by $\text{cutrk}_G(X,Y)$. A *branch decomposition* of $V(G)$ is a pair (T,η) where T is a tree with $|V(G)|$ leaves whose internal nodes all have degree 3 and η is a bijection from $V(G)$ to the leafs of T. For each $e = \{s,t\} \in E(T)$, there is an induced cut $\{X_s, X_t\}$ of G: For $x \in \{s,t\}$, the set X_x contains exactly the nodes $\eta^{-1}(\ell)$ where $\ell \in V(T)$ is a leaf that is in the same connected component of T as x if e is removed. Now, the *width* of e (with respect to cutrk_G) is $\text{cutrk}_G(X_s, X_t)$, and the *rankwidth* of the decomposition (T,η) is the maximum width over all edges of T. The *rankwidth* of G is the minimum rankwidth of all branch decompositions of G. It is well known that the cliquewidth of a complete graph is equal to 1, and this is also true for the rankwidth. For a given graph and fixed k there is an algorithm that finds a branch decomposition of width k in FPT time (cubic in $|V(G)|$), or reports correctly that none exists [64].

5.3 TREEWIDTH RESULTS

We start with some basic relationships between different restriction parameters for unrelated scheduling, especially the treewidths of the different graphs for a given instance. Similar relationships have been determined for the three graphs in the context of constraint satisfaction [129].

Remark 5.7. $tw_p \geqslant \max_i |\mathcal{J}(i)| - 1$ and $tw_d \geqslant \max_j |\mathcal{M}(j)| - 1$.

To see this, note that the sets $\mathcal{J}(i)$ and $\mathcal{M}(j)$ are cliques in the primal and dual graphs, respectively.

Remark 5.8. $tw_i \leqslant tw_p + 1$ and $tw_i \leqslant tw_d + 1$. On the other hand, $tw_p \leqslant (tw_i + 1)\max_i |\mathcal{J}(i)| - 1$ and $tw_d \leqslant (tw_i + 1)\max_j |\mathcal{M}(j)| - 1$.

These properties were pointed out by Kalaitis and Vardi [102] in a different context. Note that this remark together with Theorem 5.1 implies the results of Theorem 5.2 concerning the parameter $\max_i |\mathcal{J}(i)|$. Furthermore, in the case of machine scheduling with only 1 job and m machines or n jobs and only 1 machine, the primal graph has treewidth 0 or $n-1$ and the dual $m-1$ or 0, respectively, while the incidence graph in both cases has treewidth 1.

DYNAMIC PROGRAMS. We show how a given tree decomposition $(T, \{X_t \mid t \in V(T)\})$ of width k for any one of the three graphs can be used to design a dynamic program for the corresponding instance I of unrelated scheduling. Selecting a node as the root of the decomposition, the dynamic program works in a bottom-up manner from the leaves to the root. We assume that the decomposition has the following simple form: For each leaf node $t \in V(T)$ the bag X_t is empty and we fix one of these nodes as the root a of T. Furthermore, each internal node t has exactly two children $\ell(t)$ and $r(t)$ (left and right), and each node $t \neq a$ has one parent $p(t)$. We denote the descendants of t with $\mathrm{desc}(t)$. A decomposition of this form can be generated from any other one without increasing the width and growing only linearly in size through the introduction of dummy nodes. The bag of a dummy node is either empty or identical to the one of its parent. In the literature, the so called *nice* tree decomposition due to Kloks [99] is often used which has an even simpler structure and could be utilized in the following as well.

For each of the graphs and each node $t \in V(T)$, we define sets $\check{J}_t \subseteq \mathcal{J}$ and $\check{M}_t \subseteq \mathcal{M}$ of *inactive* jobs and machines along with sets J_t and M_t of *active* jobs and machines. The active jobs and machines in each case are defined based on the respective bag X_t, and the inactive ones have the property that they were active for a descendant $t \in \mathrm{desc}(t)$ of t but are not at t. In addition, there are *nearly inactive* jobs $\tilde{J}_t$ and machines $\tilde{M}_t$, which are the jobs and machines that are deactivated when going from t to its parent $p(t)$ (for the root a we set $\tilde{J}_t = \tilde{M}_t = \emptyset$). The sets are defined so that certain conditions hold. The first two are that the (nearly) inactive jobs may only be processed on active or inactive machines, and the (nearly) inactive machines can only process active or inactive jobs:

$$\mathcal{M}(\check{J}_t \cup \tilde{J}_t) \subseteq M_t \cup \check{M}_t \tag{5.1}$$

$$\mathcal{J}(\check{M}_t \cup \tilde{M}_t) \subseteq J_t \cup \check{J}_t \tag{5.2}$$

Above, we use the notation $\mathcal{M}(J^*) = \bigcup_{j \in J^*} \mathcal{M}(j)$ for any $J^* \subseteq \mathcal{J}$ and $\mathcal{J}(M^*) = \bigcup_{i \in M^*} \mathcal{J}(i)$ for any $M^* \subseteq \mathcal{M}$. Furthermore, the (nearly)

inactive jobs and machines of the children of an internal node t form a disjoint union of the inactive jobs and machines of t, respectively:

$$\check{J}_t = \check{J}_{\ell(t)} \dot\cup \tilde{J}_{\ell(t)} \dot\cup \check{J}_{r(t)} \dot\cup \tilde{J}_{r(t)} \tag{5.3}$$

$$\check{M}_t = \check{M}_{\ell(t)} \dot\cup \tilde{M}_{\ell(t)} \dot\cup \check{M}_{r(t)} \dot\cup \tilde{M}_{r(t)} \tag{5.4}$$

For any two sets A and B, we emphasize by writing $A \dot\cup B$ that the union $A \cup B$ is disjoint, i.e., $A \cap B = \emptyset$. Now, at each node of the decomposition the basic idea is to perform three steps:

1. Combine the information from the children (for internal nodes).
2. Consider the nearly inactive jobs and machines:
   - *Primal and incidence graph:* Try all possible ways of scheduling active jobs on nearly inactive machines.
   - *Dual and incidence graph:* Try all possible ways of scheduling nearly inactive jobs on active machines.
3. Combine the information from the last two steps.

For the second step, the dynamic programs described in Section 5.2 are used as subprocedures. We now consider each of the three graphs.

THE PRIMAL GRAPH. In the primal graph, all the vertices are jobs, and we define the active jobs of a tree node t to be exactly the jobs that are included in the respective bag, i.e., $J_t = X_t$. The inactive jobs are those that are not included in X_t but are in a bag of some descendant of t, and the nearly inactive ones are those that are active at t but inactive at $p(t)$, i.e., $\check{J}_t = \{j \in \mathcal{J} \mid j \notin X_t \wedge \exists t' \in \operatorname{desc}(t) : j \in X_{t'}\}$ and $\tilde{J}_t = J_t \setminus J_{p(t)}$. Moreover, the inactive machines are the ones on which some inactive job may be processed, and the (nearly in-)active machines are those that can process (nearly in-)active jobs and are not inactive, i.e., $\check{M}_t = \mathcal{M}(\check{J}_t)$, $M_t = \mathcal{M}(J_t) \setminus \check{M}_t$ and $\tilde{M}_t = \mathcal{M}(\tilde{J}_t) \setminus \check{M}_t$. For these definitions we get:

LEMMA 5.9. *The conditions (5.1)-(5.4) hold, as well as:*

$$\mathcal{J}(\tilde{M}_t) \subseteq J_t \tag{5.5}$$

$$\mathcal{M}(\check{J}_t \cup \tilde{J}_t) = \tilde{M}_t \cup \check{M}_t \tag{5.6}$$

Proof. (5.1) and (5.6):

$$\mathcal{M}(\check{J}_t \cup \tilde{J}_t) = \mathcal{M}(\check{J}_t) \cup \mathcal{M}(\tilde{J}_t) = \check{M}_t \cup (\mathcal{M}(\tilde{J}_t) \setminus \check{M}_t) = \check{M}_t \cup \tilde{M}_t$$

This yields (5.6) and (5.6) implies (5.1).

(5.2) and (5.5): Let $i \in \check{M}_t \cup \tilde{M}_t$ and $j \in \mathcal{J}(i)$. We first consider the case that $i \in \check{M}_t$. Then there is a job $j' \in \check{J}_t$ with $i \in \mathcal{M}(j')$. If $j = j'$, we have $j \in \check{J}_t$ and otherwise $\{j, j'\} \in E(G)$. Because of (T2), there is a node $t' \in V(T)$ with $j, j' \in J_{t'}$. Since $j' \in \check{J}_t$, we have $j' \notin J_t$. This together

with (T3) gives $t' \in \mathrm{desc}(t)$. Now, $j \notin J_t$ implies $j \in \check{J}_t$. Therefore, we have $j \in J_t \cup \check{J}_t$. Next, we consider the case that $i \in \tilde{M}_t$. In this case, there is a job $j' \in \tilde{J}_t$ with $i \in \mathcal{M}(j')$, and, for each job $j'' \in \check{J}_t$, we have $i \notin \mathcal{M}(j'')$. If $j = j'$, we have $j \in J_t$ and otherwise $\{j, j'\} \in E(G)$. Because of (T2), there is again a node $t' \in V(T)$ with $j, j' \in J_{t'}$. Since $j, j' \notin \check{J}_t$, $j' \notin J_{p(t)}$ and $j' \in J_t$ we get $j \in J_t$ using (T3). This also implies (5.5).

(5.3): All but $(\check{J}_{\ell(t)} \cup \tilde{J}_{\ell(t)}) \cap (\check{J}_{r(t)} \cup \tilde{J}_{r(t)}) = \emptyset$ follows directly from the definitions. Assuming there is a job $j \in (\check{J}_{\ell(t)} \cup \tilde{J}_{\ell(t)}) \cap (\check{J}_{r(t)} \cup \tilde{J}_{r(t)})$, we get $j \in J_t$ because of (T3), yielding a contradiction.

(5.4): Because of (5.3) and the definitions, we get $\check{M}_t = \check{M}_{\ell(t)} \cup \tilde{M}_{\ell(t)} \cup \check{M}_{r(t)} \cup \tilde{M}_{r(t)}$, and $\check{M}_{s(t)} \cap \tilde{M}_{s(t)}$ for $s \in \{\ell, r\}$ is clear from the definitions. Therefore, it remains to show $(\check{M}_{\ell(t)} \cup \tilde{M}_{\ell(t)}) \cap (\check{M}_{r(t)} \cup \tilde{M}_{r(t)}) = \emptyset$. We assume that there is a machine i in this cut. Then there are jobs $j_s \in \check{J}_{s(t)} \cup \tilde{J}_{s(t)}$ for $s \in \{\ell, r\}$ with $i \in \mathcal{M}(j_s)$. We have $\{j_\ell, j_r\} \in E$, and, because of (T2), there is a node t' with $j_\ell, j_r \in J_{t'}$. Because of $j_\ell, j_r \notin J_t$ and (T3), we have a contradiction. □

For $J \subseteq \mathcal{J}$ and $M \subseteq \mathcal{M}$, let $\Gamma(J, M) = \{J' \subseteq J \mid \forall j \in J' : \mathcal{M}(j) \cap M \neq \emptyset\}$. Let $t \in V(T)$, $J \in \Gamma(J_t, \check{M}_t)$, and $J' \in \Gamma(J_t \setminus \tilde{J}_t, \check{M}_t \cup \tilde{M}_t)$. We set $S(t, J)$ and $\tilde{S}(t, J')$ to be the sets of feasible schedules for the instances $I[\check{J}_t \cup J, \check{M}_t]$ and $I[\check{J}_t \cup \tilde{J}_t \cup J', \check{M}_t \cup \tilde{M}_t]$, respectively. We will consider $\mathrm{OPT}(S(t, J))$ and $\mathrm{OPT}(\tilde{S}(t, J'))$.

First note that $\mathrm{OPT}(I) = \mathrm{OPT}(S(a, \emptyset))$ (remember that a is the root of T). Moreover, for a leaf node t, there are neither jobs nor machines and $\mathrm{OPT}(S(t, \emptyset)) = \mathrm{OPT}(\tilde{S}(t, \emptyset)) = \mathrm{OPT}(\{\emptyset\}) = 0$ holds. Hence, let t be a non-leaf node. We first consider how $\mathrm{OPT}(S(t, J))$ can be computed from the children of t (Step 1). Due to (T3), the jobs from J are already active on at least one of the direct descendants of t. Because of this and (5.4), J may be split in two parts $J_\ell \dot\cup J_r = J$ where $J_s \in \Gamma(J_{s(t)} \setminus \tilde{J}_{s(t)}, \check{M}_{s(t)} \cup \tilde{M}_{s(t)})$ for $s \in \{\ell, r\}$. Let $\Phi(J)$ be the set of such pairs (J_ℓ, J_r).

LEMMA 5.10. *We have:*

$$\mathrm{OPT}(S(t, J)) = \min_{(J_\ell, J_r) \in \Phi(J)} \max_{s \in \{\ell, r\}} \mathrm{OPT}(\tilde{S}(s(t), J_s))$$

Proof. Let $\sigma^* \in S(t, J)$ be optimal. Since $J \subseteq J_t$, we have $J \cap \tilde{J}_{s(t)} = \emptyset$ for $s \in \{\ell, r\}$. Let $J^*_s = \sigma^{*-1}(\check{M}_{s(t)} \cup \tilde{M}_{s(t)}) \cap J$. Because of (5.4), we have $J = J^*_\ell \dot\cup J^*_r$ and $J^*_s \in \Gamma(J_{s(t)} \setminus \tilde{J}_{s(t)}, \check{M}_{s(t)} \cup \tilde{M}_{s(t)})$ obviously holds. Let $\sigma^*_s = \sigma^*|_{J_s \cup \check{J}_{s(t)} \cup \tilde{J}_{s(t)}}$. Because of (5.6), we have $\sigma^*_s \in \tilde{S}(s(t), J^*_s)$ and (5.3) implies $\sigma^* = \sigma^*_\ell \cup \sigma^*_r$ (here we consider functions as sets of pairs). This yields:

$$\begin{aligned} \mathrm{OPT}(S(t, J)) &= C_{\max}(\sigma^*) \\ &= \max_{s \in \{\ell, r\}} C_{\max}(\sigma^*_s) \\ &\geqslant \max_{s \in \{\ell, r\}} \mathrm{OPT}(\tilde{S}(s(t), J^*_s)) \end{aligned}$$

$$\geqslant \min_{(J_\ell, J_r) \in \Phi(J)} \max_{s \in \{\ell, r\}} \textsc{opt}(\tilde{S}(s(t), J_s))$$

Now, let $(J_\ell, J_r) \in \Phi(J)$ be minimizing the right-most term in the above expression and $\sigma_s \in \tilde{S}(s(t), J_s)$ be optimal. Then (5.3) and (5.4) imply that $\sigma := \sigma_\ell \cup \sigma_r$ is in $S(t, J)$. Therefore, we have $C_{\max}(\sigma) \geqslant C_{\max}(\sigma^*)$. By definition we also have $C_{\max}(\sigma) = \min_{(J_\ell, J_r)} \max_s \textsc{opt}(\tilde{S}(s(t), J_s))$ and the claim follows. □

Consider the computation of $\textsc{opt}(\tilde{S}(t, J'))$, that is, step 3. We may split J' and $\tilde{J}_t$ into a set going to the nearly inactive and a set going to the inactive machines. We set $\Psi(J')$ to be the set of pairs (A, X) with $J' \cup \tilde{J}_t = A \dot\cup X$, $A \in \Gamma(\tilde{J}_t \cup J', \tilde{M}_t)$ and $X \in \Gamma(\tilde{J}_t \cup J', \check{M}_t)$.

LEMMA 5.11. *We have:*

$$\textsc{opt}(\tilde{S}(t, J')) = \min_{(A,X) \in \Psi(J')} \max\{\textsc{opt}(S(t, X)), \textsc{opt}(I[A, \tilde{M}_t])\}$$

Proof. Let $\sigma^* \in \tilde{S}(t, J')$ be an optimal schedule. Because of (5.5), we have $\sigma^{*-1}(\tilde{M}_t) \subseteq J' \cup \tilde{J}_t$. We set $A^* = \sigma^{*-1}(\tilde{M}_t)$ and $X^* = (J' \cup \tilde{J}_t) \setminus A$. Then $(A^*, X^*) \in \Psi(J')$. Let $\check\sigma^* = \sigma^*|_{\check{J}_t \cup X^*}$ and $\tilde\sigma^* = \sigma^*|_A$. Then $\check\sigma^* \in S(t, X^*)$ and $\tilde\sigma^*$ is a feasible schedule for $I[A^*, \tilde{M}_t]$. Because of (5.3) and (5.4), we have $\sigma = \check\sigma^* \dot\cup \tilde\sigma^*$ and:

$$\begin{aligned}
\textsc{opt}(\tilde{S}(t, J')) &= C_{\max}(\sigma^*) \\
&= \max\{C_{\max}(\check\sigma^*), C_{\max}(\tilde\sigma^*)\} \\
&\geqslant \max\{\textsc{opt}(S(t, X^*)), \textsc{opt}(I[A^*, \tilde{M}_t])\} \\
&\geqslant \min_{(A,X) \in \Psi(J')} \max\{\textsc{opt}(S(t, X)), \textsc{opt}(I[A, \tilde{M}_t])\}
\end{aligned}$$

Now, let $(A, X) \in \Psi(J')$ be minimizing the right-most term in the above expression, $\check\sigma \in S(t, X)$ be optimal, and $\tilde\sigma$ be an optimal schedule for $I[A, \tilde{M}_t]$. Then (5.3) and (5.4) yield $\sigma := \check\sigma \dot\cup \tilde\sigma \in \tilde{S}(t, J')$, and therefore $C_{\max}(\sigma^*) \leqslant C_{\max}(\sigma)$. By definition $C_{\max}(\sigma)$ also equals $\min_{(A,X)} \max\{\textsc{opt}(S(t, X)), \textsc{opt}(I[A, \tilde{M}_t])\}$ and the claim follows. □

Determining the values $\textsc{opt}(I[A, \tilde{M}_t])$ corresponds to Step 2. Note that these values can be computed using the first dynamic program from Section 5.2 in time $2^{\mathcal{O}(k)} \times \mathcal{O}(m)$.

THE DUAL GRAPH. For the dual graph, the (in-)active jobs and machines are defined analogously: The active machines for a tree node t are the ones in the respective bag, the inactive machines are those that were active for some descendant but are not active for t, and the nearly inactive machines are those that are active at t but inactive at its parent, i.e., $M_t = X_t$, $\check{M}_t = \{i \in \mathcal{M} \mid i \notin M_t \wedge \exists t' \in \text{desc}(t) : i \in X_{t'}\}$ and $\tilde{M}_t = M_t \setminus \check{M}_{p(t)}$. Furthermore, the inactive jobs are those that may be processed on some inactive machine and the (nearly in-)active ones are those that can be processed on some (nearly in-)active machine and are not inactive, i.e., $\check{J}_t = \mathcal{J}(\check{M}_t)$, $J_t = \mathcal{J}(M_t) \setminus \check{J}_t$ and $\tilde{J}_t = \mathcal{J}(\tilde{M}_t) \setminus \check{J}_t$.

With these definitions, we get the following lemma which can be proved analogously to Lemma 5.9:

LEMMA 5.12. *The conditions (5.1)-(5.4) hold, as well as:*

$$\mathcal{M}(\tilde{J}_t) \subseteq M_t \tag{5.7}$$

$$\mathcal{J}(\check{M}_t \cup \tilde{M}_t) = \tilde{J}_t \cup \check{J}_t \tag{5.8}$$

□

We need some extra notation. Like we did in Section 5.2, we consider load vectors $\lambda \in \mathbb{Z}_{\geqslant 0}^{M}$ where $M \subseteq \mathcal{M}$ is a set of machines. We say that a schedule σ fulfills λ if $\lambda_i = \sum_{j \in \sigma^{-1}(i)} p_{ij}$ for each $i \in M$. For any set S of schedules for I, we denote the set of load vectors for M that are fulfilled by at least one schedule from S with $\Lambda(S, M)$. Furthermore, we denote the set of all schedules for I with $S(I)$, and, for a subset of jobs $J \subseteq \mathcal{J}$, we write $\Lambda(J, M)$ as a shortcut for $\Lambda(S(I[J, M]), M)$. Let $t \in V(T)$. We set $S(t) = S(I[\check{J}_t, \check{M}_t \cup M_t])$ and $\tilde{S}(t) = S(I[\check{J}_t \cup \tilde{J}_t, \check{M}_t \cup M_t])$. Moreover, for $\lambda \in \Lambda(S(t), M_t)$ and $\lambda' \in \Lambda(\tilde{S}(t), M_t)$, we set $S(t, \lambda) \subseteq S(t)$ and $\tilde{S}(t, \lambda') \subseteq \tilde{S}(t)$ to be those schedules that fulfill λ and λ', respectively. We now consider $\textsc{opt}(S(t, \lambda))$ and $\textsc{opt}(\tilde{S}(t, \lambda'))$.

First note $\textsc{opt}(I) = \textsc{opt}(S(a, \emptyset))$. Moreover, for a leaf node t, we have neither jobs nor machines and $\Lambda(S(t), M_t) = \Lambda(\tilde{S}(t), M_t) = \{\emptyset\}$. Therefore, $\textsc{opt}(S(t, \emptyset)) = \textsc{opt}(\tilde{S}(t, \emptyset)) = \textsc{opt}(\{\emptyset\}) = 0$. Hence, let t be a non-leaf node. Again, we first consider how $\textsc{opt}(S(t, \lambda))$ can be computed from the children of t. Because of (5.3), λ may be split into a left and a right part. For two machine sets M, M', let $\tau_{M,M'} : \mathbb{Z}_{\geqslant 0}^{M} \to \mathbb{Z}_{\geqslant 0}^{M'}$ be a transformation function for load vectors where the entry of $\tau_{M,M'}(\lambda)$ indexed by i equals λ_i for $i \in M \cap M'$ and 0 otherwise. We set $\Xi(\lambda)$ to be the set of pairs $(\lambda_\ell, \lambda_r)$ with $\lambda = \tau_{M_{\ell(t)}, M_t}(\lambda_\ell) + \tau_{M_{r(t)}, M_t}(\lambda_r)$, and $\lambda_s \in \Lambda(\tilde{S}(s(t)), M_{s(t)})$ for $s \in \{\ell, r\}$.

LEMMA 5.13. *We have:*

$$\textsc{opt}(S(t, \lambda)) = \min_{(\lambda_\ell, \lambda_r) \in \Xi(\lambda)} \max_{s \in \{\ell, r\}} \textsc{opt}(\tilde{S}(s(t), \lambda_s))$$

Proof. Let $\sigma^* \in S(t, \lambda)$ be optimal. Because of (5.1), we have $\sigma^*(\check{J}_{s(t)} \cup \tilde{J}_{s(t)}) \subseteq M_{s(t)} \cup \check{M}_{s(t)}$ for $s \in \{\ell, r\}$. Let $\sigma_s^* = \sigma^*|_{\check{J}_{s(t)} \cup \check{J}_{s(t)}}$ and λ_s^* the load vector that σ_s^* fulfills on $M_{s(t)}$. Then we have $\sigma_s^* \in \tilde{S}(s(t), \lambda_s^*)$ and $(\lambda_\ell^*, \lambda_r^*) \in \Xi(\lambda)$. Because of (5.3) and (5.4), we have $\sigma^* = \sigma_\ell^* \dot{\cup} \sigma_r^*$. Moreover, we have:

$$\begin{aligned}
\textsc{opt}(S(t, \lambda)) &= C_{\max}(\sigma^*) \\
&= \max_{s \in \{\ell, r\}} C_{\max}(\sigma_s^*) \\
&\geqslant \max_{s \in \{\ell, r\}} \textsc{opt}(\tilde{S}(s(t), \lambda_s^*)) \\
&\geqslant \min_{(\lambda_\ell, \lambda_r) \in \Xi(\lambda)} \max_{s \in \{\ell, r\}} \textsc{opt}(\tilde{S}(s(t), \lambda_s))
\end{aligned}$$

Now, let $(\lambda_\ell, \lambda_r) \in \Xi(\lambda)$ be minimizing the right-most term in the above expression and $\sigma_s \in \tilde{S}(s(t), \lambda_s)$ be optimal. Then $\sigma := \sigma_\ell \cup \sigma_r$ is in $S(t, \lambda)$ and $C_{\max}(\sigma) = \min_{(\lambda_\ell,\lambda_r)} \max_s \textsc{opt}(\tilde{S}(s(t), \lambda_s))$. Since furthermore $C_{\max}(\sigma) \geqslant C_{\max}(\sigma^*)$, the claim follows. □

Note that determining $\textsc{opt}(S(t, \lambda))$ corresponds to Step 1. Next, we consider $\textsc{opt}(\tilde{S}(t, \lambda'))$ (see Step 3). We may split λ' into the load due to inactive and that due to nearly inactive jobs. Note that the nearly inactive jobs can only be processed by active machines (5.7). We set $\Upsilon(\lambda')$ to be the set of pairs (α, ξ) with $\lambda' = \alpha + \xi$, $\alpha \in \Lambda(\tilde{J}_t, M_t)$ and $\xi \in \Lambda(S(t), M_t)$.

LEMMA 5.14. *We have:*

$$\textsc{opt}(\tilde{S}(t, \lambda')) = \min_{(\alpha,\xi)\in\Upsilon(\lambda')} \max\big(\{\textsc{opt}(S(t, \xi))\} \cup \{\lambda'_i \mid i \in M_t\}\big)$$

Proof. Let $\sigma^* \in \tilde{S}(t, \lambda')$ be optimal. Then (5.7) implies $\sigma^*(\tilde{J}_t) \subseteq M_t$. We set $\tilde{\sigma}^* = \sigma^*|_{\tilde{J}_t}$ and $\check{\sigma}^* = \sigma^*|_{\check{J}_t}$. Furthermore, let α^* be the load vector fulfilled by $\tilde{\sigma}^*$ and ξ^* the one fulfilled by $\check{\sigma}^*$ on M_t. Then $\tilde{\sigma}^*$ is a feasible schedule for $I[\tilde{J}_t, M_t]$ fulfilling α^*, $\check{\sigma}^* \in S(t, \xi^*)$ and $(\alpha^*, \xi^*) \in \Upsilon(\lambda')$. Furthermore, (5.3) yields $\sigma^* = \tilde{\sigma}^* \dot{\cup} \check{\sigma}^*$. We get:

$$\begin{aligned}
\textsc{opt}(\tilde{S}(t, \lambda')) &= C_{\max}(\sigma^*) \\
&= \max\left(\{C_{\max}(\check{\sigma}^*)\} \cup \{\lambda'_i \mid i \in M_t\}\right) \\
&\geqslant \max\left(\{\textsc{opt}(S(t, \xi^*))\} \cup \{\lambda'_i \mid i \in M_t\}\right) \\
&\geqslant \min_{(\alpha,\xi)\in\Upsilon(\lambda')} \max\left(\{\textsc{opt}(S(t, \xi))\} \cup \{\lambda'_i \mid i \in M_t\}\right)
\end{aligned}$$

Now, let $(\alpha, \xi) \in \Upsilon(\lambda')$ be minimizing the right-most term in the above expression, $\check{\sigma} \in S(t, \xi)$ be optimal, and $\tilde{\sigma}$ be a feasible schedule for $I[\tilde{J}_t, M_t]$ fulfilling α. Then $\sigma := \check{\sigma} \cup \tilde{\sigma} \in \tilde{S}(t, \lambda')$ and therefore $C_{\max}(\sigma) \geqslant C_{\max}(\sigma^*)$. Since we additionally have

$$C_{\max}(\sigma) = \min_{(\alpha,\xi)} \max\left(\{\textsc{opt}(S(t, \xi))\} \cup \{\lambda'_i \mid i \in M_t\}\right),$$

the claim follows. □

The set $\Lambda(\tilde{J}_t, M_t)$ can be computed using the second dynamic program described in Section 5.2 in time $L^{\mathcal{O}(k)} \times \mathcal{O}(n)$ if L is again a bound on the number of distinct loads that can occur on each machine. This corresponds to Step 2.

THE INCIDENCE GRAPH. For the incidence graph, we combine the ideas that we used for the two other graphs. The situation is slightly more complicated because we have to handle the jobs and machines simultaneously. All the job sets are defined like in the primal, and all the machine sets like in the dual graph case, i.e., $J_t = X_t \cap \mathcal{J}$, $M_t = X_t \cap \mathcal{M}$, $\check{J}_t = \{j \in \mathcal{J} \mid j \notin X_t \wedge \exists t' \in \text{desc}(t) : j \in X_{t'}\}$, $\check{M}_t = \{i \in \mathcal{M} \mid i \notin M_t \wedge \exists t' \in \text{desc}(t) : i \in X_{t'}\}$, $\tilde{J}_t = J_t \setminus J_{p(t)}$, and $\tilde{M}_t = M_t \setminus \check{M}_{p(t)}$.

With these definitions the conditions (5.1)-(5.4) follow almost directly from the definitions together with (T2) and (T3). The proofs for the recurrence relations in this paragraph have the same structure as the proofs for the other recurrence relations and no new ideas are needed. Therefore, they are omitted.

Let $t \in V(t)$, $J \in \Gamma(J_t, \check{M}_t)$ and $J' \in \Gamma(J_t \setminus \tilde{J}_t, \check{M}_t \cup \tilde{M}_t)$. We set $S(t, J)$ to be the set of feasible schedules σ for $I[\check{J}_t \cup J, \check{M}_t \cup M_t]$ that schedule the jobs from J on inactive machines, i.e., $\sigma(j) \in \check{M}_t$ for each $j \in J$. Moreover, $\tilde{S}(t, J')$ is the set of schedules for $I[\check{J}_t \cup \tilde{J}_t \cup J', \check{M}_t \cup M_t]$ that schedule the jobs from J' on (nearly) inactive machines $\tilde{M}_t \cup \check{M}_t$. The sets of schedules that in addition fulfill a load vector $\lambda \in \Lambda(S(t, J), M_t)$ or $\lambda' \in \Lambda(\tilde{S}(t, J')$ are denoted by $S(t, J, \lambda)$ and $\tilde{S}(t, J', \lambda')$. We consider $\text{OPT}(S(t, J, \lambda))$ and $\text{OPT}(\tilde{S}(t, J', \lambda'))$.

First note $\text{OPT}(I) = \text{OPT}(S(a, \emptyset, \emptyset))$. For a leaf note t there are neither jobs nor machines and therefore $\text{OPT}(S(t, \emptyset, \emptyset)) = \text{OPT}(\emptyset)) = 0$. Hence, let t be a non-leaf node. Like before, we first consider $\text{OPT}(S(t, J, \lambda))$. Both J and λ may be split into a left and a right part and we set $\Phi(J)$ like before. Moreover, for $(J_\ell, J_r) \in \Phi(J)$ we define $\Xi(\lambda, (J_\ell, J_r))$ to be the set of pairs $(\lambda_\ell, \lambda_r)$ with $\lambda_s \in \Lambda(\tilde{S}(s(t), J_s), M_{s(t)})$ for $s \in \{\ell, r\}$.

LEMMA 5.15. *We have:*

$$\text{OPT}(S(t, J, \lambda)) = \min_{(J_\ell, J_r), (\lambda_\ell, \lambda_r)} \max_{s \in \{\ell, r\}} \text{OPT}\left(\tilde{S}(s(t), J_s, \lambda_s)\right) \qquad \square$$

Next we consider $\text{OPT}(\tilde{S}(t, J', \lambda'))$. The set J' again may be split into a part going to the inactive and a part going to the nearly inactive machines, while the nearly inactive jobs $\tilde{J}_t$ have to be split into a part going to the inactive and a part going to the active machines (note that in this case (5.7) does not hold). Therefore, we set $\Psi(J')$ to be the set of pairs (A, X) with $J' = A \dot{\cup} X$, $A \cap J' \in \Gamma(J', \tilde{M}_t)$, $A \cap \tilde{J}_t \in \Gamma(\tilde{J}_t, M_t)$ and $X \in \Gamma(\tilde{J}_t \cup J', \check{M}_t)$. The splitting of λ' is more complicated as well because in this case all of the active machines may receive load from the nearly inactive jobs, and the nearly inactive machines may additionally receive load from the active but not nearly inactive jobs ((5.5) does not hold). Therefore, we set $\Upsilon(\lambda', (A, X))$ to be the set of triplets (α, β, ξ) with $\alpha \in \Lambda(A \cap J', \tilde{M}_t)$, $\beta \in \Lambda(A \cap \tilde{J}_t, M_t)$, $\xi \in \Lambda(S(t, X), M_t)$ and $\lambda' = \tau_{\tilde{M}_t, M_t}(\alpha) + \beta + \xi$.

LEMMA 5.16. *We have:*

$$\text{OPT}(\tilde{S}(t, J', \lambda')) = \min_{(A, X), (\alpha, \beta, \xi)} \max\left(\{\text{OPT}(S(t, X, \xi))\} \cup \{\lambda'(i) \mid i \in M_t\}\right)$$

$\square$

Note that the sets $\Lambda(A \cap J', \tilde{M}_t)$ and $\Lambda(A \cap \tilde{J}_t, M_t)$ can be computed in time $L^{O(k)}$ using the second dynamic program described in Section 5.2, if L is again a bound on the number of distinct loads that can occur on each machine.

ANALYSIS. Using above arguments, we can design dynamic programs that use the simple dynamic programs from Section 5.2 as subroutines.

In case of the primal graph, all the considered sets, that is, $\Gamma(J_t, \check{M}_t)$, $\Gamma(J_t \setminus \tilde{J}_t, \check{M}_t \cup \tilde{M}_t)$, $\Phi(J)$, and $\Psi(J')$, are upper bounded in cardinality by 2^{k+1}. This yields a running time of $2^{\mathcal{O}(k)} \times \mathcal{O}(m|V(T)|)$. Note that a tree decomposition exist whose number of nodes is linear in the number of vertices of the original graph (see, e.g., [18, 99]).

For the dual graph, note we have an upper bound of L^{k+1} for the cardinality of each of the considered sets, that is, $\Lambda(S(t), M_t)$, $\Lambda(\tilde{S}(t), M_t)$, $\Lambda(\tilde{J}_t, M_t)$, $\Xi(\lambda)$, and $\Upsilon(\lambda')$. Furthermore, the above considerations also imply how the sets can be computed for each node. We get a running time of $L^{\mathcal{O}(k)} \times \mathcal{O}(n|V(G)|)$ in this case.

Lastly, the cardinality of each of the occurring sets for the case of the incidence graph can be upper bounded by 2^{k+1} or L^{k+1} in the same manner. Assuming $L \geqslant 2$, we get a running time of $L^{\mathcal{O}(k)} \times \mathcal{O}(|V(G)|)$.

Optimal schedules can be found via backtracking proving the Theorems 5.1 and 5.2. Theorem 5.3 follows by the combination of the dynamic programs and a rounding scheme similar to that in Section 5.2. Note that in some sense the above results *lift* the basic dynamic programs to a more general setting.

5.4 CLIQUE- AND RANKWIDTH RESULTS

In this section, we will utilize the rankwidth and a corresponding branch decomposition. The result for the cliquewidth is due to the fact that $\mathrm{rw}(G) \leqslant \mathrm{cw}(G) \leqslant 2^{\mathrm{rw}(G)+1} - 1$ [123].

First, we want to argue that there is not much to be gained when considering primal or dual graphs with bounded rankwidth (or cliquewidth). For this, consider any instance I of the restricted assignment problem. By adding a job with processing time $\mathrm{OPT}(I)$ that can be processed on every machine, and a machine that can only process this new job, we get a modified instance I'. Any schedule for one of the instances can trivially be transformed into a schedule for the other without an increase in the makespan. However, while the rankwidth of the primal or dual graph of I could have been arbitrarily high, the rankwidth of the primal and dual graph of I' are both equal to one, because these graphs are complete.

We study the case when the rankwidth of the incidence graph is bounded by a constant k. Moreover, we assume that also the number d of distinct job sizes is bounded by a constant which we can do because of the following result. Let $\mathcal{I}$ be some class of instances of restricted assignment which is invariant with respect to changing the processing times of jobs and the introduction of copies of jobs.

LEMMA 5.17 (ROUNDING LEMMA). *If there is a PTAS for instances from $\mathcal{I}$ for which the number of distinct processing times is bounded by a constant, then there is also a PTAS for any instance from $\mathcal{I}$.*

Proof. Let $I \in \mathcal{I}$, $\varepsilon > 0$ and B an upper bound of $\mathrm{OPT}(I)$ with $B \leqslant 2\,\mathrm{OPT}$. Such a bound B can be found in polynomial time for example with the 2-approximation by Lenstra et al. [108]. Moreover, let $\delta := \min\{1/3, \varepsilon/7\}$. We call jobs j *big*, if $p_j > \delta B$ and otherwise *small*. Next, we construct a modified instance I'. This instance has the same machine set and for each big job j a job j' with the same restrictions and processing time $p_{j'} := \delta^2 B \lceil \frac{p_j}{\delta^2 B} \rceil$ is included in the job set. This yields $p'_j \leqslant p_j + \delta^2 B \leqslant (1+\delta)p_j$. For each small job j in I, we introduce $\lceil \frac{n p_j}{\delta B} \rceil \in \mathcal{O}(n)$ many jobs with the same restrictions as j and with processing time $\frac{\delta B}{n}$. Note that I' has a has at most $1/\delta + 1$ many distinct processing times and that $I' \in \mathcal{I}$. Moreover, the size of I' is polynomial in the size of I.

Given an optimal solution of I, consider the solution of I' we get by scheduling both the big and the small jobs in I' the same way as their analogues in I. The big jobs on a machine can cause an increase of the processing time of at most factor $(1+\delta)$, while for each small job of I there may be an increase of at most $\frac{\delta B}{n}$. Therefore, we get:

$$\mathrm{OPT}(I') \leqslant \mathrm{OPT}(I) + \delta\,\mathrm{OPT}(I) + \delta B \leqslant (1+3\delta)\,\mathrm{OPT}(I)$$

Now given a PTAS for instances of $\mathcal{I}$ for which the number of distinct processing times is bounded by a constant, we can compute a schedule σ' for I' with $C_{\max}(\sigma) \leqslant (1+\delta)\,\mathrm{OPT}(I')$ in polynomial time. We use σ' to construct a schedule for σ for I. In this schedule the big jobs are assigned in the same way as there analogous in I'. For the small jobs we need some additional consideration. Let S and S' be the set of small jobs in I and I' respectively. Moreover, for $j \in S$ let $S'(j)$ be the set of small jobs that were inserted in I' due to j. The assignment of $S(j)$ in σ' can be seen as a *fractional* assignment of j. We find a rounding for this fractional assignment of the small jobs. For each machine i and small job $j \in S$ let $x'_{ij} = |\{j' \in S(j) \mid \sigma'(j) = i\}|/|S(j)|$. Furthermore, let t_i be the summed up processing time that machine i receives in the schedule σ' from small jobs, i.e., $t_i = |\{j' \in S' \mid \sigma'(j) = i\}| \frac{\delta B}{n}$. Then (x'_{ij}) is a solution of the following linear program:

$$\sum_{i \mathcal{M}(j)} x_{ij} = 1 \qquad \forall j \in S \tag{5.9}$$

$$\sum_{j \in S} p_j x_{ij} \leqslant t_i \qquad \forall i \in \mathcal{M} \tag{5.10}$$

$$x_{ij} \geqslant 1 \qquad \forall j \in S, i \in \mathcal{M}$$

Using the rounding approach by Lenstra et al. [108], we can transform this solution in polynomial time into an integral solution (x^*_{ij}) fulfilling (5.9) and instead of (5.10) the modified constraint:

$$\sum_{j \in S} p_j x_{ij} \leqslant t_i + \max_{j \in S} p_j \qquad \forall i \in \mathcal{M}$$

We set σ to assign the small jobs according to (x^*_{ij}). Since $\max_{j \in S} p_j \leqslant \delta B$, we get $C_{max}(\sigma) \leqslant C_{max}(\sigma') + \delta B$. Now, $\delta := \min\{1/3, \varepsilon/7\}$ and the above inequalities yield:

$$C_{max}(\sigma) \leqslant ((1+\delta)(1+3\delta) + 2\delta)\, \textsc{opt}(I) \leqslant (1+\varepsilon)\, \textsc{opt}(I)$$

□

For each PTAS result in this context (see [46, 97, 119, 122, 134]) some rounding and simplification approach is introduced and all of these approaches are rather similar. We remark that the above approach can be used for all of the above cases (and for other cases as well). Hence, we simplify and unify this aspect of PTAS results for variants of restricted assignment. Note, however, that the included rounding procedure is not optimized for running time. Finally note that we can indeed use the above lemma:

LEMMA 5.18. *Let* I *be some instance of restricted assignment and* I' *be another instance that was constructed from* I *by changing processing times of jobs and by the introducing copies of jobs. Then the incidence graphs of* I *and* I' *have the same rankwidth.*

Proof. First note that changing processing times does not change the incidence graph. We now assume that I' was created from I by adding exactly one copy j' of some job j. Let (T, η) be any branch decomposition for I. It suffices to construct a branch decomposition (T', η') for I' that has the same rankwidth as (T, η). For this, let v be the leaf of T corresponding to j, i.e., $\eta^{-1}(v) = j$. We define T' to be the tree we get if we add two additional nodes u and u' that are exclusively connected to v. Moreover, we define $\eta'(j) = u$, $\eta'(j') = u'$ and $\eta'(x) = \eta(x)$ for any other job or machine x. Now, let $e' \in E(T') \cap E(T)$ be some edge that was not newly added and (X, Y) and (X', Y') be the corresponding cuts in the incidence graphs of I and I', respectively. Furthermore, let $A[X, Y]$ and $A[X', Y']$ denote the corresponding adjacency matrices. Then $A[X', Y']$ can be derived from $A[X, Y]$ by copying the column or row corresponding to j in order to fill the column or row corresponding to j'. Hence, the two matrices have the same rank. Moreover, the rank corresponding to matrices associated with the newly added edges is 1 since they are connected to leafs. □

DYNAMIC PROGRAM. We present a dynamic program to solve the restricted assignment problem using a branch decomposition (T, η)

with rankwidth k for the incidence graph. First, we give some intuition on why a bounded rankwidth is useful.

For any Graph (V, E) and $X \subseteq V$, we say that $u, v \in V$ have the same *connection type with respect to* X if $N(u) \cap X = N(v) \cap X$ ($N(w)$ denotes the neighborhood of a vertex w). If X is clear from the context, we say that u and v have the same connection type. Now, let $e = \{a, b\} \in E(T)$ be some edge of the branch decomposition and $\{X_{e,a}, X_{e,b}\}$ the respective cut of T, i.e., $X_{e,x}$ for $x \in \{a, b\}$ is the set of vertices of T that are in the same connected component as x when the edge e is removed. Then $\{X_{e,a}, X_{e,b}\}$ induces a partition of both the jobs and machines by $J_{e,x} := \{j \in \mathcal{J} \mid \eta(j) \in X_{e,x}\}$ and $M_{e,x} := \{i \in \mathcal{M} \mid \eta(j) \in X_{e,x}\}$ for $x \in \{a, b\}$. We will utilize the following observation:

Remark 5.19. Let $x, y \in \{a, b\}$ with $x \neq y$. The number of distinct connection types of $J_{e,x}$ with respect to $M_{e,y}$ is bounded by 2^k.

Proof. This due to the definition of the rankwidth and the simple fact that there are only 2^k distinct linear combinations of k vectors over the field with two elements GF(2). □

In the rest of this section, we first show how the branch decomposition can be used in a straightforward way to solve restricted assignment (with exponential running time). The basic idea for this is that each edge of the decomposition corresponds to a partition of the job and machine sets and an optimal solution may be found by trying all possible ways of moving jobs between partitions. At the machine-leafs, all arriving jobs have to be processed with no jobs going out, and at the job-leafs, all jobs have to be send away with no jobs coming in. From this, the procedure can work up to some root edge. Next, we argue that it is sufficient to consider only certain locally defined classes of job sets. The crucial part here is that the number of these classes can be polynomially bounded because the number of distinct sizes and connection types of jobs are constant.

JOB SETS. Let $e = \{a, b\} \in E(T)$ again be some edge of the tree T and $\{X_{e,a}, X_{e,b}\}$ the corresponding cut of T. We fix a schedule σ and make some basic observations. There is a set of jobs $\vec{J}_{a,b} \subseteq J_{e,a}$ that σ assigns to machines from $M_{e,b}$. We will use the intuition that $\vec{J}_{a,b}$ is sent through e from a to b (see also Figure 5.2). The node b may be an inner node or a leaf. Moreover, if b is a *leaf*, $\eta^{-1}(t)$ may be a job j^* or a machine i^*. In the first case, σ sends no jobs to t and j^* to a. In the second case, no jobs are sent to a and the jobs send to b should be feasible on i^*. Now, any set that is sent through an edge and arrives at an *internal node* will be split into two parts: one going forth through the second and one through the third edge. And looking at it the other way around: Any set that is sent by a schedule through an edge coming from an inner node, is put together from two parts, one coming from the second and one coming from the third edge.

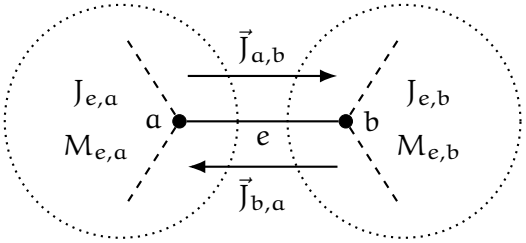


Figure 5.2: Edge e slits the instance into two parts. In a given schedule, some jobs from one part may be processed in the other. We use the intuition that these jobs are send through the edge.

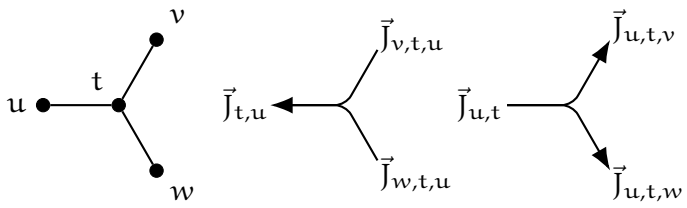


Figure 5.3: In a given schedule, the jobs that are send through an edge coming from or going to an inner node may be partitioned into the jobs that came from or will continue through the two incident edges, respectively.

We formalize this notion. Let t be an internal node of T with neighbors $u, v, w \in V(T)$. Then, for each pair of neighbors x, y of t, there are job sets $\vec{J}_{x,t,y} \subseteq \vec{J}_{x,t} \cap \vec{J}_{t,y}$ such that:

$$\vec{J}_{u,t} = \vec{J}_{u,t,v} \dot{\cup} \vec{J}_{u,t,w} \qquad \vec{J}_{t,u} = \vec{J}_{v,t,u} \dot{\cup} \vec{J}_{w,t,u} \tag{5.11}$$

See also Figure 5.3. It is rather easy to see that sets $\vec{J}_{s,t}$ that are feasible at the leafs and fulfill the conditions (5.11) uniquely define a feasible schedule.

Using these observations, we now discuss how (the value of) an optimal schedule can be found by considering different job sets that may be sent through the edges. For this, we use an intuition of up and down with a above and b below. Let $\check{J} = \vec{J}_{a,b} \subseteq J_{e,a}$ and $\hat{J} = \vec{J}_{b,a} \subseteq J_{e,b}$ be some candidate sets to be sent up and down, respectively, through e. We set $I_{e,x}(\hat{J}, \check{J}) = I[(J_{e,x} \setminus \vec{J}_{x,y}) \cup \vec{J}_{y,x}, M_{e,x}]$ for $x, y \in \{a, b\}$ with $x \neq y$, i.e., the subinstances of I induced by e if $\hat{J}$ and $\check{J}$ are send up or down, respectively. Note that the instance I is split into the two subinstances. Moreover, let Γ_e be the set of pairs $(\hat{J}, \check{J})$. Then:

$$\textsc{opt}(I) = \min_{(\hat{J},\check{J}) \in \Gamma_e} \max\{\textsc{opt}(I_{e,a}(\hat{J}, \check{J})), \textsc{opt}(I_{e,b}(\hat{J}, \check{J}))\} \tag{5.12}$$

We now consider the computation of $\textsc{opt}(I_{e,b}(\hat{J}, \check{J}))$ for the two cases when b is an internal node or a leaf. If b is a leaf, it may correspond to a job or a machine, i.e., $\eta^{-1}(b) = j^* \in \mathcal{J}$ or $\eta^{-1}(b) = i^* \in \mathcal{M}$. In the first case, we have $\{j^*\} = J_{e,b}$ and get $\textsc{opt}(I_{e,b}(\hat{J}, \check{J})) = \infty$ if $\hat{J} \neq \{j^*\}$ or $\check{J} \neq \emptyset$ and $\textsc{opt}(I_{e,b}(\hat{J}, \check{J})) = 0$ otherwise. In the second case, $\hat{J}$ is

empty since there are no jobs at b. We get $\text{OPT}(I_{e,b}(\hat{J},\check{J})) = \sum_{j\in\check{J}} p_j$ if $\check{J} \subseteq \mathcal{J}(i^*)$ and $\text{OPT}(I_{e,b}(\hat{J},\check{J})) = \infty$ otherwise.

Now, let b be an internal node that is connected to two lower nodes ℓ and r via edges e_ℓ and e_r (see Figure 5.4). We say that ℓ and e_ℓ are

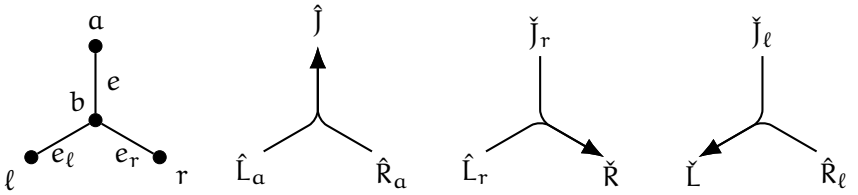


Figure 5.4: An inner node b and certain job sets that are send through the incident edges.

on the left, while r and e_r are on the right. Recursively, we assume that for any $(\hat{L},\check{L}) \in \Gamma_{e_\ell}$ and $(\hat{R},\check{R}) \in \Gamma_{e_r}$ we know $\text{OPT}(I_{e_\ell,\ell}(\hat{L},\check{L}))$ and $\text{OPT}(I_{e_r,r}(\hat{R},\check{R}))$, respectively. We want to identify the set $\Lambda_e(\hat{J},\check{J})$ of tuples $(\hat{L},\check{L},\hat{R},\check{R})$ that for fixed $(\hat{J},\check{J})$ may occur in a schedule, i.e., fulfill condition (5.11) for all edges from $\{e, e_\ell, e_r\}$. For $\hat{J}$, it is clear which part is coming from the left and which from the right and we set $\hat{L}_a \subseteq J_{e_\ell,\ell}$ and $\hat{R}_a \subseteq J_{e_r,r}$ accordingly such that $\hat{J} = \hat{L}_a \dot\cup \hat{R}_a$. The other four sets in which the job sets going up and down could be split can all be tried. More precisely, for each $\check{J}_\ell, \check{J}_r \subseteq \check{J}$ with $\check{J} = \check{J}_\ell \dot\cup \check{J}_r$, $\hat{L}_r \subseteq J_{e_\ell,\ell} \setminus \hat{J}_\ell$ and $\hat{R}_\ell \subseteq J_{e_r,r} \setminus \hat{J}_r$ the tuple $(\hat{L}_r \dot\cup \hat{L}_a, \check{J}_\ell \dot\cup \hat{R}_\ell, \hat{R}_\ell \dot\cup \hat{R}_a, \check{J}_r \dot\cup \hat{L}_r)$ is in $\Lambda_e(\hat{J},\check{J})$ and the set is defined by such tuples (see Figure 5.4). We get:

$$\text{OPT}(I_{e,b}(\hat{J},\check{J})) = \min_{(\hat{L},\check{L},\hat{R},\check{R})} \max\{\text{OPT}(I_{e_\ell,\ell}(\hat{L},\check{L})), \text{OPT}(I_{e_r,r}(\hat{R},\check{R}))\} \tag{5.13}$$

Using these considerations, restricted assignment can be solved by choosing a root edge $e^* = \{a^*, b^*\}$ that is incident to a leaf a^* corresponding to a job and designing a dynamic program working from leaf edges to the root edge using (5.13). Now, (5.12) for $e = e^*$ together with the considerations for leaf nodes yield $\text{OPT}(I) = \text{OPT}(I_{e^*,b^*}(\emptyset, \eta^{-1}(a^*)))$. The running time of such an algorithm is exponential in the input length.

CLASSES OF JOBS. Let $\check{J}, \check{J}' \subseteq J_{e,a}$. There are some cases in which $\check{J}$ and $\check{J}'$ are in some sense similar and it holds that $\text{OPT}(I_{e,b}(\hat{J},\check{J})) = \text{OPT}(I_{e,b}(\hat{J},\check{J}'))$. This is the case if there is a bijection $\alpha : \check{J} \to \check{J}'$ such that j and $\alpha(j)$ have the same connection type with respect to $M_{e,b}$ and $p_j = p_{\alpha(j)}$ for each $j \in \hat{J}$. By this, an equivalence relation $\sim_{e,a}$ can be defined, and analogue considerations can be made for sets that are send up yielding an equivalence relation $\sim_{e,b}$. Now the observation (5.12) can be reformulated in terms of equivalence classes:

$$\begin{aligned} &\text{OPT}(I) \\ &= \min_{([\hat{J}],[\check{J}])} \max\Big\{ \min_{\check{J}'\in[\check{J}]} \text{OPT}(I_{e,a}(\hat{J}',\check{J})), \min_{\check{J}'\in[\check{J}]} \text{OPT}(I_{e,b}(\hat{J},\check{J}')) \Big\} \end{aligned} \tag{5.14}$$

In this equation we consider equivalence classes $[\check{J}]$ and $[\hat{J}]$ belonging to the relations $\sim_{e,a}$ and $\sim_{e,b}$, respectively. Moreover, $\hat{J}$ and $\check{J}$ are arbitrary representatives of these classes. We will now develop a sensible representation for the equivalence classes.

We drop the notion of up and down for the following considerations, i.e., $b \in e$ is just one of two nodes of some edge e. We fix some ordering of the different processing times with $p(i)$ denoting the i-th processing time for $i \in [d]$. Any set of jobs J' induces a vector $\lambda \in \mathbb{Z}^d_{\geqslant 0}$ where λ_i is the number of jobs in J' that have the i-th processing time, i.e., $\lambda_i = |\{j \in J' \mid p_j = p(i)\}|$. We set $p(\lambda) = \sum_{i\in[d]} p(i)\lambda_i$. Let $\kappa(e,b)$ be the number of connection types of jobs from $J_{e,b}$ with respect to $M_{e,a}$. Note that due to Remark 5.19 we get $\kappa(e,b) \leqslant 2^k$. Again, we fix some ordering of the connection types. For $i \in [\kappa(e,b)]$ let $\varphi_{e,b}(i)$ be the size vector induced by the i-th connection type of $J_{e,b}$ with respect to $M_{e,a}$ and $M_{e,a}(i) \subseteq M_{e,a}$ the machines from $M_{e,a}$ the respective jobs may be processed on. Furthermore, the concatenation of these size vectors is denoted by $\varphi_{e,b}$, that is, with a slight abuse of notation we may write $\varphi_{e,b} = (\varphi_{e,b}(1), \dots, \varphi_{e,b}(\kappa(e,b)))$. Now, the equivalence classes of $\sim_{e,b}$ can naturally be represented and characterized by vectors $\iota \leqslant \varphi_{e,b}$. We have:

Remark 5.20. For each $e \in E(T)$ and $b \in e$ there are at most $n^{\kappa(e,b)d}$ different vectors $\iota \leqslant \varphi_{e,b}$.

We now study the splitting behaviour of job classes at inner nodes. Consider a set J' that is sent through an edge $f = \{u,v\}$ and then forth through an incident edge $g = \{v,w\}$. Then, there are vectors ι_f and ι_g representing J' in the context of f and g respectively. Now, let J'' be some other set represented by ι_f in the context of f. Then J'' will also be represented by ι_g in the context of g, that is, ι_f translates uniquely into ι_g. We formalize this notion by the definition of a translation function $\tau_{f,g} : \{\iota \mid \iota \leqslant \varphi_{f,u}\} \to \{\iota' \mid \iota' \leqslant \varphi_{g,v}\}$. For each $i \in [\kappa(f,u)]$, there is a unique $i' \in [\kappa(g,v)]$ with $M_{f,v}(i) \cap M_{g,w} = M_{g,w}(i')$, i.e., the i-th connection type of $J_{f,u}$ translates into the i'-th connection type of $J_{g,v}$. Let $\xi_{f,g} : [\kappa(f,u)] \to [\kappa(g,v)]$ be given by $i \mapsto i'$. For each $\iota \leqslant \varphi_{f,u}$ and $i' \in [\kappa(g,v)]$, let $\iota'(i') \in \mathbb{Z}^d_{\geqslant 0}$ be given by $\iota'(i') = \sum_{i\in\xi^{-1}_{f,g}(i')} \iota(i)$. We set $\tau_{f,g}(\iota) = (\iota'(1), \dots, \iota'(\kappa(g,v)))$

With this, we can formulate an analogue of (5.11) for job classes. Hence, we again fix some schedule σ. Let $\iota_{a,b}$ be the representative of the set of jobs that σ sends from a to b. Moreover, let t be an inner node with neighbors u, v, w. For neighbors x, y of t, the set $\vec{J}_{x,t,y}$ considered in the last paragraph now has a representative both in the context of $\{x,t\}$ and $\{t,y\}$. Fixing the first one $\iota_{x,t,y} \leqslant \iota_{x,t}$, the second one can be obtained via the transformation function, yielding:

$$\begin{aligned} \iota_{u,t} &= \iota_{u,t,v} + \iota_{u,t,w} \\ \iota_{t,u} &= \tau_{(\{v,t\},\{u,t\})}(\iota_{v,t,u}) + \tau_{(\{w,t\},\{u,t\})}(\iota_{w,t,u}) \end{aligned} \tag{5.15}$$

We now return to our notion of up and down ($e = \{a, b\} \in E(T)$ with a above and b below). Let $\hat{\iota} \leqslant \varphi_{e,b}$ and $\check{\iota} \leqslant \varphi_{e,a}$ be candidate job classes to be send up and down e. Considering (5.14), we set $\textsc{opt}(e, \hat{\iota}, \check{\iota}) = \min_{\hat{J}' \in [\hat{J}]} \textsc{opt}(I_{e,a}(\hat{J}', \check{J}))$ where $\hat{\iota}$ and $\check{\iota}$ represent $[\hat{J}]$ and $[\check{J}]$, respectively.

For the case when b is a leaf not much changes. If $\eta^{-1}(b)$ is a job j^*, the class of $\{j^*\}$ has only one element and is represented by $\varphi_{e,b}$. Therefore, we get that $\textsc{opt}(e, \hat{\iota}, \check{\iota}) = 0$ for $\hat{\iota} = \varphi_{e,b}$ and $\check{\iota} = 0$, and $\textsc{opt}(e, \hat{\iota}, \check{\iota}) = \infty$ otherwise. If $\eta^{-1}(b)$ is a machine i^*, we have $\varphi_{e,b} = 0$ and there are only two possible connection types for jobs from $J_{e,a}$ because jobs can be processed on i^* or not, i.e., $\kappa(e, a) \leqslant 2$. In any case, we get $\textsc{opt}(e, \hat{\iota}, \check{\iota}) = \sum_{i \in [\kappa(e,a)]} p(\check{\iota}(i))$ (remember that $p(\lambda) = \sum_{i \in [d]} p(i)\lambda_i$ for any vector λ of job size multiplicities).

Now, let b be an inner node again with lower neighbors ℓ and r to which it is connected via edges e_ℓ and e_r. We may assume that we know the values $\textsc{opt}(e_\ell, \hat{\lambda}, \check{\lambda})$ and $\textsc{opt}(e_r, \hat{\rho}, \check{\rho})$ for candidate job classes $(\hat{\lambda}, \check{\lambda}, \hat{\rho}, \check{\rho})$ to go up or down the left or right edge, respectively. We want to identify the set $\Xi_e(\hat{\iota}, \check{\iota})$ of quadruples $(\hat{\lambda}, \check{\lambda}, \hat{\rho}, \check{\rho})$ that are compatible with $\hat{\iota}$ and $\check{\iota}$, i.e., that fulfill (5.15). For this, let $\check{\iota}_\ell, \check{\iota}_r \leqslant \check{\iota}$ with $\check{\iota}_\ell + \check{\iota}_r = \check{\iota}$, $\hat{\lambda}_\ell, \hat{\lambda}_r \leqslant \varphi_{e_\ell,\ell}$ with $\hat{\lambda}_\ell + \hat{\lambda}_r \leqslant \varphi_{e_\ell,\ell}$, and $\hat{\rho}_\ell, \hat{\rho}_r \leqslant \varphi_{e_r,r}$ with $\hat{\rho}_\ell + \hat{\rho}_r \leqslant \varphi_{e_r,r}$ such that $\tau_{e_\ell,e}(\hat{\lambda}_\ell) + \tau_{e_r,e}(\hat{\rho}_r) = \hat{\iota}$. By setting $\hat{\lambda} = \hat{\lambda}_\ell + \hat{\lambda}_r$, $\check{\lambda} = \tau_{e,e_\ell}(\check{\iota}_\ell) + \tau_{e_r,e_\ell}(\hat{\rho}_\ell)$, $\hat{\rho} = \hat{\rho}_\ell + \hat{\rho}_r$ and $\check{\rho} = \tau_{e,e_r}(\check{\iota}_r) + \tau_{e_\ell,e_r}(\hat{\lambda}_r))$ we get such a tuple and the set $\Xi_e(\hat{\iota}, \check{\iota})$ is defined by such tuples.

LEMMA 5.21. *We have:*

$$\textsc{opt}(e, \hat{\iota}, \check{\iota}) = \min_{(\hat{\lambda},\check{\lambda},\hat{\rho},\check{\rho})} \max\bigl\{\textsc{opt}(e_\ell, \hat{\lambda}, \check{\lambda}), \textsc{opt}(e_r, \hat{\rho}, \check{\rho})\bigr\}$$

Proof. If the left-hand side equals ∞, it is not hard to see that the equation holds, and we therefore assume $\textsc{opt}(e, \hat{\iota}, \check{\iota}) < \infty$. For given $\hat{\iota} \leqslant \varphi_{e,a}$ and $\check{\iota} \leqslant \varphi_{e,b}$, let $\check{J} \subseteq J_{e,a}$ be any set represented by $\check{\iota}$ and $\hat{J}' \subseteq J_{e,b}$ be an optimal one represented by $\hat{\iota}$, i.e., $\textsc{opt}(e, \hat{\iota}, \check{\iota}) = \textsc{opt}(I_{e,b}(\hat{J}', \check{J}))$. Let σ^* be an optimal schedule for $I_{e,b}(\hat{J}', \check{J})$. Furthermore, let $\hat{L}^*$, $\check{L}^*$, $\hat{R}^*$ and $\check{R}^*$ be the sets that σ^* sends up or down through e_ℓ or e_r, respectively, and let $\hat{\lambda}^*$, $\check{\lambda}^*$, $\hat{\rho}^*$ and $\check{\rho}^*$ be the representatives of their classes. Than σ^* induces schedules σ^*_ℓ and σ^*_r for $I_{e,\ell}(\hat{L}^*, \check{L}^*)$ and $I_{e,r}(\hat{R}^*, \check{R}^*)$. We get:

$$\begin{aligned} C_{\max}(\sigma^*) &= \max\{C_{\max}(\sigma^*_\ell), C_{\max}(\sigma^*_r)\} \\ &\geqslant \max\{\textsc{opt}(e_\ell, \hat{\lambda}^*, \check{\lambda}^*), \textsc{opt}(e_r, \hat{\rho}^*, \check{\rho}^*)\} \\ &\geqslant \min_{\substack{(\hat{\lambda},\check{\lambda},\hat{\rho},\check{\rho}) \in \\ \Xi_e(\hat{\iota},\check{\iota})}} \max\{\textsc{opt}(e_\ell, \hat{\lambda}, \check{\lambda}), \textsc{opt}(e_r, \hat{\rho}, \check{\rho})\} \end{aligned}$$

Now, we choose $(\hat{\lambda}, \check{\lambda}, \hat{\rho}, \check{\rho}) \in \Xi_e(\hat{\iota}, \check{\iota})$ minimizing the last term in the above expression with the corresponding splitting vectors $\check{\iota}_\ell$, $\check{\iota}_r$, $\hat{\lambda}_\ell$, $\hat{\lambda}_r$, $\hat{\rho}_\ell$, $\hat{\rho}_r$. Moreover, let $\hat{L}'$ and $\hat{R}'$ be optimal sets represented by $\hat{\lambda}$ and $\hat{\rho}$, respectively. Splitting $\hat{L}'$, $\hat{R}'$ and $\check{J}$ corresponding to the

splitting of their job classes we can obtain sets $\check{L}$, $\check{R}$ and $\hat{J}$ that are represented by $\check{\lambda}$, $\check{\rho}$ and $\hat{\iota}$ respectively and fulfill (5.11). We now have $\text{OPT}(e_\ell, \hat{\lambda}, \check{\lambda}) = \text{OPT}(I_{e_\ell,\ell}(\hat{L}', \check{L}))$ and $\text{OPT}(e_r, \hat{\rho}, \check{\rho}) = \text{OPT}(I_{e_r,r}(\hat{R}', \check{R}))$. Let σ_ℓ and σ_r be respective optimal schedules. Than $\sigma := \sigma_\ell \cup \sigma_r$ is a schedule for $I_{e,b}(\hat{J}, \check{J})$ and we have:

$$C_{\max}(\sigma) = \min_{(\hat{\lambda},\check{\lambda},\hat{\rho},\check{\rho})} \max\{\text{OPT}(e_\ell, \hat{\lambda}, \check{\lambda}), \text{OPT}(e_r, \hat{\rho}, \check{\rho})\}$$

Hence, we have $C_{\max}(\sigma^*) \geqslant C_{\max}(\sigma)$. Since σ^* was chosen optimal with an optimal class representative, we have furthermore $C_{\max}(\sigma) \geqslant C_{\max}(\sigma^*)$. This yields the claimed equation. Moreover, we get that $\hat{J}$ is optimal as well. □

RESULTS. With these considerations a dynamic program for restricted assignment can be defined. This can be done in a way such that its running time is in $m^2 n^{\mathcal{O}(d2^k)}$ proving Theorem 5.4 together with Lemma 5.17 (the rounding lemma) and the considerations of Section 5.2.

BI-COGRAPHS We show that the hierarchical, tree-hierarchical and nested cases are all special cases of the case that the incidence graph is a bi-cograph. Bi-cographs were introduced as a bipartite analogue of cographs [53].

DEFINITION 5.22. Let $G = (A \dot\cup B, E)$ be a bipartite graph. The *bi-complement* of G is the graph $(A \dot\cup B, \{\{a, b\} \mid a \in A, b \in B, \{a, b\} \notin E\})$. A graph is called *bi-cograph* if and only if it is bipartite and can be reduced to isolated vertices by recursively bi-complementing its connected bipartite subgraphs.

It is known [54] that their cliquewidth and therefore also their rankwidth is bounded by 4. Furthermore, a certain decomposition of a given bi-cograph similar to cotrees of cographs can be found very efficiently by recursively bi-complementing the connected bipartite subgraphs [53]. This decomposition can easily be turned into a branch decomposition for which in the application studied here the number of connection types of jobs $\kappa(e, u)$ for every edge e of the decomposition and $v \in e$ is bounded by 2.

LEMMA 5.23. *Let* I *be an instance of restricted assignment with hierarchical, tree-hierarchical or nested restrictions. Then the incidence graph of* I *is a bi-cograph.*

Proof. We first consider the case that I has tree-hierarchical restrictions. Let T be a corresponding rooted tree with $V(T) = \mathcal{M}$. Then there is at least one machine (the root of T) that can process all jobs. After bi-complementing the connected bipartite subgraphs of the incidence graph this machine is isolated. This can be repeated: After bi-complementing two more times the nearest descendants of the root

in T that cannot process all jobs will be isolated. Iterating this, at some point all machines and therefore also all jobs will be isolated.

Now, let I be an instance with nested restrictions. Note that the jobs $j \in \mathcal{J}$ with maximal $\mathcal{M}(j)$ (with respect to $\subseteq$) are all in different connected components of the incidence graph and connected to all machines in their component. Hence, they are isolated after bi-complementing the first time. If we bi-complement a second time and remove these jobs, we get a new instance with nested restrictions and less jobs. By iterating this argument the claim follows. □

5.5 OTHER OBJECTIVE FUNCTIONS

We briefly discuss the applicability of our results for other objective functions that have prominently been studied for scheduling on unrelated parallel machines and restricted assignment. Namely, we consider the ℓ_p-norm of machines loads, i.e., $\|\lambda^\sigma\|_p = (\sum_{i\in\mathcal{M}}(\lambda_i^\sigma)^p)^{1/p}$ and $\lambda_i^\sigma = \sum_{j\in\sigma^{-1}(i)} p_{ij}$; and the minimum machine load $C_{\min}(\sigma) = \min_i \sum_{j\in\sigma^{-1}(i)} p_{ij}$.

TREEWIDTH RESULTS. The basic dynamic programs discussed in Section 5.2 are used as subroutines in the dynamic programs utilizing tree decomposition. Hence, our first step is to argue that they can be adapted for the changed objective functions.

The first dynamic program iterates through the machines. Like before, we denote the objective value of an optimal scheduling of the instance given by the first i machines and all jobs except J with $\text{OPT}(i, J) = \text{OPT}(I[\mathcal{J} \setminus J, [i]])$. However, now we consider the maximization of $C_{\min}$ and the minimization of $\sum_i \lambda_i^p$, respectively. Note that minimizing $\sum_i \lambda_i^p$ is equivalent to minimizing $\|\lambda\|_p$. We have

$$\text{OPT}(i, J) = \max_{J\subseteq J'\subseteq\mathcal{J}} \min\Big\{\text{OPT}(i-1, J'), \sum_{j\in J'\setminus J} p_{ij}\Big\}$$

in the context of $C_{\min}$, and

$$\text{OPT}(i, J) = \min_{J\subseteq J'\subseteq\mathcal{J}} \Big(\text{OPT}(i-1, J') + \big(\sum_{j\in J'\setminus J} p_{ij}\big)^p\Big)$$

in the context of $\sum_i \lambda_i^p$.

The second dynamic program is based on computing possible load vectors for the first j jobs. It is obvious that the respective objective values can be computed directly from the load vectors and the dynamic program works for both objective functions as well.

Using the second dynamic programs to get an FPTAS, however, is a little bit more complicated than in the unrelated scheduling case for both objective functions. For $C_{\min}$ we can compute a bound B with $B \leqslant \text{OPT} \leqslant (n-m-1)B$ using the approximation algorithm by Bezáková and Dani [15]. Note that the load on some machines

might be arbitrarily high even for an optimal solution, but we can bound the load values nevertheless: If a processing time is bigger than $(n-m-1)B$, we can reduce it to this value. Then, if assigning a job to a machine would raise its load to a value bigger than $2(n-m-1)B$, its load was already bigger than $(n-m-1)B$ and we do not have to consider this possibility. Now, the situation could occur that a job cannot be assigned to any machine because the load would exceed $2(n-m-1)B$ for each of them. If this is the case, a corresponding partial schedule is already optimal, and all remaining jobs can be assigned arbitrarily. Hence, we can round the processing times down to the next multiple of $\frac{\varepsilon}{n(1+\varepsilon)}B$, and get $\mathcal{O}(n(n-m-1)/\varepsilon)$ many possible distinct load values and therefore an FPTAS with running time $(n^2/\varepsilon)^{\mathcal{O}(m)} \times \mathcal{O}(n)$.

For the case of $\|\lambda\|_p$ minimization, Azar et al. [12] pointed out how the triangle inequality and the convexity of the norm function can be utilized to get an FPTAS as well. We briefly discuss the needed ideas. Let $P = \sum_{j\in\mathcal{J}} \min\{p_{ij} \mid i \in \mathcal{M}\}$, λ a load vector that is fulfilled by some schedule that assigns all jobs on a machine where its processing time is minimal, and λ^* a load vector that is fulfilled by some optimal schedule. We have $\|\lambda^*\|_p \leqslant \|\lambda\|_p \leqslant \|\lambda\|_1 = P$ and this implies $\lambda^*_i \leqslant P$ for each machine $i \in \mathcal{M}$. On the other hand, we have $\|\lambda^*\|_p \geqslant P$ and therefore $\|\lambda^*\|_p \geqslant (\sum_{i\in\mathcal{M}}(P/m)^p)^{1/p} = m^{1/p}P/m$. Hence, rounding the processing times up to the next integer multiple of $\varepsilon P/nm$ and bounding the load values by $(1+\varepsilon/nm)P$ yields an FPTAS with running time $(nm/\varepsilon)^{\mathcal{O}(m)} \times \mathcal{O}(n)$.

The dynamic programs that utilize the treewidth in some sense *lift* the basic dynamic programs to a more general setting. It is easy to verify that this works for the other objectives as well. Essentially, in the recurrence relations the role of maximization and minimization has to be switched for $C_{\min}$, and the maximizations have to be replaced by summations for $\|\lambda\|_p$. For example the recurrence relations for the primal graph described in Lemma 5.10 and 5.11 with analogue notation translate to

$$\begin{aligned}
\textsc{opt}(S(t,J)) &= \max_{(J_\ell,J_r)\in\Phi(J)} \min_{s\in\{\ell,r\}} \textsc{opt}(\tilde{S}(s(t),J_s)) \\
\textsc{opt}(\tilde{S}(t,J')) &= \max_{(A,X)\in\Psi(J')} \min\bigl\{\textsc{opt}(S(t,X)), \textsc{opt}(I[A,\tilde{M}_t])\bigr\}
\end{aligned}$$

for the case of $C_{\min}$ maximization and

$$\begin{aligned}
\textsc{opt}(S(t,J)) &= \min_{(J_\ell,J_r)\in\Phi(J)} \sum_{s\in\{\ell,r\}} \textsc{opt}(\tilde{S}(s(t),J_s)) \\
\textsc{opt}(\tilde{S}(t,J')) &= \max_{(A,X)\in\Psi(J')} \textsc{opt}(S(t,X)) + \textsc{opt}(I[A,\tilde{M}_t])
\end{aligned}$$

for $\sum_i \lambda_i^p$ minimization.

Summarizing, all the results concerning the treewidth can be transferred to the other objectives as well.

CLIQUE- AND RANKWIDTH RESULTS. Like for the treewidth results, the dynamic program utilizing the rankwidth and the branch decomposition has to be changed only slightly to adapt them for the changed objective functions and we do not elaborate on that. Therefore, it remains to argue that Lemma 5.17 (the rounding lemma) can be adapted.

For the case of $C_{\min}$ minimization, this can be done in a straightforward fashion: There is a constant approximation algorithm [6] that can be used to get a lower bound B of the optimum and the big jobs can be defined and rounded analogously. The small jobs can be split up as well, yielding one extra job size. A schedule for the rounded instance can be transformed into a schedule for the original one with a loss of at most δB on each machine, using an adaptation of the result by Lenstra et al. [108] due to Bezáková and Dani [15].

For the case of $\|\lambda\|_p$ minimization, on the other hand, the situation is more complicated. First note that there is an approximation algorithm with a constant rate [12] that can be used to get an upper bound B of the optimal schedule, and the algorithm by Lenstra et al. [108] can be used to find an integral assignment with only slightly increased load on each machine. However, an increase in the load of at most X on each machine, translates into an overall increase of $m^{1/p}X$ in the objective function, when using the triangle inequality. Hence, defining X in terms of ε and some upper bound B on the optimum will not work without additional information. Using the ideas that we introduced for the FPTAS in the treewidth case, we still get a QPTAS (an approximation scheme with quasi-polynomial running time): Let $P = \sum_{j \in \mathcal{J}} p_j$ and λ^* a load vector that is fulfilled by some optimal schedule. Like before, we have $\lambda_i^* \leqslant P$ for each machine $i \in \mathcal{M}$ and $\|\lambda^*\|_p \geqslant m^{1/p}P/m$. We consider a job as big if $p_j \geqslant \delta P/m$ and as small otherwise. The small jobs are handled like before and the big jobs are rounded via a geometric rounding step, that is, $\bar{p}_j = (1+\delta)^x \delta P/m$ with $x = \lceil \log_{1+\delta}(p_j m)/(\delta P) \rceil$ yielding an instance with $\mathcal{O}(\log_{1+\delta} m/\delta)$ many processing times. Note that an increase in the load by a factor of $(1+\delta)$ on each machine increases the objective function by at most $(1+\delta)$. Hence, an optimal solution for the rounded instance gives a $(1+\varepsilon)$-approximation of the original one for $\delta = \varepsilon/2$. Solving an instance with $\mathcal{O}(\log_{1+\delta} m/\delta)$ many processing times takes $m^2 n^{\mathcal{O}(2^k \log_{1+\delta}(m/\delta))}$ time, and therefore we get a QPTAS and not a PTAS in this case.

5.6 OPEN PROBLEMS

Generally speaking, it is not fully understood which kind of restrictions do (not) allow for the design of an approximation scheme. The graph framework discussed in this chapter might be a viable tool to deepen the knowledge in this regard and, more generally, to better

understand the hardness imposed by assignment restrictions. For the primal graph, some work in this direction was already done by Page, Solis-Oba and Maack [125], who showed, e.g., that instances with a triangle-free primal graph are polynomial time solvable, while the full 1.5 inapproximability holds for diamond-free graphs.

Furthermore, the work presented in this chapter provided some insights into restricted assignment from the FPT perspective, and it seems a viable research goal to deepen this understanding.

6

MACHINE SCHEDULING WITH SETUP TIMES

6.1 INTRODUCTION

In this chapter, we present an augmented formulation of the classical integer linear program of configurations (configuration IP) and demonstrate its use in the EPTAS design for scheduling problems with setup times. Configuration IPs are widely used in the context of scheduling or packing problems in which items have to be distributed to multiple target locations. The configurations describe possible placements on a single location, and the integer linear program (IP) is used to choose a proper selection covering all items. Two fundamental problems, for which configuration IPs have prominently been used, are bin packing and machine scheduling (see Section 2.2). For bin packing, the configuration IP was introduced as early as 1961 by Gilmore and Gomory [55], and the recent results for both problems typically use configuration IPs as a core technique, see, e.g., [56, 73]. In the present work, we consider scheduling problems and therefore discuss the configuration IP in more detail using the example of machine scheduling.

CONFIGURATION IP FOR MACHINE SCHEDULING. In the problem of machine scheduling, a set $\mathcal{J}$ of n jobs is given together with processing times p_j for each job j and a number m of identical machines. The objective is to find a schedule $\sigma : \mathcal{J} \to [m]$ such that the makespan is minimized, that is, the latest finishing time of any job $C_{\max}(\sigma) = \max_{i \in [m]} \sum_{j \in \sigma^{-1}(i)} p_j$. For a given makespan bound, the configurations may be defined as multiplicity vectors indexed by the occurring processing times such that the overall length of the chosen processing times does not violate the bound. The configuration IP is then given by variables x_C for each configuration C; constraints ensuring that there is a machine for each configuration, i.e., $\sum_C x_C = m$; and further constraints due to which the jobs are covered, i.e., $\sum_C C_p x_C = |\{j \in \mathcal{J} \mid p_j = p\}|$ for each processing time p. In combination with certain simplification techniques, this type of IP is often used in the design of approximation schemes. For machine scheduling, we demonstrated this in Section 2.2 and showed that only a constant number of configurations is needed, which leads to an integer program with a constant number of variables. Integer programs of that kind can be efficiently solved using the classical algorithm by Lenstra and Kannan [88, 89], yielding an EPTAS for machine scheduling. It is well-known that machine scheduling is strongly NP-hard,

and therefore it admits no optimal polynomial time algorithm, unless P=NP, and there neither is hope for an FPTAS.

MACHINE SCHEDULING WITH CLASSES. The configuration IP is used in a wide variety of approximation schemes for machine scheduling problems [5, 73]. However, for scheduling problems where the jobs have to meet some additional requirements, such as class dependencies, the approach often ceases to work. A problem emerging, in this case, is that the additional requirements have to be represented in the configurations, resulting in a super-constant number of variables in the IP. We elaborate on this using a concrete example: Consider the variant of machine scheduling in which the jobs are partitioned into K setup classes. For each job j, a class k_j is given; and for each class k, a setup time s_k has to be paid on a machine if a job belonging to that class is scheduled on it, i.e., $C_{\max}(\sigma) = \max_{i\in[m]} \left(\sum_{j\in\sigma^{-1}(i)} p_j + \sum_{k\in\{k_j \mid j\in\sigma^{-1}(i)\}} s_k \right)$. With some effort, simplification steps similar to the ones for machine scheduling can be applied. In the course of this, the setup times as well can be bounded in number and guaranteed to be sufficiently big [74]. However, it is not hard to see that the configuration IP still cannot be trivially extended while preserving its solvability. For instance, extending the configurations with multiplicities of setup times will not work because then we have to make sure that a configuration is used for a fitting subset of classes. This would create the need to encode class information into the configurations or introduce other class dependent variables.

MODULE CONFIGURATION IP. Our approach to deal with the class dependencies of the jobs is to cover the job classes with so-called modules and cover the modules in turn with configurations in an augmented IP, called the module configuration IP (MCIP). In the setup class model, for instance, the modules may be defined as combinations of setup times and multiplicity vectors of processing times, and the configurations, in turn, as multiplicity vectors of module sizes. The number of both the modules and the configurations will typically be bounded by a constant. To cover the classes by modules, each class is provided with its own set of modules, that is, there are variables for each pair of class and module. Since the number of classes is part of the input, the number of variables in the resulting MCIP is super-constant, and therefore the algorithm by Lenstra and Kannan [88, 89] is not the proper tool for the solving of the MCIP. However, the MCIP has a certain simple structure: The mentioned variables are partitioned into uniform classes each corresponding to the set of modules, and for each class, the modules have to do essentially the same, that is, cover the jobs of the class. Utilizing these properties, we can formulate the MCIP in the framework of n-fold integer programs—a class of IPs

whose variables and constraints fulfill certain uniformity requirements. In 2013 Hemmecke, Onn, and Romanchuk [63] presented the first FPT algorithm (see Section 2.1) for n-fold IPs, that is, an algorithm with a running time $f(k) \times \text{poly}(|I|)$ where k is some parameter (or a sequence of parameters) depending in the instance. In the MCIP the corresponding parameters can be properly bounded which enables the present result. For a more detailed description of n-fold IPs and the MCIP, the reader is referred to Section 6.2 and 6.3, respectively.

Using the MCIP, we are able to formulate an EPTAS for machine scheduling in the setup class model described above. Before, only a regular PTAS with running time $nm^{\mathcal{O}(1/\varepsilon^5)}$ was known [74]. To the best of our knowledge, this is the first use of n-fold integer programing in the context of approximation algorithms.

RESULTS AND METHODOLOGY. To show the conceptual power of the MCIP, we utilize it for two more problems: The *splittable* and the *preemptive* setup model of machine scheduling. In both variants, for each job j, a setup time s_j is given. Each job may be partitioned into multiple parts that can be assigned to different machines, but before any part of the job can be processed the setup time has to be paid. In the splittable model, job parts belonging to the same job can be processed in parallel, and therefore it suffices to find a partition of the jobs and an assignment of the job parts to machines. This is not the case for the preemptive model, in which additionally a starting time for each job part has to be found, and two parts of the same job may not be processed in parallel. In 1999, Schuurman and Woeginger [133] presented a polynomial time algorithm for the preemptive model with approximation guarantee $4/3 + \varepsilon$, and for the splittable case, a guarantee of $5/3$ was achieved by Chen, Ye, and Zhang [29]. These are the best known approximation guarantees for the problems at hand. We show that solutions arbitrarily close to the optimum can be found in polynomial time:

THEOREM 6.1: There is an EPTAS for minimum makespan scheduling on identical parallel machines in the setup-class model, as well as in the preemptive and splittable setup models.

Note that all three problems are strongly NP-hard, due to trivial reductions from machine scheduling, and our results are therefore in some sense best-possible. We have the following running times:

- $2^{\mathcal{O}(1/\varepsilon^3 \log^4 1/\varepsilon)} Knm \log(n) \log^5(K)$ in the setup class model.
- $2^{\mathcal{O}(1/\varepsilon^2 \log^3 1/\varepsilon)} n^2 \log^2(m) \log^5(n)$ in the splittable model.
- $2^{2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}} nm \log(m) \log^5(n)$ in the preemptive model.

Summing up, the main achievement of this chapter is the development of the module configuration IP and its application in the design

of approximation schemes. Up to now, EPTAS or even PTAS results seemed out of reach for the considered problems, and for the preemptive model, we provide the first improvement in 20 years. The simplification techniques developed for the splittable and preemptive model in order to employ the MCIP are original and in the latter case quite sophisticated and therefore interesting by themselves. Furthermore, we expect the MCIP to be applicable to other packing and scheduling problems as well, in particular for variants of machine scheduling and bin packing with additional class dependent constraints. On a more conceptual level, we have presented a first demonstration of the potential of n-fold integer programming in the theory of approximation algorithms and hope to inspire further studies in this direction.

We conclude this paragraph with a more detailed overview of our results and their presentation. For all three EPTAS results, we employ the classical dual approximation framework by Hochbaum and Shmoys [65] to get a guess of the makespan T (see Section 2.2). In the following section, we develop the module configuration IP and argue that it is indeed an n-fold IP. The EPTAS results follow the same basic approach described above for machine scheduling: We find a schedule for a simplified instance via the MCIP and transform it into a schedule for the original one. The simplification steps typically include rounding of the processing and setup times using standard techniques, as well as the removal of certain jobs which later can be reinserted via carefully selected greedy procedures. For the splittable and preemptive model, we additionally have to prove that schedules with a certain simple structure exist, and in the preemptive model, the MCIP has to be extended. In Section 6.4 the basic versions of the EPTAS are presented, and in Section 6.5 some improvements of the running time for the splittable and the setup class model are discussed.

RELATED WORK. For an overview on n-fold IPs and their applications, we refer to the book by Onn [121]. The first FPT algorithm for n-fold IPs was presented by Hemmecke, Onn, and Romanchuk [63] in 2013, and it has a running time with a cubic dependence in n. In 2018, Eisenbrand, Hunkenschröder and Klein [42] and independently Koutecký, Levin and Onn [104] developed algorithms with running times with near quadratic dependence in n and improved dependencies in the parameters. Very recently, a near linear dependence in n was achieved by Jansen, Lassota and Rohwedder [76] as well as Eisenbrand et al. [44]. The former result also provides improved dependencies in the parameters. For an overview on recent results on n-fold IPs and related topics we refer to [44].

There have been recent applications of n-fold integer programming to scheduling problems in the context of parameterized algorithms: Knop and Koutecký [100] showed, among other things, that the problem of makespan minimization on unrelated parallel machines where

the processing times are dependent on both jobs and machines is fixed-parameter tractable with respect to the maximum processing time and the number of distinct machine types. This was generalized to the parameters maximum processing time and rank of the processing time matrix by Chen et al. [31]. Furthermore, Knop, Koutecký, and Mnich [101] provided an improved algorithm for a special type of n-fold IPs, yielding improved running times for several applications of n-fold IPs including results for scheduling problems.

There is extensive literature concerning scheduling problems with setup times. We highlight a few closely related results and otherwise refer to the surveys [2–4]. The setup class model was first considered by Mäcker et al. [112] in the special case that all classes have the same setup time. They designed a 2-approximation and additionally a $(3/2+\varepsilon)$-approximation for the case that the overall length of the jobs from each class is bounded. Jansen and Land [74] presented a simple 3-approximation with linear running time, a $(2+\varepsilon)$-approximation, and the aforementioned PTAS for the general setup class model. As indicated before, Chen et al. [29] developed a 5/3-approximation for the splittable model. A generalization of this, in which both setup and processing times are job and machine dependent, has been considered by Correa et al. [36]. They achieve a $(1+\phi)$-approximation where ϕ denotes the golden ratio, using a newly designed linear programming formulation. Moreover, there are recent results concerning machine scheduling in the splittable model considering the sum of (weighted) completion times as the objective function, e.g. [35, 131]. For the preemptive model, a PTAS for the special case that all jobs have the same setup time has been developed by Schuurman and Woeginger [133]. The mentioned $(4/3+\varepsilon)$-approximation for the general case [133] follows the same approach. Furthermore, a combination of the setup class and the preemptive model has been considered in which the jobs are scheduled preemptively, but the setup times are class dependent. Monma and Potts [117] presented, among other things, a $(2-1/(\lfloor m/2\rfloor+1))$-approximation for this model, and later Chen [27] achieved improvements for some special cases. Recently, Deppert and Jansen [39] presented 1.5-approximations with near-linear running times for this problem, the splittable, and the non-preemptive setup class model.

6.2 PRELIMINARIES

In the following, we establish some concepts and notations, formally define the considered problems, and discuss n-fold integer programs.

PROBLEMS. For all three of the considered problems, a set $\mathcal{J}$ of n jobs with processing times $p_j \in \mathbb{Q}_{>0}$ for each job $j \in \mathcal{J}$ and a number of machines m is given. In the preemptive and the splittable model,

the input additionally includes a setup time $s_j \in \mathbb{Q}_{>0}$ for each job $j \in \mathcal{J}$; while in the setup class model, it includes a number K of setup classes, a setup class $k_j \in [K]$ for each job $j \in \mathcal{J}$, as well as setup times $s_k \in \mathbb{Q}_{>0}$ for each $k \in [K]$.

We take a closer look at the definition of a schedule in the preemptive model. The jobs may be split. Therefore, partition sizes $\kappa : \mathcal{J} \to \mathbb{Z}_{>0}$, together with processing time fractions $\lambda_j : [\kappa(j)] \to (0, 1]$ such that $\sum_{k \in [\kappa(j)]} \lambda_j(k) = 1$ have to be found, meaning that job j is split into $\kappa(j)$ many parts and the k-th part for $k \in [\kappa(j)]$ has processing time $\lambda_j(k)p_j$. This given, we define $\mathcal{J}' = \{(j,k) \mid j \in \mathcal{J}, k \in [\kappa(j)]\}$ to be the set of job parts. Now, an assignment $\sigma : \mathcal{J}' \to [m]$ along with starting times $\xi : \mathcal{J}' \to \mathbb{Q}_{>0}$ has to be determined such that any two job parts assigned to the same machine or belonging to the same job do not overlap. More precisely, we have to assure that for each two job parts $(j,k), (j',k') \in \mathcal{J}'$ with $\sigma(j,k) = \sigma(j',k')$ or $j = j'$, we have $\xi(j,k) + s_j + \lambda_j(k)p_j \leqslant \xi(j',k')$ or vice versa. A schedule is given by $(\kappa, \lambda, \sigma, \xi)$, and the makespan can be defined as $C_{\max} = \max_{(j,k) \in \mathcal{J}'}(\xi(j,k) + s_j + \lambda_j(k)p_j)$. Note that the variant of the problem in which overlap between a job part and setup of the same job is allowed is equivalent to the one presented above. This was pointed out by Schuurmann and Woeginger [133] and can be seen with a simple swapping argument.

In the splittable model, it is not necessary to determine starting times for the job parts because, given the assignment σ, the job parts assigned to each machine can be scheduled as soon as possible in arbitrary order without gaps. Hence, in this case, the output is of the form $(\kappa, \lambda, \sigma)$, and the makespan can be defined as $C_{\max} = \max_{i \in [m]} \sum_{(j,k) \in \sigma^{-1}(i)} (s_j + \lambda_j(k)p_j)$.

Lastly, in the setup class model, the jobs are not split, and the jobs assigned to each machine can be scheduled in batches comprised of the jobs of the same class assigned to the machine without overlaps and gaps. The output is therefore just an assignment $\sigma : \mathcal{J} \to [m]$, and the makespan is given by $C_{\max} = \max_{i \in [m]} \sum_{j \in \sigma^{-1}(i)} p_j + \sum_{k \in \{k_j \mid j \in \sigma^{-1}(i)\}} s_k$.

Note that in the preemptive and the setup class model, we can assume that the number of machines is bounded by the number of jobs: If there are more machines than jobs, placing each job on a private machine yields an optimal schedule in both models, and the remaining machines can be ignored. This, however, is not the case in the splittable model, which causes a minor problem in the following.

n-FOLD INTEGER PROGRAMS. We briefly define n-fold integer programs (IPs) following the notation of [63] and [100], and state the main algorithmic result needed in the following. Let $n, r, s, t \in \mathbb{Z}_{>0}$

be integers and A be an integer $((r+ns)\times nt)$-matrix of the following form:

$$A = \begin{pmatrix} A_1 & A_1 & \cdots & A_1 \\ A_2 & 0 & \cdots & 0 \\ 0 & A_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_2 \end{pmatrix}$$

The matrix A is the so-called n-fold product of the bimatrix $\binom{A_1}{A_2}$, with A_1 an $r \times t$ and A_2 an $s \times t$ matrix. Furthermore, let $w, \ell, u \in \mathbb{Z}^{nt}$ and $b \in \mathbb{Z}^{r+ns}$. Then the n-fold integer programming problem is given by:

$$\min\{wx \mid Ax = b, \ell \leqslant x \leqslant u, x \in \mathbb{Z}^{nt}\}$$

We set Δ to be the maximum absolute value occurring in A. There are several algorithms for solving n-fold IPs. We use the recent result by Jansen, Lassota and Rohwedder [76]:

THEOREM 6.2: Let φ be the encoding length of the largest number occurring in the input. The n-fold integer programming problem can be solved in time $(rs\Delta)^{\mathcal{O}(r^2s+s^2)}\varphi^2 nt \log^5(nt)$.

The variables x can naturally be partitioned into *bricks* $x^{(q)}$ of dimension t for each $q \in [n]$ such that $x = (x^{(1)}, \dots x^{(n)})$. Furthermore, we denote the constraints corresponding to A_1 as *globally uniform* and the ones corresponding to A_2 as *locally uniform*. Hence, r is the number of globally and s the number of locally uniform constraints (ignoring their n-fold duplication), t the *brick size* and n the *brick number*.

6.3 MODULE CONFIGURATION IP

In this section, we state the configuration IP for machine scheduling; introduce a basic version of the module configuration IP (MCIP) that is already sufficiently general to work for both the splittable and setup class model; and lastly show that the configuration IP can be expressed by the MCIP in multiple ways. We will use the dual approximation framework by Hochbaum and Shmoys [65] and hence always assume that a guess T of the makespan is given (see Section 2.2). Before that, however, we formally introduce the concept of *configurations*.

Given a set of objects A, such as jobs, a configuration C of these objects is a vector of multiplicities indexed by the objects, i.e., $C \in \mathbb{Z}^A_{\geqslant 0}$. For given sizes $\Lambda(a)$ of the objects $a \in A$, the size $\Lambda(C)$ of a configuration C is defined as $\sum_{a\in A} C_a \Lambda(a)$. Moreover, for a given upper bound B, we define $\mathcal{C}_A(B)$ to be the set of configurations of A that are bounded in size by B, that is, $\mathcal{C}_A(B) = \{C \in \mathbb{Z}^A_{\geqslant 0} \mid \Lambda(C) \leqslant B\}$.

CONFIGURATION IP. We provide a recollection of the configuration IP for machine scheduling. Let P be the set of distinct processing times for some instance I with multiplicities n_p for each $p \in P$, meaning, I includes exactly n_p jobs with processing time p. The size $\Lambda(p)$ of a processing time p is given by itself. Furthermore, let T be a guess of the optimal makespan. The configuration IP for I and T is given by variables $x_C \geqslant 0$ for each $C \in \mathcal{C}_P(T)$ and the following constraints:

$$\sum_{C \in \mathcal{C}_P(T)} x_C = m \tag{6.1}$$

$$\sum_{C \in \mathcal{C}_P(T)} C_p x_C = n_p \qquad \forall p \in P \tag{6.2}$$

Due to constraint (6.1), exactly one configuration is chosen for each machine; while (6.2) ensures that the correct number of jobs or job sizes is covered.

MODULE CONFIGURATION IP. Let $\mathcal{B}$ be a set of basic objects (e.g. jobs or setup classes) and let there be D integer values $B_1, \ldots, B_D$ for each basic object $B \in \mathcal{B}$ (e.g. processing time or numbers of different kinds of jobs). Our approach is to cover the basic objects with so-called *modules* and, in turn, cover the modules with configurations. Depending on the context, modules correspond to batches of jobs or job piece sizes together with a setup time and can also encompass additional information like a starting time. Let $\mathcal{M}$ be a set of such modules. In order to cover the basic objects, each module $M \in \mathcal{M}$ also has D integer values $M_1, \ldots, M_D$. Furthermore, each module M has a size $\Lambda(M)$ and a set of eligible basic objects $\mathcal{B}(M)$. The latter is needed because not all modules are compatible with all basic objects, e.g., because they do not have the right setup times. The configurations are used to cover the modules, however, it typically does not matter which module exactly is covered, but rather which size the module has. Let H be the set of distinct module sizes, i.e., $H = \{\Lambda(M) \mid M \in \mathcal{M}\}$, and for each module size $h \in H$ let $\mathcal{M}(h)$ be the set of modules with size h. We consider the set $\mathcal{C}$ of configurations of module sizes which are bounded in size by a guess of the makespan T, i.e., $\mathcal{C} = \mathcal{C}_H(T)$. In the preemptive case, configurations need to additionally encompass information about starting times of modules, and therefore the definition of configurations will be slightly more complicated in that case.

Since we want to choose configurations for each machine, we have variables x_C for each $C \in \mathcal{C}$ and constraints corresponding to (6.1). Furthermore, we choose modules with variables y_M for each $M \in \mathcal{M}$, and because we want to cover the chosen modules with configurations, we have some analogue of constraint (6.2), say $\sum_{C \in \mathcal{C}(T)} C_h x_C = \sum_{M \in \mathcal{M}(h)} y_M$ for each module size $h \in H$. It turns out, however, that to properly cover the basic objects with modules, we need the variables

y_M for each basic object, and this is where n-fold IPs come into play. The variables stated so far form a brick of the variables of the n-fold IP, and there is one brick for each basic object, that is, we have, for each $B \in \mathcal{B}$, variables $x_C^{(B)}$ for each $C \in \mathcal{C}$, and $y_M^{(B)}$ for each $M \in \mathcal{M}$. Using the upper bounds of the n-fold model, variables $y_M^{(B)}$ are set to zero, if B is not eligible for M; and we set the lower bounds of all variables to zero. Sensible upper bounds for the remaining variables will be typically clear from context. Besides that, the module configuration integer program MCIP (for $\mathcal{B}$, $\mathcal{M}$ and $\mathcal{C}$) is given by:

$$\sum_{B\in\mathcal{B}} \sum_{C\in\mathcal{C}} x_C^{(B)} = m \tag{6.3}$$

$$\sum_{B\in\mathcal{B}} \sum_{C\in\mathcal{C}(T)} C_h x_C^{(B)} = \sum_{B\in\mathcal{B}} \sum_{M\in\mathcal{M}(h)} y_M^{(B)} \qquad \forall h \in H \tag{6.4}$$

$$\sum_{M\in\mathcal{M}} M_d y_M^{(B)} = B_d \qquad \forall B \in \mathcal{B}, d \in [D] \tag{6.5}$$

It is easy to see that the constraints (6.3) and (6.4) are globally uniform. They are the mentioned adaptations of (6.1) and (6.2). The constraint (6.5), on the other hand, is locally uniform and ensures that the basic objects are covered.

Note that, while the duplication of the configuration variables does not carry meaning, it also does not upset the model: Consider the modified MCIP that is given by not duplicating the configuration variables. A solution $(\tilde{x}, \tilde{y})$ for this IP gives a solution (x, y) for the MCIP by fixing some basic object B^*, setting $x_C^{(B^*)} = \tilde{x}_C$ for each configuration C, setting the remaining configuration variables to 0, and copying the remaining variables. Given a solution (x, y) for the MCIP, on the other hand, gives a solution for the modified version $(\tilde{x}, \tilde{y})$ by setting $\tilde{x}_C = \sum_{B\in\mathcal{B}} x_C^B$ for each configuration C. Summarizing we get:

Observation 6.3. The MCIP is an n-fold IP with brick-size $t = |\mathcal{M}| + |\mathcal{C}|$, brick number $n = |\mathcal{B}|$, $r = |H| + 1$ globally uniform and $s = D$ locally uniform constraints.

Moreover, in all of the considered applications, we will minimize the overall size of the configurations, i.e., $\sum_{B\in\mathcal{B}} \sum_{C\in\mathcal{C}} \Lambda(C) x_C^{(B)}$. This will be required because in the simplification steps of our algorithms some jobs are removed and have to be reinserted later, and we therefore have to make sure that no space is wasted.

FIRST EXAMPLE. We conclude the section by pointing out several different ways to replace the classical configuration IP for machine scheduling with the MCIP, thereby giving some intuition for the model. The first possibility is to consider the jobs as the basic objects and their processing times as their single value ($\mathcal{B} = \mathcal{J}$, $D = 1$); the modules are the processing times ($\mathcal{M} = P$), and a job is eligible for a module, if its processing time matches; and the configurations are all

the configurations bounded in size by T. Another option is to choose the processing times as basic objects keeping all the other definitions essentially like before. Lastly, we could consider the whole set of jobs or the whole set of processing times as a single basic object with $D = |P|$ different values. In this case, we can define the set of modules as the set of configurations of processing times bounded by T.

6.4 EPTAS RESULTS

In this section, we present approximation schemes for each of the three considered problems. Like before, we use the dual approximation framework by Hochbaum and Shmoys [65] and hence assume for each problem that a guess of the makespan T is given (see Section 2.2). Note that for each of the problems a simple 1.5-approximation with near linear running time is known [39] and hence this has no influence on the claimed asymptotic running times. Each of the results follows the same approach: The instance is carefully simplified, a schedule for the simplified instance is found using the MCIP, and this schedule is transformed into a schedule for the original instance. The presentation of the result is also similar for each problem: We first discuss how the instance can be sensibly simplified and how a schedule for the simplified instance can be transformed into a schedule for the original one. Next, we discuss how a schedule for the simplified instance can be found using the MCIP, and, lastly, we summarize and analyze the taken steps.

For the sake of clarity, we have given rather formal definitions for the problems at hand in Section 6.2. In the following, however, we will use the terms in a more intuitive fashion for the most part, and we will, for instance, often take a geometric rather than a temporal view on schedules and talk about the *length* or the *space* taken up by jobs and setups on machines rather than time. In particular, given a schedule for an instance of any one of the three problems together with an upper bound for the makespan T, the *free space* with respect to T on a machine is defined as the summed up lengths of time intervals between 0 and T in which the machine is idle. The free space (with respect to T) is the summed up free space of all the machines. For bounds T and L for the makespan and the free space, respectively, we say that a schedule is a (T, L)-schedule if its makespan is at most T and the free space with respect to T is at least L.

When transforming the instance, we will increase or decrease processing and setup times and fill in or remove extra jobs. Consider a (T', L')-schedule where T' and L' denote some arbitrary makespan or free space bounds. If we fill in extra jobs or increase processing or setup times, but can bound the increase on each machine by some bound b, we end up with a $(T' + b, L')$-schedule for the transformed instance. In particular, we have the same bound for the free space

because we properly increased the makespan bound. If, on the other hand, jobs are removed or setup times decreased, we obviously still have a (T', L')-schedule for the transformed instance. This will be used frequently in the following.

Setup Class Model

We start with the setup class model. In this case, we can essentially reuse the simplification steps that were developed by Jansen and Land [74] for their PTAS. The main difference between the two procedures is that we solve the simplified instance via the MCIP, while they used a dynamic program. For the sake of self-containment, we include our own simplification steps, but remark that they are strongly inspired by those from [74]. In Section 6.5, we present a more elaborate rounding procedure resulting in an improved running time.

SIMPLIFICATION OF THE INSTANCE. In the following, we distinguish *big setup* jobs j belonging to classes k with setup times $s_k \geqslant \varepsilon^3 T$ and *small setup* jobs with $s_k < \varepsilon^3 T$. We denote the corresponding subsets of jobs by $\mathcal{J}^{\text{bst}}$ and $\mathcal{J}^{\text{sst}}$, respectively. Furthermore, we call a job *tiny* or *small*, if its processing time is smaller than $\varepsilon^4 T$ or εT, respectively, and *big* or *large* otherwise. For any given set of jobs J, we denote the subset of tiny jobs from J with J_{tiny} and the small, big, and large jobs analogously (see Table 6.1 for an overview). We simplify the instance in four steps, aiming for an instance that exclusively includes big jobs with big setup times and additionally only a constant number of distinct processing and setup times. For technical reasons, we assume $\varepsilon \leqslant 1/2$.

We proceed with the first simplification step. Let I_1 be the instance given by the job set $\mathcal{J} \setminus \mathcal{J}^{\text{sst}}_{\text{small}}$ and Q the set of setup classes completely contained in $\mathcal{J}^{\text{sst}}_{\text{small}}$, i.e., $Q = \{k \mid \forall j \in \mathcal{J} : k_j = k \Rightarrow j \in \mathcal{J}^{\text{sst}}_{\text{small}}\}$. An obvious lower bound on the space taken up by the jobs from $\mathcal{J}^{\text{sst}}_{\text{small}}$ in any schedule is given by $L = \sum_{j \in \mathcal{J}^{\text{sst}}_{\text{small}}} p_j + \sum_{k \in Q} s_k$. Note that the instance I_1 may include a reduced number K' of setup classes.

LEMMA 6.4. *A schedule for* I *with makespan* T *induces a* (T, L)*-schedule for* I_1*, that is, a schedule with makespan* T *and free space at least* L*; and*

s_{k_j} \ p_j	$< \varepsilon^4 T$	$< \varepsilon T$	$\geqslant \varepsilon T$
$< \varepsilon^3 T$	$\mathcal{J}^{\text{sst}}_{\text{tiny}}$	$\mathcal{J}^{\text{sst}}_{\text{small}}$	$\mathcal{J}^{\text{sst}}_{\text{large}}$
$\geqslant \varepsilon^3 T$	$\mathcal{J}^{\text{bst}}_{\text{tiny}}$	$\mathcal{J}^{\text{bst}}_{\text{small}}$	$\mathcal{J}^{\text{bst}}_{\text{large}}$

Table 6.1: Overview on the job classifications

any (T', L)*-schedule for* I_1 *can be transformed into a schedule for* I *with makespan at most* $(1+\varepsilon)T' + \varepsilon T + 2\varepsilon^3 T$.

Proof. The first claim is obvious and we therefore assume that we have a (T', L)-schedule for I_1. We group the jobs from $\mathcal{J}^{sst}_{small}$ by setup classes and first consider the groups with summed up processing time at most $\varepsilon^2 T$. For each of these groups, we check whether the respective setup class contains a large job. If this is the case, we schedule the complete group on a machine on which such a large job is already scheduled if possible using up free space. Since the large jobs have a length of at least εT, there are at most $T'/(\varepsilon T)$ many large jobs on each machine, and therefore the schedule on the respective machine has length at most $(1+\varepsilon)T'$, or there is free space with respect to T' left. If, on the other hand, the respective class does not contain a large job and is therefore fully contained in $\mathcal{J}^{sst}_{small'}$ we create a container including the whole class and its setup time. Note that the overall length of the container is at most $(\varepsilon^2 + \varepsilon^3)T \leqslant \varepsilon T$ (using $\varepsilon \leqslant 1/2$). Next, we create a sequence containing the containers and the remaining jobs ordered by setup class. We insert the items from this sequence greedily into the remaining free space in a next-fit fashion exceeding T' on each machine by at most one item from the sequence, thereby creating an error of at most εT. This can be done because we had a free space of at least L, and the inserted objects had an overall length of at most L. To make the resulting schedule feasible, we have to insert some setup times. However, because the overall length of the jobs from each class in need of a setup is at least $\varepsilon^2 T$, and the sequence was ordered by classes, there are at most $T'/(\varepsilon^2 T) + 2$ distinct classes without a setup time on each machine. Inserting the missing setup times will therefore increase the makespan by at most $(T'/(\varepsilon^2 T) + 2)\varepsilon^3 T = \varepsilon T' + 2\varepsilon^3 T$. □

Next, we deal with the remaining (large) jobs with small setup times $j \in \mathcal{J}^{sst}_{large}$. Let I_2 be the instance we get by increasing the setup times of the classes with small setup times to $\varepsilon^3 T$. We denote the setup time of class $k \in [K']$ for I_2 by s'_k. Note that there are no small setup jobs in I_2.

LEMMA 6.5. *A* (T', L')*-schedule* I_1 *induces a* $((1+\varepsilon^2)T', L')$*-schedule for* I_2*, and a* (T', L')*-schedule for* I_2 *is also a* (T', L')*-schedule for* I_1.

Proof. The first claim is true because in a schedule with makespan at most T' there can be at most $T'/(\varepsilon T)$ many large jobs on any machine, and the second claim is obvious. □

Let I_3 be the instance we get by replacing the jobs from $\mathcal{J}^{bst}_{tiny}$ with placeholders of size $\varepsilon^4 T$. More precisely, we remove $\mathcal{J}^{bst}_{tiny}$, and, for each class $k \in [K]$, we introduce $\lceil (\sum_{j \in \mathcal{J}^{bst}_{tiny}, k_j = k} p_j)/(\varepsilon^4 T) \rceil$ many jobs with processing time $\varepsilon^4 T$ and class k. We denote the job set of I_3 by $\mathcal{J}'$

and the processing time of a job $j \in \mathcal{J}'$ by p'_j. Note that I_3 exclusively contains big jobs with big setup times.

LEMMA 6.6. *If there is a* (T', L')*-schedule for* I_2*, there is also a* $((1+\varepsilon)T', L')$*-schedule; and if there is a* (T', L')*-schedule for* I_3*, there is also a* $((1+\varepsilon)T', L')$*-schedule for* I_2*.*

Proof. Note that for any (T', L')-schedule for I_2 or I_3, there are at most $T'/(\varepsilon^3 T)$ many distinct big setup classes scheduled on any machine. Hence, when considering such a schedule for I_2, we can remove the tiny jobs belonging to $\mathcal{J}^{\text{bst}}_{\text{tiny}}$ from the machines and instead fill in the placeholders, such that each machine for each class receives at most as much length from that class, as was removed, rounded up to the next multiple of $\varepsilon^4 T$. All placeholders can be placed like this and the makespan is increased by at most $(T'/(\varepsilon^3 T))\varepsilon^4 T = \varepsilon T'$. If, on the other hand, we consider such a schedule for I_3, we can remove the placeholders and instead fill in the respective tiny jobs, again overfilling by at most one job. This yields a $((1+\varepsilon)T', L')$-schedule for I_2 with the same argument. □

Lastly, we perform both a geometric and an arithmetic rounding step for the processing and setup times. The geometric rounding is needed to suitably bound the number of distinct processing and setup times, and due to the arithmetic rounding, we will be able to guarantee integral coefficients in the IP. More precisely, we set $\tilde{p}_j = (1+\varepsilon)^{\lceil \log_{1+\varepsilon} p'_j/(\varepsilon^4 T)\rceil}\varepsilon^4 T$ and $\bar{p}_j = \lceil \tilde{p}_j/\varepsilon^5 T\rceil \varepsilon^5 T$ for each $j \in \mathcal{J}'$, as well as $\tilde{s}_j = (1+\varepsilon)^{\lceil \log_{1+\varepsilon} s'_j/(\varepsilon^3 T)\rceil}\varepsilon^3 T$ and $\bar{s}_k = \lceil \tilde{s}_j/\varepsilon^5 T\rceil \varepsilon^5 T$ for each setup class $k \in [K']$. The resulting instance is called I_4.

LEMMA 6.7. *A* (T', L')*-schedule for* I_3 *induces a* $((1+3\varepsilon)T', L')$*-schedule for* I_4*, and any* (T', L')*-schedule for* I_4 *can be turned into a* (T', L')*-schedule for* I_3*.*

Proof. For the first claim, we first stretch a given schedule by $(1+\varepsilon)$. This enables us to use the processing and setup times due to the geometric rounding step. Now, using the ones due to the second step increases the schedule by at most $2\varepsilon T'$, because there where at most $T'/(\varepsilon^4 T)$ many big jobs on any machine to begin with. The second claim is obvious. □

Based on the rounding steps, we define two makespan bounds $\bar{T}$ and $\check{T}$: Let $\bar{T}$ be the makespan bound that is obtained from T by the application of the Lemmata 6.4-6.7 in sequence, i.e., $\bar{T} = (1+\varepsilon^2)(1+\varepsilon)(1+3\varepsilon)T = (1+\mathcal{O}(\varepsilon))T$. We will find a $(\bar{T}, L)$-schedule for I_4 utilizing the MCIP and afterward apply the Lemmata 6.4-6.7 backwards to get a schedule with makespan $\check{T} = (1+\varepsilon)^2\bar{T} + \varepsilon T + 2\varepsilon^3 T = (1+\mathcal{O}(\varepsilon))T$.

Let P and S be the sets of distinct occurring processing and setup times for instance I_4. Because of the rounding, the minimum and

maximum lengths of the setup and processing times, and $\varepsilon < 1$, we can bound $|P|$ and $|S|$ by $\mathcal{O}(\log_{1+\varepsilon} 1/\varepsilon) = \mathcal{O}(1/\varepsilon \log 1/\varepsilon)$.

UTILIZATION OF THE MCIP. At this point, we can employ the module configuration IP. The basic objects in this context are the setup classes, i.e., $\mathcal{B} = [K']$, and the different values are the numbers of jobs with a certain processing time, i.e., $D = |P|$. We set $n_{k,p}$ to be the number of jobs from setup class $k \in [K']$ with processing time $p \in P$. The modules correspond to batches of jobs together with a setup time. Batches of jobs can be modeled as configurations of processing times, that is, multiplicity vectors indexed by the processing times. Hence, we define the set of modules $\mathcal{M}$ to be the set of pairs of configurations of processing times and setup times with a summed up size bounded by $\bar{T}$, i.e., $\mathcal{M} = \{(C, s) \mid C \in \mathcal{C}_P(\bar{T}), s \in S, s + \Lambda(C) \leqslant \bar{T}\}$, and write $M_p = C_p$ and $s_M = s$ for each module $M = (C, s) \in \mathcal{M}$. The values of a module M are given by the numbers M_p and its size $\Lambda(M)$ by $s_M + \sum_{p \in P} M_p p$. Remember that the configurations $\mathcal{C}$ are the configurations of module sizes H that are bounded in size by $\bar{T}$, i.e., $\mathcal{C} = \mathcal{C}_H(\bar{T})$. A setup class is eligible for a module if the setup times fit, i.e., $\mathcal{B}_M = \{k \in [K'] \mid s_k = s_M\}$. Lastly, we establish $\varepsilon^5 T = 1$ by scaling.

For the sake of readability, we state the resulting constraints of the MCIP with adapted notation and without duplication of the configuration variables:

$$\sum_{C \in \mathcal{C}} x_C = m \tag{6.6}$$

$$\sum_{C \in \mathcal{C}} C_h x_C = \sum_{k \in [K']} \sum_{M \in \mathcal{M}(h)} y_M^{(k)} \qquad \forall h \in H \tag{6.7}$$

$$\sum_{M \in \mathcal{M}} M_p y_M^{(k)} = n_{k,p} \qquad \forall k \in [K'], p \in P \tag{6.8}$$

Note that the coefficients are all integral and this includes those of the objective function, i.e., $\sum_C \Lambda(C) x_C$, because of the scaling step.

LEMMA 6.8. *With the above definitions, there is a* $(\bar{T}, L)$*-schedule for* I_4 *if and only if the* MCIP *has a solution with objective value at most* $m\bar{T} - L$.

Proof. Let there be a $(\bar{T}, L)$-schedule for I_4. Then the schedule on a given machine corresponds to a distinct configuration C that can be determined by counting for each possible module size h the batches of jobs from the same class whose length together with the setup time adds up to an overall length of h. Note that the length of this configuration is equal to the used up space on that machine. We fix an arbitrary setup class k and set the variables $x_C^{(k)}$ accordingly (and $x_C^{(k')} = 0$ for $k' \neq k$ and $C \in \mathcal{C}$). By this setting, we get an objective value of at most $m\bar{T} - L$ because there was at least L free space in the schedule. For each class k and module M, we count the number of machines on which there are exactly M_p jobs with processing time p

from class k for each $p \in P$ and set $y_M^{(k)}$ accordingly. It is easy to see that the constraints are satisfied by these definitions.

Given a solution (x, y) of the MCIP, we define a corresponding schedule: Because of (6.6), we can match the machines to configurations such that each machine is matched to exactly one configuration. If machine i is matched to C, for each module size h, we create C_h slots of length h on i. Next, we divide the setup classes into batches. For each class k and module M, we create $y_M^{(k)}$ batches of jobs from class k with M_p jobs with processing time p for each $p \in P$ and place the batch together with the corresponding setup time into a fitting slot on some machine. Because of (6.8) and (6.7), all jobs can be placed by this process. Note that the used space equals the overall size of the configurations, and we therefore have free space of at least L. □

RESULT. Using the above results, we can formulate and analyze the following procedure:

ALGORITHM 6.9.

1. Generate the modified instance I_4:
   - Remove the small jobs with small setup times.
   - Increase the setup times of the remaining classes with small setup times.
   - Replace the tiny jobs with big setup times.
   - Round up the resulting processing and setup times.
2. Build and solve the MCIP for I_4.
3. If the MCIP is infeasible, or the objective value greater than $m\bar{T} - L$, report that I has no solution with makespan T.
4. Otherwise build the schedule with makespan $\bar{T}$ and free space at least L for I_4.
5. Transform the schedule into a schedule for I with makespan at most $\breve{T}$:
   - Use the prerounding processing and setup times.
   - Replace the placeholders by the tiny jobs with big setup times.
   - Use the orignal setup times of the classes with small setup times.
   - Insert the small jobs with small setup times into the free space.

The procedure is correct due to the above results. To analyze its running time, we first bound the parameters of the MCIP. We have $|\mathcal{B}| = K' \leqslant K$ and $D = |P|$ by definition and $|\mathcal{M}| = \mathcal{O}(|S|(1/\varepsilon^3)^{|P|}) =$

$2^{\mathcal{O}(1/\varepsilon \log^2 1/\varepsilon)}$ because $|S|, |P| \in \mathcal{O}(1/\varepsilon \log 1/\varepsilon)$. This is true due to the last rounding step which also implies $|H| = \mathcal{O}(1/\varepsilon^5)$ yielding $|\mathcal{C}| = |H|^{\mathcal{O}(1/\varepsilon^3)} = 2^{\mathcal{O}(1/\varepsilon^3 \log 1/\varepsilon)}$. According to Observation 6.3, this yields a brick size of $t = 2^{\mathcal{O}(1/\varepsilon^3 \log 1/\varepsilon)}$, a brick number of K, $r = \mathcal{O}(1/\varepsilon^5)$ globally, and $s = \mathcal{O}(1/\varepsilon \log 1/\varepsilon)$ locally uniform constraints for the MCIP. We have $\Delta = \mathcal{O}(1/\varepsilon^5)$ because all occurring values in the processing time matrix are bounded in $\bar{T}$, and we have $\bar{T} = \mathcal{O}(1/\varepsilon^5)$ due to the scaling. Furthermore, the values of the objective function, the right hand side, and the upper and lower bounds on the variables are bounded by $\mathcal{O}(n/\varepsilon^5)$ yielding a bound of $\mathcal{O}(\log(n/\varepsilon^5))$ for the encoding length of the biggest number in the input φ.

By Theorem 6.2 and some arithmetic, the MCIP can be solved in time:

$$(rs\Delta)^{\mathcal{O}(r^2 s + s^2)} \varphi^2 Kt \log^5(Kt) = 2^{\mathcal{O}(1/\varepsilon^{11} \log^2 1/\varepsilon)} K \log^2(n) \log^5(K)$$

When building the actual schedule, we iterate through the jobs and machines like indicated in the proof of Lemma 6.8 yielding the following:

THEOREM 6.10: The algorithm for the setup class model finds a schedule with makespan $(1 + \mathcal{O}(\varepsilon))T$ or correctly determines that there is no schedule with makespan T in time:

$$2^{\mathcal{O}(1/\varepsilon^{11} \log^2 1/\varepsilon)} Knm \log(n) \log^5(K)$$

Splittable Model

The approximation scheme for the splittable model presented in this section is probably the easiest one discussed in this chapter. There is, however, one problem concerning this procedure: Its running time is polynomial in the number of machines which might be exponential in the input size. In Section 6.5, we show how this problem can be overcome and further improve the running time.

SIMPLIFICATION OF THE INSTANCE. In this context, the set of big setup jobs $\mathcal{J}^{bst}$ is given by the jobs with setup times at least εT and the small setup jobs $\mathcal{J}^{sst}$ are all the others. Let $L = \sum_{j \in \mathcal{J}^{sst}} (s_j + p_j)$. Because every job has to be scheduled and every setup has to be paid at least once, L is a lower bound on the summed up space due to small jobs in any schedule. Let I_1 be the instance that we get by removing all the small setup jobs from the given instance I.

LEMMA 6.11. *A schedule with makespan T for I induces a (T, L)-schedule for I_1; and any (T', L)-schedule for I_1 can be transformed into a schedule for I with makespan at most $T' + \varepsilon T$.*

Proof. The first claim is obvious. Hence, consider a sequence consisting of the jobs from $\mathcal{J}^{sst}$ together with their setup times where the setup

time of a job is the direct predecessor of the job. We insert the setup times and jobs from this sequence greedily into the schedule in a next-fit fashion: Given a machine, we keep inserting the items from the sequence on to the machine at the end of the schedule until the taken up space reaches T'. If the current item does not fit exactly, we cut it, such that the used space on the machine is exactly T'. Then we continue with the next machine. We can place the whole sequence like this without exceeding the makespan T', because we have free space of at least L which is the summed up length of the items in the sequence. Next, we remove each setup time that was placed only partly on a machine together with those that were placed at the end of the schedule Furthermore, we insert a fitting setup time for the jobs that were scheduled without one, which can happen only once for each machine. This yields a feasible schedule whose makespan is increased by at most εT. □

Next, we round up the processing and setup times of I_1 to the next multiple of $\varepsilon^2 T$, that is, for each job $j \in \mathcal{J}$, we set $\bar{p}_j = \lceil p_j/(\varepsilon^2 T)\rceil \varepsilon^2 T$ and $\bar{s}_j = \lceil s_j/(\varepsilon^2 T)\rceil \varepsilon^2 T$. We call the resulting instance I_2 and denote its job set by $\mathcal{J}'$.

LEMMA 6.12. *If there is a (T, L')-schedule for I_1, then there is also a $((1+2\varepsilon)T, L')$-schedule for I_2 in which the length of each job part is a multiple of $\varepsilon^2 T$; and any (T', L')-schedule for I_2 yields a (T', L')-schedule for I_1.*

Proof. Consider a (T, L)-schedule for I_1. There are at most $1/\varepsilon$ jobs scheduled on each machine since each setup time has a length of at least εT. On each machine, we extend each occurring setup time and the processing time of each occurring job part by at most $\varepsilon^2 T$ to round it to a multiple of $\varepsilon^2 T$. This step extends the makespan by at most $2\varepsilon T$. Since now each job part is a multiple of $\varepsilon^2 T$, the total processing time of the job is a multiple of $\varepsilon^2 T$ too. However, its overall length might be greater than its rounded processing time, and we simply discard some processing time in this case. The second claim is obvious. □

Based on the two Lemmata, we define two makespan bounds $\bar{T} = (1+2\varepsilon)T$ and $\check{T} = \bar{T} + \varepsilon T = (1+3\varepsilon)T$. We will use the MCIP to find a $(\bar{T}, L)$-schedule for I_2 in which the length of each job part is a multiple of $\varepsilon^2 T$. Using the two Lemmata, this will yield a schedule with makespan at most $\check{T}$ for the original instance I.

UTILIZATION OF THE MCIP. The basic objects, in this context, are the (big setup) jobs, i.e., $\mathcal{B} = \mathcal{J}^{\text{bst}} = \mathcal{J}'$, and they have only one value ($D = 1$), namely, their processing time. Moreover, the modules are defined as the set of pairs of job piece sizes and setup times, i.e., $\mathcal{M} = \left\{(q, s) \,\middle|\, s, q \in \{x\varepsilon^2 T \mid x \in \mathbb{Z}, 0 < x \leqslant 1/\varepsilon^2\}, s \geqslant \varepsilon T\right\}$, and we write $s_M = s$ and $q_M = q$ for each module $M = (q, s) \in \mathcal{M}$. Corresponding

to the value of the basic objects, the value of a module M is q_M, and its size $\Lambda(M)$ is given by $q_M + s_M$. A job is eligible for a module if the setup times fit, i.e., $\mathcal{B}_M = \{j \in \mathcal{J}' \mid s_j = s_M\}$. In order to ensure integral values, we establish $\varepsilon^2 T = 1$ via a simple scaling step. The set of configurations $\mathcal{C}$ is comprised of all configurations of module sizes H that are bounded in size by $\bar{T}$, i.e., $\mathcal{C} = \mathcal{C}_H(\bar{T})$. We state the constraints of the MCIP for the above definitions with adapted notation and without duplication of the configuration variables:

$$\sum_{C \in \mathcal{C}} x_C = m \tag{6.9}$$

$$\sum_{C \in \mathcal{C}} C_h x_C = \sum_{j \in \mathcal{J}'} \sum_{M \in \mathcal{M}(h)} y_M^{(j)} \qquad \forall h \in H \tag{6.10}$$

$$\sum_{M \in \mathcal{M}} q_M y_M^{(j)} = p_j \qquad \forall j \in \mathcal{J}' \tag{6.11}$$

Note that we additionally minimize the summed up size of the configurations via the objective function $\sum_C \Lambda(C) x_C$.

LEMMA 6.13. *With the above definitions, there is a $(\bar{T}, L)$-schedule for I_2 in which the length of each job piece is a multiple of $\varepsilon^2 T$ if and only if the* MCIP *has a solution with objective value at most $m\bar{T} - L$.*

Proof. Given such a schedule for I_2, the schedule on each machine corresponds to exactly one configuration C that can be derived by counting the job pieces and setup times with the same summed up length h and setting C_h accordingly. This yields the values for the x variables. The size of the configuration C is equal to the used space on the respective machine. Hence, the objective value is bounded by $m\bar{T} - L$. Furthermore, for each job j and job part length q, we count the number of times a piece of j with length q is scheduled and set $y^{(j)}_{(q,s_j)}$ accordingly. It is easy to see that the constraints are satisfied.

Now, let (x, y) be a solution to the MCIP with objective value at most $m\bar{T} - L$. We use the solution to construct a schedule: For each configuration C we reserve x_C machines. On each of these machines we create C_h slots of length h for each module size $h \in H$. Note that because of (6.9), there is the exact right number of machines for this. Next, consider each job j and possible job part length q and create $y^{(j)}_{(q,s_j)}$ split pieces of length q and place them together with a setup of s_j into a slot of length $s_j + q$ on any machine. Because of (6.11), the entire job is split up by this, and because of (6.10), there are enough slots for all the job pieces. Note that the used space in the created schedule is equal to the objective value of (x, y) and therefore there is at least L free space. □

RESULT. Summing up, we can find a schedule of length at most $(1 + 3\varepsilon)T$ or correctly determine that there is no schedule of length T with the following procedure:

ALGORITHM 6.14.

1. Generate the modified instance I_2:
   - Remove the small setup jobs.
   - Round the setup and processing times of the remaining jobs.
2. Build and solve the MCIP for this case.
3. If the IP is infeasible, or the objective value greater than $m\bar{T} - L$, report that I has no solution with makespan T.
4. Otherwise build the schedule with makespan $\bar{T}$ and free space at least L for $\bar{I}$.
5. Transform the schedule into a schedule for I with makespan at most $\breve{T}$:
   - Use the original processing and setup times.
   - Greedily insert the small setup jobs.

To assess the running time of the procedure, we mainly need to bound the parameters of the MCIP, namely $|\mathcal{B}|$, $|H|$, $|\mathcal{M}|$, $|\mathcal{C}|$ and D. By definition, we have $|\mathcal{B}| = |\mathcal{J}'| \leqslant n$ and $D = 1$. Since all setup times and job piece lengths are multiples of $\varepsilon^2 T$ and bounded by T, we have $|\mathcal{M}| = \mathcal{O}(1/\varepsilon^4)$ and $|H| = \mathcal{O}(1/\varepsilon^2)$. This yields $|\mathcal{C}| \leqslant |H|^{\mathcal{O}(1/\varepsilon)} = 2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}$ because the size of each module is at least εT and the size of the configurations bounded by $(1+2\varepsilon)T$.

According to Observation 6.3, we now have a brick number of $|\mathcal{B}| = n$, brick-size $t = 2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}$, $r = |H| + 1 = \mathcal{O}(1/\varepsilon^2)$ globally uniform, and $s = D = 1$ locally uniform constraints. Because of the scaling step, all occurring numbers in the constraint matrix of the MCIP are bounded by $\mathcal{O}(1/\varepsilon^2)$, and therefore we have $\Delta = \mathcal{O}(1/\varepsilon^2)$. Furthermore, each occurring number can be bounded by $\mathcal{O}(m/\varepsilon^2)$ and this is an upper bound for each variable as well yielding $\varphi = \mathcal{O}(\log(m/\varepsilon^2))$. Hence, the MCIP can be solved in time:

$$(rs\Delta)^{\mathcal{O}(r^2 s + s^2)} \varphi^2 nt \log^5(nt) = 2^{\mathcal{O}(1/\varepsilon^4 \log 1/\varepsilon)} n \log^2(m) \log^5(n)$$

While the first step of the procedure is obviously dominated by the above, this is not the case for the remaining ones. In particular, building the schedule from the IP solution has linear costs in both n and m if the procedure described in the proof of Lemma 6.13 is realized in a straight-forward fashion. Note that the number of machines m could be exponential in the number of jobs, and therefore the described procedure is a PTAS only for the special case of $m = \text{poly}(n)$. However, this limitation can be overcome with a little extra effort, as we discuss in Section 6.5.

THEOREM 6.15: The algorithm for the splittable model finds a schedule with makespan at most $(1+3\varepsilon)T$ or correctly determines that there is no schedule with makespan T in time:

$$2^{\mathcal{O}(1/\varepsilon^4 \log 1/\varepsilon)} nm \log(m) \log^5(n)$$

Preemptive Model

In the preemptive model, we have to actually consider the timeline of the schedule on each machine, instead of just the assignment of the jobs or job pieces, and this causes some difficulties. For instance, we will have to argue that it suffices to look for a schedule with few possible starting points, and we will have to introduce additional constraints in the IP in order to ensure that pieces of the same job do not overlap. Our first step in dealing with this extra difficulty is to introduce some concepts and notation: For a given schedule with a makespan bound T, we call a job piece together with its setup a *block*, and we call the schedule X-layered, for some value X, if each block starts at a multiple of X. Corresponding to this, we call the time in the schedule between two directly succeeding multiples of X a *layer* and the corresponding time on a single machine a *slot*. We number the layers bottom to top and identify them with their number, that is, the set of layers Ξ is given by $\{\ell \in \mathbb{Z}_{>0} \mid (\ell-1)X \leqslant T\}$. Note that in an X-layered schedule there is at most one block in each slot, and for each layer there can be at most one block of each job present. Furthermore, we slightly alter the definition of free space for X-layered schedules: We solely count the space from slots that are completely free. If in such a schedule for each job there is at most one slot occupied by this job but not fully filled, we additionally call the schedule *layer-compliant*.

Simplification of the Instance

In the preemptive model, we distinguish *big*, *medium* and *small* setup jobs using two parameters δ and μ: The big setup jobs $\mathcal{J}^{\text{bst}}$ are those with setup time at least δT, the small $\mathcal{J}^{\text{sst}}$ have a setup time smaller than μT, and the medium $\mathcal{J}^{\text{mst}}$ are the ones in between. We set $\mu = \varepsilon^2\delta$ and choose $\delta \in \{\varepsilon^1, \dots, \varepsilon^{2/\varepsilon^2}\}$ such that the summed up processing time together with the summed up setup time of the medium setup jobs is upper bounded by $m\varepsilon T$, i.e., $\sum_{j\in\mathcal{J}^{\text{mst}}}(s_j + p_j) \leqslant m\varepsilon T$. If there is a schedule with makespan T, such a choice is possible because of the pidgeon hole principle and because the setup time of each job has to occur at least once in any schedule. Similar arguments are widely used, e.g., in the context of geometrical packing algorithms. Furthermore, we distinguish the jobs by processing times calling those with processing time at least εT *big* and the others *small*. For a given set of jobs J, we call the subsets of big or small jobs J_{big} or J_{small}, respectively. In Table 6.2, we present an overview of the job classification described above.

p_j \ s_j	$< \mu T$	$\geqslant \mu T, < \delta T$	$\geqslant \delta T$
$< \varepsilon T$	$\mathcal{J}^{sst}_{small}$	$\mathcal{J}^{mst}_{small}$	$\mathcal{J}^{bst}_{small}$
$\geqslant \varepsilon T$	$\mathcal{J}^{sst}_{big}$	$\mathcal{J}^{mst}_{big}$	$\mathcal{J}^{bst}_{big}$

Table 6.2: Overview on the job classifications

We perform three simplification steps, aiming for an instance in which the small and medium setup jobs are big; small setup jobs have setup time 0; and for which an $\varepsilon\delta T$-layered, layer-compliant schedule exists.

Let I_1 be the instance we get by removing the small jobs with medium setup times $\mathcal{J}^{mst}_{small}$ from the given instance I.

LEMMA 6.16. *If there is a schedule with makespan at most T for I, then there is also such a schedule for I_1; and if there is a schedule with makespan at most T' for I_1, then there is a schedule with makespan at most $T' + (\varepsilon + \delta)T$ for I.*

Proof. The first claim is obvious. For the second, we create a sequence containing the jobs from $\mathcal{J}^{mst}_{small}$ each directly preceded by its setup time. Recall that the overall length of the objects in this sequence is at most $m\varepsilon T$, and the length of each job is bounded by εT. We greedily insert the objects from the sequence considering each machine in turn. On the current machine, we start at time $T' + \delta T$ and keep inserting until $T' + \delta T + \varepsilon T$ is reached. If the current object is a setup time, we discard it and continue with the next machine and object. If, on the other hand, it is a job, we split it such that the remaining space on the current machine can be perfectly filled. We can place all objects like this, however the first job part placed on a machine might be missing a setup. We can insert the missing setups because they have length at most δT and between time T' and $T' + \delta T$ there is free space. □

Next, we consider the jobs with small setup times: Let I_2 be the instance we get by removing the small jobs with small setup times $\mathcal{J}^{sst}_{small}$ and setting the setup time of the big jobs with small setup times to zero, i.e., $\bar{s}_j = 0$ for each $j \in \mathcal{J}^{sst}_{big}$. Note that in the resulting instance each small job has a big setup time. Furthermore, let $L := \sum_{j \in \mathcal{J}^{sst}_{small}} p_j + s_j$. Then L is an obvious lower bound for the space taken up by the jobs from $\mathcal{J}^{sst}_{small}$ in any schedule.

LEMMA 6.17. *If there is a schedule with makespan at most T for I_1, then there is also a (T, L)-schedule for I_2; and if there is a γT-layered (T', L)-schedule for I_2 with T' a multiple of γT, then there is also a schedule with makespan at most $(1 + \gamma^{-1}\mu)T' + (\mu + \varepsilon)T$ for I_1.*

Proof. The first claim is obvious, and for the second consider a γT-layered (T', L)-schedule for I_2. We create a sequence that contains the jobs of $\mathcal{J}^{sst}_{small}$ and their setups such that each job is directly preceded

by its setup. Remember that the remaining space in partly filled slots is not counted as free space. Hence, since the overall length of the objects in the sequence is L, there is is enough space in the free slots of the schedule to place them. We do so in a greedy fashion guaranteeing that each job is placed on exactly one machine: We insert the objects from the sequence into the free slots considering each machine in turn, starting on the current machine from the beginning of the schedule, and moving on towards its end. If an object cannot be fully placed into the current slot there are two cases: It could be a job or a setup. In the former case, we cut it and continue placing it in the next slot, or, if the current slot was the last one, we place the rest at the end of the schedule. In the latter case, we discard the setup and continue with the next slot and object. The resulting schedule is increased by at most εT, which is caused by the last job placed on a machine.

To get a proper schedule for I_1 we have to insert some setup times: For the large jobs with small setup times and for the jobs that were cut in the greedy procedure. We do so by inserting a time window of length μT at each multiple of γT and at the end of the original schedule on each machine. By this, the schedule is increased by at most $\gamma^{-1}\mu T' + \mu T$. Since all the job parts in need of a setup are small and did start at multiples of μT or at the end, we can insert the missing setups. Note that blocks that span over multiple layers are cut by the inserted time windows. This, however, can easily be repaired by moving the cut pieces properly down. □

We continue by rounding the medium and big setup and all the processing times. In particular, we round the processing times and the big setup times up to the next multiple of $\varepsilon\delta T$ and the medium setup times to the next multiple of $\varepsilon\mu T$, i.e., $\bar{p}_j = \lceil p_j/(\varepsilon\delta T)\rceil \varepsilon\delta T$ for each job j, $\bar{s}_j = \lceil s_j/(\varepsilon\delta T)\rceil \varepsilon\delta T$ for each big setup job $j \in \mathcal{J}^{\text{bst}}$, and $\bar{s}_j = \lceil s_j/(\varepsilon\mu T)\rceil \varepsilon\mu T$ for each medium setup job $j \in \mathcal{J}^{\text{mst}}_{\text{big}}$.

LEMMA 6.18. *If there is a* (T, L)*-schedule for* I_2*, then there is also an* $\varepsilon\delta T$*-layered, layer-compliant* $((1+3\varepsilon)T, L)$*-schedule for* I_3*; and if there is a* γT*-layered* (T', L)*-schedule for* I_3*, then there is also such a schedule for* I_2*.*

While the second claim is easy to see, the proof of the first is rather elaborate and unfortunately a bit tedious. Hence, since we believe Lemma 6.18 to be fairly plausible by itself, we postpone its proof to the end of the section and proceed discussing its use.

For the big and small setup jobs, both processing and setup times are multiples of $\varepsilon\delta T$. Therefore, the length of each of their blocks in an $\varepsilon\delta T$-layered, layer-compliant schedule is a multiple of $\varepsilon\delta T$. For a medium setup job, on the other hand, we know that the overall length of its blocks has the form $x\varepsilon\delta T + y\varepsilon\mu T$, with non-negative integers x and y. In particular, it is a multiple of $\varepsilon\mu T$ because $\varepsilon\delta T = (1/\varepsilon^2)\varepsilon\mu T$. In a $\varepsilon\delta T$-layered, layer-compliant schedule, for each medium setup job the length of all but at most one block is a multiple of $\varepsilon\delta T$ and

therefore a multiple of $\varepsilon\mu T$. If both the overall length and the lengths of all but one block are multiples of $\varepsilon\mu T$, this is also true for the one remaining block. Hence, we will use the MCIP not to find an $\varepsilon\delta T$-layered, layer-compliant schedule in particular, but an $\varepsilon\delta T$-layered one with block sizes as described above and maximum free space.

Based on the simplification steps, we define two makespan bounds $\bar{T}$ and $\breve{T}$: Let $\bar{T}$ be the makespan bound we get by the application of the Lemmata 6.16-6.18, i.e., $\bar{T} = (1+3\varepsilon)T$. We will use the MCIP to find an $\varepsilon\delta T$-layered $(\bar{T}, L)$-schedule for I_3 and apply the Lemmata 6.16-6.18 backwards to get a schedule for I with makespan at most $\breve{T} = (1+(\varepsilon\delta)^{-1}\mu)\bar{T} + (\mu+\varepsilon)T + (\varepsilon+\delta)T \leqslant (1+9\varepsilon)T$ (using $\varepsilon \leqslant 1/2$).

Utilization of the MCIP

Similar to the splittable case, the basic objects are the (big) jobs, i.e., $\mathcal{B} = \mathcal{J}_{big}$, and their single value is their processing time ($D = 1$). The modules, on the other hand, are more complicated, because they additionally need to encode which layers are exactly used and, in case of the medium jobs, to which degree the last layer is filled. For the latter, we introduce buffers, representing the unused space in the last layer and define modules as tuples (ℓ, q, s, b) of starting layer, job piece size, setup time and buffer size. For a module $M = (\ell, q, s, b)$, we write $\ell_M = \ell$, $q_M = q$, $s_M = s$ and $b_M = b$, and we define the size $\Lambda(M)$ of M as $s + q + b$. The overall set of modules $\mathcal{M}$ is the union of the modules for big, medium and small setup jobs $\mathcal{M}^{bst}$, $\mathcal{M}^{mst}$ and $\mathcal{M}^{sst}$ that are defined in the following. For this, let $Q^{bst} = \{q \mid q = x\varepsilon\delta T, x \in \mathbb{Z}_{>0}, q \leqslant \bar{T}\}$ and $Q^{mst} = \{q \mid q = x\varepsilon\mu T, x \in \mathbb{Z}_{>0}, q \leqslant \bar{T}\}$ be the sets of possible job piece sizes of big and medium setup jobs; $S^{bst} = \{s \mid s = x\varepsilon\delta T, x \in \mathbb{Z}_{\geqslant 1/\varepsilon}, s \leqslant \bar{T}\}$ and $S^{mst} = \{s \mid s = x\varepsilon\mu T, x \in \mathbb{Z}_{\geqslant 1/\varepsilon}, s \leqslant \delta T\}$ be the sets of possible big and medium setup times; $B = \{b \mid b = x\varepsilon\mu T, x \in \mathbb{Z}_{\geqslant 0}, b < \varepsilon\delta T\}$ the set of possible buffer sizes; and $\Xi = \{1, \dots, 1/(\varepsilon\delta) + 3/\delta\}$ the set of layers. We set:

$$\begin{aligned}
\mathcal{M}^{bst} &= \left\{(\ell, q, s, 0) \,\middle|\, \ell \in \Xi, q \in Q^{bst}, s \in S^{bst}, (\ell-1)\varepsilon\delta T + s + q \leqslant \bar{T}\right\} \\
\mathcal{M}^{mst} &= \Big\{(\ell, q, s, b) \in \Xi \times Q^{mst} \times S^{mst} \times B \,\Big|\, \\
&\qquad\qquad x = s + q + b \in \varepsilon\delta T\mathbb{Z}_{>0}, (\ell-1)\varepsilon\delta T + x \leqslant \bar{T}\Big\} \\
\mathcal{M}^{sst} &= \left\{(\ell, \varepsilon\delta T, 0, 0) \,\middle|\, \ell \in \Xi\right\}
\end{aligned}$$

Concerning the small setup modules, note that the small setup jobs have a setup time of 0 and therefore may be covered slot by slot. We establish $\varepsilon\mu T = 1$ via scaling, to ensure integral values. A big, medium or small job is eligible for a module if it is also big, medium or small, respectively, and the setup times fit.

We have to avoid that two modules M_1, M_2 whose corresponding time intervals overlap are used to cover the same job or in the same configuration. Such an overlap is given if there is some layer ℓ used by both of them, that is, $(\ell_M - 1)\varepsilon\delta T \leqslant (\ell-1)\varepsilon\delta T < (\ell_M - 1)\varepsilon\delta T + \Lambda(M)$

for both $M \in \{M_1, M_2\}$. Hence, for each layer $\ell \in \Xi$, we set $\mathcal{M}_\ell \subseteq \mathcal{M}$ to be the set of modules that use layer ℓ. Furthermore, we partition the modules into groups Γ by size and starting layer, i.e., $\Gamma = \{G \subseteq \mathcal{M} \mid M, M' \in G \iff \Lambda(M) = \Lambda(M') \wedge \ell_M = \ell_{M'}\}$. The size of a group $G \in \Gamma$ is the size of a module from G, i.e. $\Lambda(G) = \Lambda(M)$ for $M \in G$. Unlike before we consider *configurations of module groups* rather than module sizes. More precisely, the set of configurations $\mathcal{C}$ is given by the configurations of groups, such that for each layer at most one group using this layer is chosen, i.e., $\mathcal{C} = \{C \in \mathbb{Z}^{\Gamma}_{\geqslant 0} \mid \forall \ell \in \Xi : \sum_{G \subseteq \mathcal{M}_\ell} C_G \leqslant 1\}$. With this definition we prevent overlap conflicts on the machines. Note that unlike in the cases considered so far, the size of a configuration does not correspond to a makespan in the schedule, but to used space, and the makespan bound is realized in the definition of the modules instead of in the definition of the configurations. To also avoid conflicts for the jobs, we extend the basic MCIP with additional locally uniform constraints. In particular, the constraints of the extended MCIP for the above definitions with adapted notation and without duplication of the configuration variables are given by:

$$\sum_{C \in \mathcal{C}} x_C = m \tag{6.12}$$

$$\sum_{C \in \mathcal{C}(T)} C_G x_C = \sum_{j \in \mathcal{J}} \sum_{M \in G} y_M^{(j)} \qquad \forall G \in \Gamma \tag{6.13}$$

$$\sum_{M \in \mathcal{M}} q_M y_M^{(j)} = p_j \qquad \forall j \in \mathcal{J} \tag{6.14}$$

$$\sum_{M \in \mathcal{M}_\ell} y_M^{(j)} \leqslant 1 \qquad \forall j \in \mathcal{J}, \ell \in \Xi \tag{6.15}$$

Like in the first two cases, we minimize the summed-up size of the configurations via the objective function $\sum_C \Lambda(C) x_C$. Note that in this case the size of a configuration does not have to equal its height. It is easy to see that the last constraint is indeed locally uniform. However, since we have an inequality instead of an equality, we have to introduce $|\Xi|$ slack variables in each brick, yielding:

Observation 6.19. The MCIP extended like above is an n-fold IP with brick-size $t = |\mathcal{M}| + |\mathcal{C}| + |\Xi|$, brick number $n = |\mathcal{J}|$, $r = |\Gamma| + 1$ globally uniform and $s = D + |\Xi|$ locally uniform constraints.

LEMMA 6.20. *With the above definitions, there is an $\varepsilon\delta T$-layered $(\bar{T}, L)$-schedule for I_3 in which the length of a block is a multiple of $\varepsilon\delta T$ if it belongs to a small or big setup job or a multiple of $\varepsilon\mu T$ otherwise, if and only if the extended* MCIP *has a solution with objective value at most $m\bar{T} - L$.*

Proof. We first consider such a schedule for I_3. For each machine, we can derive a configuration that is given by the starting layers of the blocks together with the summed-up length of the slots the respective block is scheduled in. The size of the configuration C is equal to the used space on the respective machine. Hence, we can fix some arbitrary

job j and set $x_C^{(j)}$ to the number of machines corresponding to j (and $x_C^{(j')} = 0$ for $j' \neq j$). Keeping in mind that in an $\varepsilon\delta T$-layered schedule the free space is given by the free slots, the above definition yields an objective value bounded by $m\bar{T} - L$ because there was free space of at least L. Next, we consider the module variables for each job j in turn: If j is a small setup job, we set $y^{(j)}_{(\ell,\varepsilon\delta T,0,0)}$ to 1 if j occurs in ℓ and to 0 otherwise. Now, let j be a big setup job. For each of its blocks, we set $y^{(j)}_{(\ell,z-s_j,s_j,0)} = 1$, where ℓ is the starting layer and z the length of the block. The remaining variables are set to 0. Lastly, let j be a medium setup job. For each of its blocks, we set $y^{(j)}_{(\ell,z-s_j,s_j,b)} = 1$, where ℓ is the starting layer of the block, z its length and $b = \lceil z/(\varepsilon\delta T)\rceil\varepsilon\delta T - z$. Again, the remaining variables are set to 0. It is easy to verify that all constraints are satisfied by this solution.

If, on the other hand, we have a solution (x, y) to the MCIP with objective value at most $m\bar{T} - L$, we reserve $\sum_j x_C^{(j)}$ machines for each configuration C. There are enough machines to do this, because of (6.12). On each of these machines we reserve space: For each $G \in \Gamma$, we create an allocated space of length $\Lambda(G)$ starting from the starting layer of G if $C_G = 1$. Let j be a job and ℓ be a layer. If j has a small setup time, the variable $y^{(j)}_{(\ell,\varepsilon\delta T,0,0)}$ may have the value 0 or 1. In the latter case, we create a piece of length $\varepsilon\delta T$ and place it into an allocated space of length $\varepsilon\delta T$ in layer ℓ. If, on the other hand, j is a big or medium setup job, we consider each possible job part length $q \in Q^{bst}$ or $q \in Q^{mst}$, respectively, create $y^{(j)}_{(\ell,q,s_j,0)}$ or $y^{(j)}_{(\ell,q,s_j,b)}$ (with $b = \lceil q/(\varepsilon\delta T)\rceil\varepsilon\delta T - \varepsilon\delta T$) pieces of length q, and place them together with their setup time into allocated spaces of length q in layer ℓ. Because of (6.14), the entire job is split up by this, and because of (6.13), there are enough allocated spaces for all the job pieces. The makespan bound is ensured by the definition of the modules, and overlaps are avoided due to the definition of the configurations and (6.15). Furthermore, the used slots have an overall length equal to the objective value of (x, y) and therefore there is at least L free space. □

Result

Summing up the above considerations, we get:

ALGORITHM 6.21.

1. Determine a suitable class of medium setup jobs. If there is no such class, report that there is no schedule with makespan T and terminate the procedure.
2. Generate the modified instance I_3:
   - Remove the small jobs with medium setup times.
   - Remove the small jobs with small setup times, and decrease the setup time of big jobs with small setup time to 0.

- Round the big processing times, as well as the medium, and the big setup times.

3. Build and solve the MCIP for I_3.
4. If the MCIP is infeasible, or the objective value greater than $m\bar{T} - L$, report that I has no solution with makespan T.
5. Otherwise build the $\varepsilon\delta T$-layered schedule with a makespan of at most $\bar{T}$ and a free space of at least L for I_3.
6. Transform the schedule into a schedule for I with makespan at most $\breve{T}$:
   - Use the prerounding processing and setup times.
   - Insert the small jobs with small setup times into the free slots and insert the setup times of the big jobs with small setup times.
   - Insert the small jobs with medium setup times.

We analyze the running time of the procedure and start by bounding the parameters of the extended MCIP. We have $|\mathcal{B}| = n$ and $D = 1$ by definition, and the number of layers $|\Xi|$ is obviously $\mathcal{O}(1/(\varepsilon\delta)) = \mathcal{O}(1/\varepsilon^{2/\varepsilon+1}) = 2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}$. Furthermore, it is easy to see that $|Q^{bst}| = \mathcal{O}(1/(\varepsilon\delta))$, $|Q^{mst}| = \mathcal{O}(1/(\varepsilon^3\delta))$, $|S^{bst}| = \mathcal{O}(1/(\varepsilon\delta))$, $|S^{mst}| = \mathcal{O}(1/(\varepsilon^3))$, and $|B| = \mathcal{O}(1/\varepsilon^2)$. This gives us

- $|\mathcal{M}^{bst}| \leqslant |\Xi||Q^{bst}||S^{bst}|$,
- $|\mathcal{M}^{mst}| \leqslant |\Xi||Q^{mst}||S^{mst}||B|$, and
- $|\mathcal{M}^{sst}| = |\Xi|$, and therefore
- $|\mathcal{M}| = |\mathcal{M}^{bst}| + |\mathcal{M}^{mst}| + |\mathcal{M}^{sst}| = 2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}$.

Since there are $\mathcal{O}(1/(\delta\varepsilon))$ distinct module sizes, the number of groups $|\Gamma|$ can be bounded by $\mathcal{O}(|\Xi|/(\varepsilon\delta)) = 2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}$. Hence, for the number of configurations we get $|\mathcal{C}| = \mathcal{O}((1/(\varepsilon\delta))^{|\Gamma|}) = 2^{2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}}$. By Observation 6.19, the modified MCIP has $r = 2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}$ many globally and $s = 2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}$ many locally uniform constraints; its brick number is n, and its brick size is $t = 2^{2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}}$. All occurring values in the matrix are bounded by $\bar{T}$ yielding $\Delta \leqslant \bar{T} = 1/(\varepsilon\mu) + 1/\mu = 2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}$ due to the scaling step. Furthermore, the numbers in the input can be bounded by $m2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}$ and all variables can be upper bounded by $\mathcal{O}(m)$. Hence, we have $\varphi = \mathcal{O}(\log m + 1/\varepsilon \log 1/\varepsilon)$, and due to Theorem 6.2, we can solve the MCIP in time:

$$(rs\Delta)^{\mathcal{O}(r^2s+s^2)}\varphi^2 nt \log^5(nt) = 2^{2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}} n \log^2(m) \log^5(n)$$

A straight-forward realization of the procedure for the creation of the $\varepsilon\delta T$-layered $(\bar{T}, L)$-schedule for I_3 (the fifth step), which is described in the proof of Lemma 6.20, is linear with respect to m, yielding:

THEOREM 6.22: The algorithm for the preemptive model finds a schedule with makespan at most $(1+9\varepsilon)T$ or correctly determines that there is no schedule with makespan T in time:

$$2^{2^{\mathcal{O}(1/\varepsilon \log 1/\varepsilon)}} nm \log(m) \log^5(n)$$

Proof of Lemma 6.18

We divide the proof into three steps, which can be summarized as follows:

1. We transform a given (T, L)-schedule for I_2 into a $((1+3\varepsilon)T, L)$-schedule for I_3 in which the big setup jobs are already properly placed inside the layers.
2. We construct a flow network with integer capacities and a maximum flow based on the placement of the remaining jobs in the layers.
3. Using flow integrality and careful repacking, we transform the schedule into a $\varepsilon\delta T$-layered, layer-compliant schedule.

More precisely, the above transformation steps will produce a $\varepsilon\delta T$-layered, layer-compliant $((1+3\varepsilon)T, L)$-schedule with the additional properties that too much processing time may be inserted for some jobs or setup times are produced that are not followed by the corresponding job pieces. Note that this does not cause any problems: We can simply remove the extra setups and processing time pieces. For the medium jobs, this results in a placement with at most one used slot that is not fully filled, as required in a layer-compliant schedule.

STEP 1. Remember that a block is a job piece together with its setup time placed in a given schedule. Consider a (T, L)-schedule for I_2 and suppose that for each block in the schedule there is a container perfectly encompassing it. Now, we stretch the entire schedule by a factor of $(1+3\varepsilon)$ and in this process we stretch and move the containers correspondingly. The blocks are not stretched but moved in order to stay in their container, and we assume that they are positioned at the bottom, that is, at the beginning of the container. Note that we could move each block inside its respective container without creating conflicts with other blocks belonging to the same job. In the following, we use the extra space to modify the schedule. Similar techniques are widely used in the context of geometric packing algorithms.

Let j be a big setup job. In each container containing a block belonging to j, there is a free space of at least $3\varepsilon\delta T$ because the setup time of j is at least δT and therefore the container had at least that length before the stretching. Hence, we have enough space to perform the following two steps. We move the block up by at most $\varepsilon\delta T$ such that it starts at a multiple of $\varepsilon\delta T$. Next, we enlarge the setup time and the

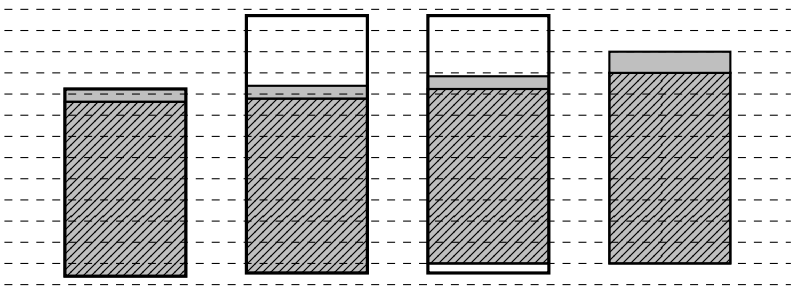


Figure 6.1: The stretching and rounding steps, for a small job part with big setup time starting in the first layer of the schedule, depicted from left to right: The schedule and the containers are stretched; the block is moved up; and the processing and the setup time are increased. The hatched part represents the setup time, the thick rectangle the container, and the dashed lines the layers, with $\varepsilon = \delta = 1/8$.

processing time by at most $\varepsilon\delta T$ such that both are multiples of $\varepsilon\delta T$. Now the setup time is equal to the rounded setup time, while the processing time might be bigger because we performed this step for each piece of the job. We outline the procedure in Figure 6.1.

We continue with the small setup jobs. These jobs are big and therefore for each of them there is a summed up free space of at least $3\varepsilon^2 T$ in the containers belonging to the respective job—more than enough to enlarge some of the pieces such that their overall length matches the rounded processing time.

Lastly, we consider the medium setup jobs. These jobs are big as well and we could apply the same argument as above, but we need to be a little bit more careful in order to additionally realize the rounding of the setup times and an additional technical step we need in the following. Fix a medium setup job j and a container filled with a block belonging to j. Since the setup time has a length of at least μT, the part of the container filled with it was increased by at least $3\varepsilon\mu T$. Hence, we can enlarge the setup time to the rounded setup time without using up space in the container that was created due to the processing time part. We do this for all blocks belonging to medium setup jobs. The extra space in the containers of a medium setup job due to the processing time parts is still at least $3\varepsilon^2 T \geqslant 3\varepsilon\delta T$. For each medium setup job j, we spend at most $\varepsilon\delta T$ of this space to enlarge its processing time to its rounded size and again at most $\varepsilon\delta T$ to create a little bit of extra processing time in the containers belonging to j. The size of this extra processing time is bounded by $\varepsilon\delta T$ and chosen in such a way that the overall length of all blocks belonging to j in the schedule is also a multiple of $\varepsilon\delta T$. Because of the rounding, the length of the added extra processing time for each j is a multiple of $\varepsilon\mu T$. The purpose of the extra processing time is to ensure integrality in the flow network, which is constructed in the next step.

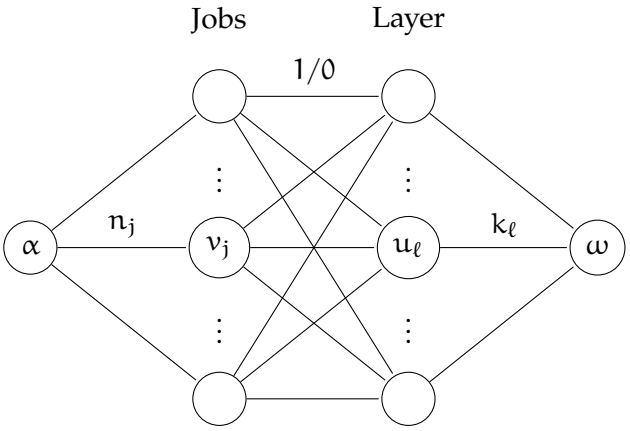


Figure 6.2: Flow network for layers and partially scheduled jobs.

Note that the free space that was available in the original schedule was not used in the above steps, in fact, it was even increased by the stretching. Hence, we have created a $((1+3\varepsilon)T, L)$-schedule for I_3—or a slightly modified version thereof—and the big setup jobs are already well-behaved with respect to the $\varepsilon\delta T$-layers, that is, they start at multiples of $\varepsilon\delta T$ and fully fill the slots they are scheduled in.

STEP 2. Note that for each job j and layer $\ell \in \Xi$, the overall length $q_{j,\ell}$ of job and setup pieces belonging to j and placed in ℓ is bounded by $\varepsilon\delta T$. We say that j is *fully*, or *partially*, or *not* scheduled in layer ℓ if $q_{j,\ell} = 1$, or $q_{j,\ell} \in (0,1)$, or $q_{j,\ell} = 0$, respectively. Let X_j be the set of layers in which j is scheduled partially and Y_ℓ the set of (medium or small setup) jobs partially scheduled in ℓ. Then $a_j = \sum_{\ell \in X_j} q_{j,\ell}$ is a multiple of $\varepsilon\delta T$, and we set $n_j = a_j/(\varepsilon\delta T)$. Furthermore, let $b_\ell = \sum_{j \in Y_\ell} q_{j,\ell}$ and $k_\ell = \lceil b_\ell/(\varepsilon\delta T) \rceil$.

Our flow network has the following structure: There is a node v_j for each medium or small setup job, a node u_ℓ for each layer ℓ, as well as a source α and a sink ω. The source node is connected to the job nodes via edges (α, v_j) with capacity n_j, and the layer nodes are connected to the sink via edges (u_ℓ, ω) with capacity k_ℓ. Lastly, there are edges (v_j, u_ℓ) between job and layer nodes with capacity 1 if j is partially scheduled in layer ℓ or 0 otherwise. In Figure 6.2, a sketch of the network is given.

The schedule can be used to define a flow f with value $\sum_j n_j$ in the network by setting $f(\alpha, v_j) = n_j$, $f(u_\ell, \omega) = b_\ell/(\varepsilon\delta T)$, and $f(v_j, u_\ell) = q_{j,\ell}/(\varepsilon\delta T)$. It is easy to verify that f is a maximum flow, and because all capacities in the flow network are integral, we can find another maximum flow f' with integral values.

STEP 3. We start by introducing some notation and a basic operation for the transformation of the schedule: Given two machines i and i' and a time t, a *machine swap* between i and i' at moment t produces

a schedule in which everything that was scheduled on i from t on is now scheduled on i' and vice versa. If on both machines there is either nothing scheduled at t, or blocks are starting or ending at t, the resulting schedule is still feasible. Moreover, if there is a block starting at t on one of the machines and another one belonging to the same job ending on the other, we can merge the two blocks and transform the setup time of the first into processing time. We assume in the following that we always merge if this is possible when performing a machine swap. Remember that by definition blocks belonging to the same job cannot overlap. However, if there was overlap, it could be eliminated using machine swaps [133].

If a given slot only contains pieces of jobs that are partially scheduled in the layer, we call the slot *usable*. Furthermore, we say that a job j is *flow assigned* to layer ℓ if $f'(v_j, u_\ell) = 1$. In the following, we will iterate through the layers, create as many usable slots as possible, reserve them for flow assigned jobs, and fill them with processing and setup time of the corresponding jobs later on. To do so, we have to distinguish different types of blocks belonging to jobs that are partially placed in a given layer: Inner blocks which lie completely inside the layer and touch at most one of its borders, upper cross-over blocks which start inside the layer and end above it, and lower cross-over blocks which start below the layer and end inside it. When manipulating the schedule layer by layer, the cross-over jobs obviously can cause problems. To deal with this, we will need additional concepts: A *repair piece* for a given block is a piece of setup time of length less than $\varepsilon\delta T$, with the property that the block and the repair piece together make up exactly one setup of the respective job. Hence, if a repair-piece is given for a block, the block is comprised completely of setup time. Moreover, we say that a slot reserved for a job j has a *dedicated setup* if there is a block of j including a full setup *starting or ending* inside the slot.

In the following, we give a detailed description of the transformation procedure followed by a high-level summarization. The procedure runs through two phases. In the first phase the layers are transformed one after another from bottom to top. After a layer is transformed the following invariants will always hold:

1. A scheduled block either includes a full setup or has a repair piece. In the latter case it was an upper cross-over block in a previous iteration.

2. Reserved slots that are not full have a dedicated setup.

Note that the invariants are trivially fulfilled in the beginning. During the first phase, we remove some job and setup parts from the schedule that are reinserted into the reserved slots in the second phase. Let $\ell \in \Xi$ denote the current layer.

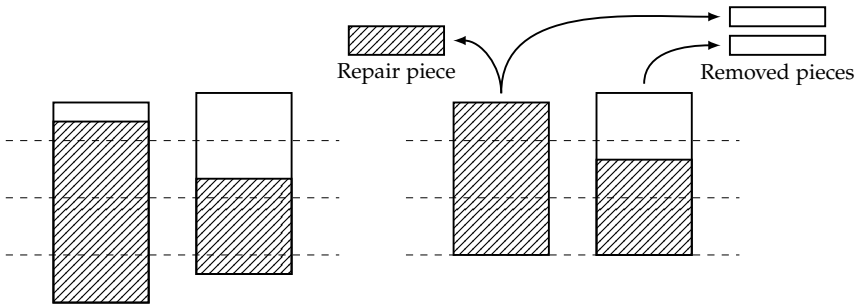


Figure 6.3: The rectangles represent blocks, the hatched parts the setup times, and the dashed lines layer borders. The push and cut step is performed on two blocks. For one of the two a repair piece is created.

In the first step, our goal is to ensure that jobs that are fully scheduled in ℓ occupy exactly one slot thereby creating as many usable slots as possible. Let j be a job that is fully scheduled in layer ℓ. If there is a block belonging to j and ending inside the layer at time t, there is another block belonging to j and starting at t because j is fully scheduled in ℓ and there are no overlaps. Hence, we can perform a machine swap at time t between the two machines the blocks are scheduled on. We do so for each job fully scheduled in the layer and each corresponding pair of blocks. After this step, there are at least k_ℓ usable slots and at most k_ℓ flow assigned jobs in layer ℓ.

Next, we consider upper cross-over blocks of jobs that are partially scheduled in the layer ℓ but are not flow assigned to it. These are the blocks that cause the most problems, and we perform a so-called *push and cut step* (see Figure 6.3) for each of them: If q is the length of the part of the block lying in ℓ, we cut away the upper part of the block of length q and move the remainder up by q. If the piece we cut away does contain some setup time, we create a repair piece for the block out of this setup time. The processing time part of the piece, on the other hand, is removed. Note that this step preserves the first invariant. The repair piece is needed in the case that the job corresponding to the respective block is flow assigned to the layer in which the block ends.

We now remove all inner blocks from the layer as well as the parts of the upper and lower cross-over blocks that lie in the layer. After this all usable slots are completely free. Furthermore, note that the first invariant might be breached by this.

Next, we arbitrarily reserve usable slots for jobs flow assigned to the layer. For this, note that due to the definition of the flow network, there are at most k_ℓ jobs flow assigned to the layer and there are at least as many usable slots, as noted above. This step might breach the second invariant as well. Using machine swaps at the upper and lower border of the layer, we then ensure that the upper and lower cross-over blocks of the jobs flow assigned to the layer lie on the same

machine as the reserved slot. Note that for each job there can be at most one upper or lower cross-over block, respectively, in the layer.

To restore the invariants, we perform the following repair steps for each job j flow assigned to the layer:

CASE 1. If there is an upper cross-over block for j or a lower cross-over block without a repair peace, we reinsert the removed part (or parts) at the end or beginning of the slot, respectively. This provides a dedicated setup for the job and furthermore the first invariant once again holds for the respective cross-over blocks.

CASE 2. If there is neither an upper nor a lower block for j, there is an inner block belonging to j. This has to be the case because otherwise the capacity in the flow network between j and ℓ is 0, and j could not have been flow assigned to ℓ. Moreover, this inner block contains a full setup, and we can place it in the beginning of the slot thus providing the dedicated setup. The invariants are both restored.

CASE 3. The last possibility is that there is no upper cross-over block but a lower cross-over block with a repair piece. In this case, the removed part of the block is fully comprised of setup and we reinsert it in the beginning of the reserved slot. Furthermore, we insert as much setup of the repair piece as possible. If the repair piece is not used up, we now consider the remainder as the new repair piece of the block. Hence, the first invariant holds, and since the slot is full in this case, the second one holds as well. If, on the other hand, the full repair piece is inserted, we thereby provide a dedicated setup for the slot and the block once again contains a full setup. In this case, the jobs does not have a repair piece anymore.

After the first phase is finished, we have to deal with the removed pieces in the second one. The overall length of the reserved slots for a job j equals the overall length a_j of its setup and job pieces from layers in which j was partially scheduled. Since we did not create or destroy any job piece, we can place the removed pieces corresponding to job j into the remaining free space of the slots reserved for j, and we do so after transforming them completely into processing time. Because of the second invariant, there is a dedicated setup in each slot, however, it may be positioned directly above the newly inserted processing time. This can be fixed by switching the processing time with the top part of the respective setup time. Furthermore, there may be some blocks that still have a repair piece. We may remove these blocks together with their repair pieces.

Lastly, all remaining usable slots are completely free at the end of this procedure, and since the others are full, they have an overall size

of at least L. We conclude the proof of Lemma 6.18 with an overview of the transformation procedure.

ALGORITHM 6.23.
Phase 1: For each layer $\ell \in \Xi$, considered bottom to top, perform the following steps:

1. Use machine swaps to ensure that jobs fully scheduled in ℓ occupy exactly one slot.
2. For each upper cross-over block of a job partially scheduled but not flow assigned to ℓ perform a push and cut step.
3. Remove inner blocks and parts of cross-over blocks that lie in ℓ.
4. Reserve usable slots for jobs flow assigned to the layer.
5. Use machine swaps to ensure, that cross-over blocks of flow assigned jobs lie on the same machine as the reserved slot.
6. For each job j flow assigned to the layer, perform one of the repair steps.

Phase 2:

1. Transform all removed pieces into processing time and insert the removed pieces into the reserved slots.
2. If processing time has been inserted ahead of the dedicated setup of the slot, reschedule properly.
3. Remove blocks that still have a repair piece.

6.5 IMPROVEMENTS OF THE RUNNING TIME

In this section, we revisit the splittable and the setup time model. For the former, we address the problem of the running time dependence in the number of machines m, and for both, we present an improved rounding procedure yielding a better running time.

Splittable Model – Machine Dependence

In the splittable model, the number of machines m may be super-polynomial in the input size because it is not bounded by the number of jobs n. Hence, we need to be careful already when defining the schedule in order to get a polynomially bounded output. We say a machine is *composite* if it contains more than one job, and we say it is *plain* if it contains at most one job. For a schedule with makespan T, we call each machine *trivial* if it is plain and has load T or if it is empty and *nontrivial* otherwise. We say a schedule with makespan T is *simple* if the number of nontrivial machines is bounded by $\binom{n}{2}$.

LEMMA 6.24. *If there is a schedule with makespan* T *for* I *there is also a simple schedule with makespan at most* T.

Proof. Let there be a schedule with makespan T for I. For the first step, let us assume there are more than $\binom{n}{2}$ composite machines. In this case, there exist two distinct machines i_1 and i_2 and two distinct jobs j_1 and j_2 such that both machines contain parts of both jobs since there are at most $\binom{n}{2}$ different pairs of jobs. For $x, y \in \{1, 2\}$, let $t(x, y)$ be the processing time combined with the setup time of job $x \in \{j_1, j_2\}$ on machine $y \in \{i_1, i_2\}$. W.l.o.g., let $t(j_1, i_1)$ be the smallest value of the four. We swap this job part and its setup time with some of the processing time of the job j_2 on machine i_2. If the processing time of j_2 on i_2 is smaller than $t(j_1, i_1)$, there is no processing time of j_2 on i_2 left and we can discard the corresponding setup time. Afterwards, the makespan has not increased and at least one machine processes one job less. We can repeat this step iteratively until there are at most $\binom{n}{2}$ machines containing more than one job.

In the second step, we shift processing time from the composite machines to the plain ones. We do this for each job until it is either not contained on a composite machine or each plain machine containing this job has load T. If the job is no longer contained on a composite machine, we shift the processing time of the job such that all except one machine containing this job has load T. Since this job does not appear on any composite machine, the number of such machines can in this case be bounded by $\binom{n-1}{2}$ by repeating the first step. Therefore, the number of nontrivial machines is bounded by $\binom{n-i}{2} + i \leqslant \binom{n}{2}$ for some $i \in \{0, \dots, n\}$. □

For a simple schedule, a polynomial representation of the solution is possible: For each job, we state the number of trivial machines containing this jobs or fix a first and last trivial machine belonging to this job. This enables a polynomial encoding length of the output, given that the remaining parts of the jobs are not fragmented into too many parts which can be guaranteed using the results of Section 6.4.

To guarantee that the MCIP finds a simple solution, we need to modify it a little. We have to ensure that nontrivial configurations are not used to often. Let $\mathcal{C}' \subseteq \mathcal{C}$ be the set of nontrivial configurations, i.e., the set of configurations containing more than one module or one module with size smaller than T. We add the following globally uniform constraint to the MCIP:

$$\sum_{C \in \mathcal{C}'} x_C \leqslant \binom{|\mathcal{J}^{\text{bst}}|}{2} \tag{6.16}$$

Since this is an inequality, we have to introduce a slack variable increasing the brick size by one. Furthermore, the bound on the biggest number occurring in the input as well as the range of the variables

has to be increased by a factor of $\mathcal{O}(n^2)$, yielding a slightly altered running time for the MCIP of:

$$2^{\mathcal{O}(1/\varepsilon^4 \log 1/\varepsilon)} n \log^2(m) \log^6(n)$$

The number of modules with maximum size denotes for each job in $\mathcal{J}^{\text{bst}}$ how many trivial machines it uses. The other modules can be mapped to the nontrivial configurations and the jobs can be mapped to the modules.

We still have to schedule the jobs in $\mathcal{J}^{\text{sst}}$. We do this as described in the proof of Lemma 6.11. We fill the nontrivial machines greedily step by step starting with the jobs having the smallest processing time. When these machines are filled, there are some completely empty machines left. Now, we estimate how many machines can be completely filled with the current job j. This can be done, by dividing the remaining processing time by $T - s_i$ in $\mathcal{O}(1)$. The remaining part is scheduled on the next free machine. This machine is filled up with the next job and again the number of machines which can be filled completely with the rest of this new job is determined. These steps are iterated until all jobs in $\mathcal{J}^{\text{sst}}$ are scheduled. This greedy procedure needs at most $\mathcal{O}(|\mathcal{J}^{\text{bst}}|(|\mathcal{J}^{\text{bst}}| - 1) + |\mathcal{J}^{\text{sst}}|) = \mathcal{O}(n^2)$ operations. Therefore, we can avoid the dependence in the number of machines at the cost of a quadratic dependency in n in the running time.

Improved Rounding Procedures

To improve the running time in the splittable and setup class model, we reduce the number of module sizes via a geometric and an arithmetic rounding step. In both cases, the additional steps are performed following all the other simplification steps. The basic idea is to include setup times together with their corresponding job pieces or batches of jobs respectively into containers with suitably rounded sizes and to model these containers using the modules. The containers have to at least as big as the objects they contain and the load on a machine is given by the summed up sizes of the containers on the machine. Let H^* be a set of container sizes. Then a H^*-structured schedule is a schedule in which each setup time together with its corresponding job piece or batch of jobs is packed in a container with the smallest size $h \in H^*$ such that the summed up size of the setup and the job piece or batch of jobs is upper bounded by h.

SPLITTABLE MODEL. Consider the instance I_2 for the splittable model described in Section 6.4. In this instance, each setup and processing time is a multiple of $\varepsilon^2 T$ and we are interested in a schedule of length $(1+2\varepsilon)T$. For each multiple h of $\varepsilon^2 T$, let $\tilde{h} = (1+\varepsilon)^{\lceil \log_{1+\varepsilon} h/(\varepsilon^2 T)\rceil} \varepsilon^2 T$, $\bar{h} = \lceil \tilde{h}/\varepsilon^2 T \rceil \varepsilon^2 T$, and $\bar{H} = \{\bar{h} \mid h \in \varepsilon^2 T \mathbb{Z}_{\geqslant 1}, h \leqslant (1+2\varepsilon)^2 T\}$. Note that $|\bar{H}| \in \mathcal{O}(1/\varepsilon \log 1/\varepsilon)$

LEMMA 6.25. *If there is a* $((1+2\varepsilon)T, L')$*-schedule for* I_2 *in which the length of each job part is a multiple of* $\varepsilon^2 T$*, there is also a* $\bar{\mathcal{H}}$*-structured* $((1+2\varepsilon)^2 T, L')$*-schedule for* I_2 *with the same property.*

Proof. Consider such a schedule for I_2 and a pair of setup time s and job piece q scheduled on some machine. Let $h = s + q$. Stretching the schedule by $(1+2\varepsilon)$ creates enough space to place the pair into a container of size $\bar{h}$, because $(1+\varepsilon)h \leqslant \tilde{h}$, and $\varepsilon h \leqslant \varepsilon^2 T$, since $s \geqslant \varepsilon T$. □

To implement this lemma into the procedure, the processing time bounds $\bar{T}$ and $\check{T}$ both have to be properly increased. Modeling a $\bar{\mathcal{H}}$-structured schedule can be done quite naturally: We simply redefine the size $\Lambda(M)$ of a module $M = (s, q) \in \mathcal{M}$ to be $\overline{(s+q)}$. With this definition, we have $|H| = |\bar{\mathcal{H}}| = \mathcal{O}(1/\varepsilon \log 1/\varepsilon)$ yielding an improved running time for solving the MCIP of:

$$2^{\mathcal{O}(1/\varepsilon^2 \log^3 1/\varepsilon)} n \log^2(m) \log^6(n)$$

Combining this with the results above and the considerations in Section 6.4 yields the running time claimed below Theorem 6.1.

SETUP CLASS MODEL. In the setup class model, an analogue approach also yields a reduced set of module sizes, that is, $|H| = \mathcal{O}(1/\varepsilon \log 1/\varepsilon)$. Therefore, the MCIP can be solved in time:

$$2^{\mathcal{O}(1/\varepsilon^3 \log^4 1/\varepsilon)} K \log^2(n) \log^5(K)$$

Hence, we get the running time claimed beneath Theorem 6.1.

6.6 OPEN PROBLEMS

For further research, one of the immediate questions is whether improved running times for the considered problems, in particular for the preemptive model, can be achieved. Concerning the preemptive model, it is also unclear whether the generalization in which the jobs are partitioned into classes and the setup time has to be paid per class admits a PTAS.

Furthermore, the MCIP could be adapted to other problems. A first step in this direction was recently taken [75] for variants of machine scheduling with class constraints in which the jobs are partitioned into classes and only a certain number of distinct classes is allowed to be present on each machine.

From a broader perspective, it would be interesting to further study the potential of new algorithmic approaches in integer programming for approximation, and, on the other hand, further study the respective techniques themselves.

7

UNIFORM SCHEDULING WITH SETUP TIMES

7.1 INTRODUCTION

In this chapter, we consider the setup class model studied in the last chapter in the context of uniform machines. More precisely, we are given a set $\mathcal{J}$ of n jobs and a set $\mathcal{M}$ of m machines. The set of jobs is partitioned into K classes. Each job j has a size p_j and belongs to exactly one of the classes $k_j \in [K]$; each class $k \in [K]$ has a setup size s_k; and each machine i has a speed v_i. The processing time of a job j or the setup time of a class k on a machine i are given by p_j/v_i or s_k/v_i, respectively. The goal is to find a schedule $\sigma : \mathcal{J} \to \mathcal{M}$ such that the makespan

$$C_{\max} = \max_{i \in \mathcal{M}} \sum_{j \in \sigma^{-1}(i)} p_j/v_i + \sum_{k \in \{k_j \mid j \in \sigma^{-1}(i)\}} s_k/v_i$$

is minimized. Note that for a job j or a setup class k, we call the values p_j and s_k the job or setup *size*, respectively, in distinction from their processing *time* $p_{ij} = p_j/v_i$ or setup *time* s_k/v_i on a given machine i.

We present a PTAS for the above problem in Section 7.2 as well as a short discussion of open problems in Section 7.3.

RELATED WORK. The first PTAS for uniform scheduling is due to Hochbaum and Shmoys [66] and the first EPTAS was presented by Jansen [72]. For a detailed literature review concerning scheduling with setup times we refer to the last chapter and Section 6.1 in particular. To the best of our knowledge, the problem variant studied in this chapter has not been considered before.

7.2 PTAS

As usual, we utilize the dual approximation framework (see Section 2.2) and therefore assume that a guess T of the optimal makespan is given. Note that there is a simple constant-factor approximation for this problem [79] based on the LPT rule (see Chapter 1) which can be used for the dual approximation. The roadmap for the PTAS is as follows:

1. Simplify the instance.
2. Find a so-called *relaxed* schedule for the simplified instance via dynamic programming, or conclude correctly that no schedule with makespan T for the original instance exists.

3. Construct a regular schedule for the simplified instance using the relaxed schedule and a greedy procedure.

4. Construct a schedule for the original instance using the one for the simplified instance.

Concerning the second and third step, first note that the makespan guess T given by the dual approximation framework enables a packing perspective on the problem: On machine i there is an amount of Tv_i free space and the jobs and setup times have to be placed into this free space. Now, a job or setup size may be big or small relative to this free space, say bigger or smaller than εTv_i. In the latter case, i can receive one additional job or setup in a PTAS, or several for another notion of big and small. Hence, we have to be cautious when placing big objects but can treat small objects with less care. We utilize this and are able to show that it is sufficient to search for a so-called *relaxed schedule*. Roughly speaking, in a relaxed schedule some jobs and setups are fractionally placed on machines for which they are small, and for jobs that are big relative to the setup time of their class, the setup is ignored.

We search for a relaxed schedule via a dynamic program. For the dynamic program, we define intervals of machine speeds, called groups, and the groups are considered one after another ordered by speeds and starting with the slowest. In each group, the speeds differ at most by a constant factor. This enables us to employ ideas for the identical machine case developed in [74] for each group. However, there has to be some information passed up from one group to the next, and this has to be properly bounded in order to bound the running time of the dynamic program. While we can use some standard ideas for classical makespan minimization on uniformly related machines (without setup times), e.g., from [66], there are problems arising from the setup classes. Mainly, we have to avoid passing on class information between the groups. As a crucial step to overcome this problem, we show that for each group there is only a bounded interval of machine speeds for which we have to properly place the setup times. In the algorithm, we define the groups wide enough and with overlap such that for each class there is a group containing the whole interval relevant for this class. When going from one group to the next, we therefore do not have to pass on class information of jobs that have not been scheduled yet. This, together with proper simplification steps enables us to properly bound the running time of the dynamic program.

In the following, we describe the PTAS in detail, starting with the simplification steps, followed by some definitions and observations that lead to the definition of a relaxed schedule. Next, we present the dynamic program, and, lastly, argue that it can indeed be used to find a relaxed schedule.

Throughout this section $\varepsilon > 0$ denotes the accuracy parameter of the PTAS and we require $1/\varepsilon \in \mathbb{Z}_{\geqslant 2}$.

SIMPLIFICATION STEPS. We perform a series of simplification steps: First, we establish minimum sizes of the occurring speeds, job and setup sizes; next, we ensure that the job sizes of a class are not much smaller than its setup size; and lastly, we round the speeds, job and setup sizes. Most of the used techniques, like geometric rounding or the replacement of small objects with placeholders with a minimum size, can be considered folklore in the design of approximation algorithms for scheduling problems. Similar arguments can be found, e.g., in [47, 66, 72, 74] or the previous chapters.

Let I be the original instance and $v_{max} = \max\{v_i \mid i \in \mathcal{M}\}$. We remove all machines with speeds smaller than $\varepsilon v_{max}/m$ and denote the smallest remaining speed after this step by v_{min}. Furthermore, we increase all job and setup sizes that are smaller than $\varepsilon v_{min} T/(n+K)$ to this value, and call the resulting instance I_1. By scaling, we assume $v_{min} T = 1$ in the following.

LEMMA 7.1. *If there is a schedule with makespan T for I, then there is also a schedule with makespan $(1+\varepsilon)^2 T$ for I_1; and if there is a schedule with makespan T' for I_1, then there is also a schedule with makespan T' for I.*

Proof. Given a schedule for I, the summed up load on machines missing in I_1 is upper bounded by $\varepsilon v_{max} T$ and we can place it on a machine with speed v_{max}. Furthermore, increasing the setup and processing sizes can increase the load on any machine by at most $\varepsilon v_{min} T$. □

The next step is to make sure that jobs are not much smaller than the setup size of their class. Let I_2 be the instance we get by replacing for each class k the jobs with a size of at most εs_k with placeholders, that is, we remove the jobs from $\mathcal{J}'_k = \{j \in \mathcal{J}_k \mid p_j \leqslant \varepsilon s_k\}$ and introduce $\lceil (\sum_{j \in \mathcal{J}'_k} p_j)/(\varepsilon s_k) \rceil$ many jobs of size εs_k belonging to class k.

LEMMA 7.2. *If there is a schedule with makespan T' for I_1, then there is also one with makespan $(1+\varepsilon)T'$ for I_2; and if there is a schedule with makespan T' for I_2, then there is also one with makespan $(1+\varepsilon)T'$ for I_2.*

Proof. Given a schedule for one of the instances, we can greedily replace jobs with the respective placeholders and vice-versa, overpacking with at most one object per class and machine. Thereby the overall load on each machine due to a class scheduled on the machine is increased at most by a factor of $(1+\varepsilon)$. □

Next, we perform rounding steps for the job and setup sizes as well as the machine speeds: For each job or setup size x, let $e(x) = \lfloor \log x \rfloor$. We round x to $2^{e(x)} + y\varepsilon 2^{e(x)}$ with $y = \lceil (x - 2^{e(x)})/(\varepsilon 2^{e(x)}) \rceil$. This rounding approach is due to Gálvez et al. [48]. Furthermore, we perform geometric rounding for machine speeds, that is, each machine speed v is rounded down to $(1+\varepsilon)^z v_{min}$ with $z = \lfloor \log_{1+\varepsilon}(v_i/v_{min}) \rfloor$. We call the rounded instance I_3.

LEMMA 7.3. *If there is a schedule with makespan T' for I_2, then there is also a schedule with makespan $(1+\varepsilon)^2 T'$ for I_3; and if there is a schedule with makespan T' for I_3, then there is also one for I_2.*

Proof. Each job and setup size is increased at most by a factor of $(1+\varepsilon)$ by the rounding and each machine speed is decreased at most by a factor of $(1+\varepsilon)$. □

Hence, if there is a schedule with makespan at most T for I, then there is also a schedule with makespan at most T_1 for I_3 with $T_1 = (1+\varepsilon)^5 T = (1+\mathcal{O}(\varepsilon))T$. Furthermore, if we should find a schedule with makespan T_2 for I_3 with $T_1 \leqslant T_2 = (1+\mathcal{O}(\varepsilon))T$, we can transform it back into a schedule for the original instance with makespan at most $T_3 = (1+\varepsilon)T_2 = (1+\mathcal{O}(\varepsilon))T$.

For the sake of simplicity, we assume in the following that the instance I is already simplified and the makespan bound T was properly increased. Summing up, we have the following properties:

(P1) $v_{\min} \geqslant \varepsilon v_{\max}/m$.

(P2) $v_{\min} T = 1$.

(P3) For each job or setup size x, we have $x \geqslant \varepsilon/(n+K)$.

(P4) For a job j of class k, we have $p_j \geqslant \varepsilon s_k$.

(P5) Each job or setup size is of the form $2^e + z\varepsilon 2^e$, with $e \in \mathbb{Z}$ and $z \in \{0, \ldots, 1/\varepsilon - 1\}$.

(P6) Each machine speed is of the form $(1+\varepsilon)^z v_{\min}$, with $z \in \mathbb{Z}_{\geqslant 0}$.

PRELIMINARIES. We define two threshold parameters $\delta = \varepsilon^2$ and $\gamma = \varepsilon^3$. For each class k, the *core jobs* belonging to that class are the ones with a job size p such that $\varepsilon s_k \leqslant p < s_k/\delta$. Bigger jobs are called *fringe jobs*. The set of core or fringe jobs of class k is denoted by $\bar{J}_k$ and $\tilde{J}_k$, respectively. The *core machines* i of class k, are the ones with $s_k \leqslant T v_i < s_k/\gamma$ and faster machines are called *fringe machines*. We have:

Remark 7.4. For each class k and each job j that belongs to k, j is either a core or a fringe job and has to be scheduled either on a core or a fringe machine of k.

To see this, note that due to (P4), there are no jobs smaller than the core jobs for each class, and machines slower than the core machines of a class k would be overpacked by the setup time s_k.

A job size p is called *small* for a speed v if $p < \varepsilon v T$, *big* if $\varepsilon v T \leqslant p \leqslant v T$, and *huge* if $p > v T$. We use these terms for jobs and machines as well, e.g., we call a job j small for machine i, if $p_j < \varepsilon v_i T$. Since $\gamma/\delta = \varepsilon$ holds, we have:

Remark 7.5. The core jobs of class k are small on fringe machines of k.

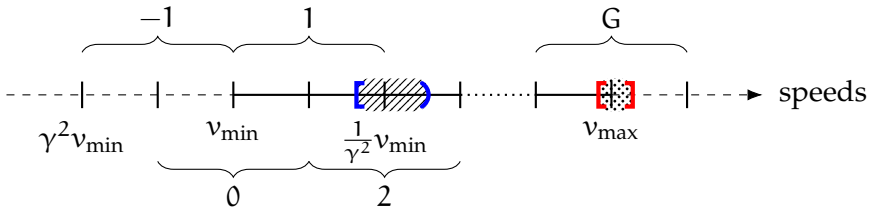


Figure 7.1: Machine speeds with logarithmic scale. The braces mark groups, the dashed interval possible speeds of core machines of some class with core group 2, and the dotted interval possible speeds of machines where some job with native group G is big.

Next, we define speed groups (see Figure 7.1). For each $g \in \mathbb{Z}$, we set $\check{v}_g = v_{\min}/\gamma^{g-1}$ and $\hat{v}_g = v_{\min}/\gamma^{g+1}$. Group g is given by the interval $[\check{v}_g, \hat{v}_g)$. Note that the groups are overlapping with each speed occurring in exactly two groups. A machine i belongs to group g if $v_i \in [\check{v}_g, \hat{v}_g)$, and we denote the set of machines belonging to g by M_g and the set of corresponding speeds by V_g, i.e., $V_g = \{v_i \mid i \in M_g\}$. By definition, the smallest group g with $M_g \neq \emptyset$ is group 0. Furthermore, let G be the biggest number with this property. Because of (P1), we have $G \leqslant m/(3\varepsilon \log(1/\varepsilon)) = \mathcal{O}(m/\varepsilon)$.

For each job j, there are up to three (succeeding) groups containing speeds for which its size is big, and at least one of them contains all such speeds. Let g be the smallest group with this property, i.e., $p_j \geqslant \varepsilon \check{v}_g T$ and $p_j < \hat{v}_g T$. We call g the *native group* of j. For a group g, the fringe jobs with native group g will be of interest in the following, and we denote the set of these jobs by $\tilde{J}_g$.

Moreover, for each class k there are at most three (succeeding) groups containing possible speeds of core machines of k, and there is at least one that contains all of them. Let g be the smallest group with this property, i.e., $s_k \geqslant \check{v}_g T$ and $s_k/\gamma < \hat{v}_g T$. We say that g is the *core group* of k. Note that k has a core group even if it has no core machines. In Figure 7.1, the groups together with native groups of jobs and core groups of classes are visualized.

Remark 7.6. Let j be a core job of class k and g be the core group of k. There is a speed v in group g such that p_j is big for v.

We have $p_j < s_k/\varepsilon^2$ because j is a core job, and we have $s_k/\varepsilon^2 < \varepsilon \hat{v}_g T$ because g is the core group of k. Hence, p_j is small for $\hat{v}_g$. Furthermore, we have $p_j \geqslant \varepsilon s_k \geqslant \varepsilon \check{v}_g T$ for the same reasons. Therefore, p_j is big or huge for $\check{v}_g$ and there lies at least one speed in between for which it is big.

RELAXED SCHEDULE. In a *relaxed schedule*, the set of jobs is partitioned into integral jobs $\mathcal{I}$ and fractional jobs $\mathcal{F}$, and an assignment $\sigma' : \mathcal{I} \to \mathcal{M}$ of the integral jobs is given. For each $j \in \mathcal{I}$, the machine $\sigma'(j)$ belongs to the native group of j if j is a fringe job, and to the

core group of k if j is a core job of class k. Setups for fringe jobs are ignored, and hence we define the relaxed load L_i of machine i to be:

$$L_i = \sum_{j \in \sigma'^{-1}(i)} p_j + \sum_{k : \sigma'^{-1}(i) \cap \check{J}_k \neq \emptyset} s_k$$

Intuitively, the fractional jobs are placed fractionally together with some minimum amount of setup in the left-over space on the machines that are faster than the ones in their respective native or core group. More formally, we say that the relaxed schedule has makespan T if $L_i \leqslant T v_i$ for each $i \in \mathcal{M}$ and the following space condition for the fractional jobs holds:

Let $\mathcal{F}_g$ be the set of fractional fringe jobs with native group g and fractional core jobs of class k with core group g. Moreover, let $A_i = T v_i - L_i$ the remaining free space on machine i with respect to T. Lastly, let W_g be the overall load of jobs from $\mathcal{F}_g$ together with one setup for each class that has core group g, has no fringe job, and has a fractional core job, that is:

$$W_g = \sum_{j \in \mathcal{F}_g} p_j + \sum_{k : \mathcal{F}_g \cap \check{J}_k \neq \emptyset, \hat{J}_k = \emptyset} s_k \tag{7.1}$$

A job $j \in \mathcal{F}_g$ should be placed on a machine that belongs to group $g+2$ or a faster group (note that the machines belonging to group $g+1$ are covered by group g and $g+2$). Hence, we set the reduced accumulated fractional load R_g for group g to be:

$$R_g = \max\left\{0, R_{g-1} + W_{g-2} - \sum_{i \in M_g \setminus M_{g+1}} A_i\right\}$$

Note that since $M_g = \emptyset$ for $g < 0$, we have $R_g = \sum_{g' \leqslant g-2} W_{g'}$. The required space condition is $R_G = W_G = W_{G-1} = 0$.

LEMMA 7.7. *If there is a schedule with makespan T for a given instance, then there is also a relaxed schedule with makespan T; and if there is a relaxed schedule with makespan T, then there is a schedule with makespan $(1 + O(\varepsilon))T$.*

Proof. The first claim is easy to see: For a given schedule σ with makespan T, the fringe jobs assigned to a machine of their native group and the core jobs assigned to the core group of their class form the set $\mathcal{I}$ and we can set $\sigma' = \sigma|_{\mathcal{I}}$. The remaining jobs form the fractional jobs and they obviously fit fractionally into the left-over space, because we have a fitting integral assignment of them. This also holds for the setups for groups with fractional jobs and no fringe jobs: There has to be at least one setup for each such class on a machine that does not belong to their core group. Dropping the setups of the fringe jobs only increases the free space further.

We consider the second claim. Let $(\mathcal{I}, \mathcal{F}, \sigma')$ be a relaxed schedule with makespan T. We construct a regular schedule and start by placing

all the integral jobs like in the relaxed schedule. To place the fractional jobs, we consider one speed group after another starting with group 0. For the current group g, we consider the jobs from $\mathcal{F}' \subset \mathcal{F}$, with $\mathcal{F}' = \mathcal{F}_{g-2}$, if $g > 0$, and $\mathcal{F}' = \bigcup_{g' \leqslant -2} \mathcal{F}_{g'}$, if $g = 0$. We partition $\mathcal{F}'$ into three sets F_1, F_2, F_3 that are treated differently. The sets are defined as follows:

- F_1 includes core jobs of classes k whose fractional core jobs have an overall size of at most s_k/ε, i.e., $\sum_{j \in \mathcal{F}' \cap J_k} p_j \leqslant s_k/\varepsilon$, and that do have a fringe job.
- F_2 includes core jobs of classes k with $\sum_{j \in \mathcal{F}' \cap J_k} p_j \leqslant s_k/\varepsilon$, and that do *not* have a fringe job.
- F_3 includes the fringe jobs in $\mathcal{F}'$ as well as the core jobs of classes k with $\sum_{j \in \mathcal{F}' \cap J_k} p_j > s_k/\varepsilon$.

Let k be a class whose fractional core jobs are included in F_1 or F_2, that is, $\sum_{j \in \mathcal{F}' \cap J_k} p_j \leqslant s_k/\varepsilon$. We will place the fractional core jobs of k all on the same machine. If the jobs are included in F_1, there exists a fringe job with class k and we can place the fractional core jobs together with such a job. A fringe job of class k has a size of at least $s_k/\delta = s_k/\varepsilon^2$, and hence the load due to the fringe job is increased at most by a factor of $(1+\varepsilon)$ by this step. This can happen at most once for each class and hence at most once for each fringe job. Since all fringe jobs of the class could be fractional, we postpone this step until all the remaining fractional jobs are placed. If, on the other hand, the fractional core jobs of class k are included in F_2, we construct a container that is filled with all the corresponding jobs together with one setup of the class. Note that the setup is already accounted for in the relaxed schedule and that the overall size of the container is upper bounded by $(1 + 1/\varepsilon)s_k$. We call a container small on a machine i if its size is upper bounded by $\varepsilon \nu_i T$. Each machine i belonging to group g or faster groups is a fringe machine of class k, and therefore we have $s_k \leqslant \gamma \nu_i T$. Hence, the size of the container is at most $(\varepsilon^2 + \varepsilon^3)\nu_i T \leqslant \varepsilon \nu_i T$ (because $\varepsilon \leqslant 1/2$), i.e., the container is small on i. We place the container in the next step.

Next, we construct a sequence of jobs and containers and apply a greedy procedure to place them. We start with an empty sequence and add all containers from the last step and all fringe jobs from F_3 in any order. The core jobs from F_3 are added sorted by classes in the end of the sequence. If there is a residual sequence that was not placed in the last iteration, we concatenate the two with the old sequence in the front. We now consider each of the machines $i \in M_g \setminus M_{g+1}$ with $L_i < \nu_i T$ in turn and repeatedly remove the first job from the sequence and insert it on the current machine until the load of the machine exceeds $\nu_i T$. Since all jobs and containers in the sequence are small on the machines of group g, they are overloaded at most

by factor of $(1+\varepsilon)$ afterwards. For each step, the overall size of jobs and containers that are left in the sequence is at most the reduced accumulated fractional load R_g, because the remaining free space on the machines has either been filled completely, or the sequence is empty. Since $R_G = W_G = W_{G-1} = 0$, all jobs and containers can be placed eventually.

Now, all jobs are properly placed, but some setups are still missing. First, we consider core jobs that have been inserted in the greedy procedure and were not included in a container. If the overall size of such core jobs of a class k placed on a machine is at least s_k/ε, adding the missing setups increases this size at most by a factor of $(1+\varepsilon)$. However, for each machine i, there can be up to two classes k without this property, namely the class that has been added first and the class that has been added last on the machine. For each class in between, all core jobs of this class in the sequence have been added to the machine, and these have sufficient overall size by construction. Furthermore, if a job of class k was placed on a machine i in the greedy procedure, i is a fringe machine of k. Hence, the load of each machine i after this step can be bounded by $(1+\varepsilon)^2 \nu_i T + 2\varepsilon^3 \nu_i T \leqslant (1+\varepsilon)^3 \nu_i T$. Lastly, we add the missing setups for the fringe jobs, resulting in an additional increase of at most $(1+\varepsilon^2)$, because a fringe job of class k has a size of at least s_k/ε^2. □

DYNAMIC PROGRAM. We use a dynamic programming approach to compute a relaxed schedule with makespan T or correctly decide that there is none. Therein, the groups of machine speeds are considered one after another starting with the slowest and going up. For a fixed group, the dynamic program can be seen as an adaptation of the one from [74] for the identical case, and the overall structure of the program is similar to approaches used for the classical problem without setup times, e.g., in [66] and [47]. However, there is some work to be done to combine these approaches and to deal with the fact that the speed groups are overlapping. In order to define the dynamic program and bound its running time, we first need some additional considerations and definitions.

Let B_g be the number of job sizes in I that are big for at least one speed of group g. This implies $\varepsilon \check{\nu}_g T \leqslant p \leqslant \hat{\nu}_g T$. We set $e(g) = \lfloor \log \varepsilon \check{\nu}_g T \rfloor$. Because of the rounding of the job sizes (see (P5)), we have $2^e + z\varepsilon 2^e$ for each size $p \in B_g$ with $e = \lfloor \log p \rfloor$ and $z \in \{0, \dots, 1/\varepsilon - 1\}$. Remember that we have $1/\varepsilon \in \mathbb{N}$. Therefore, p is an integer multiple of $\varepsilon 2^{e(g)}$. Furthermore, we have:

$$2^{e(g)} \leqslant \varepsilon \check{\nu}_g T \leqslant p \leqslant \hat{\nu}_g T = \gamma^{-2} \check{\nu}_g T \leqslant \varepsilon^{-1} \gamma^{-2} 2^{e(g)+1}$$

Hence, $|B_g| \leqslant 2/(\varepsilon^2 \gamma^2) = \mathcal{O}(1/\varepsilon^8)$.

We define a superset L_g of possible load values that can occur on a machine belonging to group g in a relaxed schedule due to integral

jobs. Such a machine may receive fringe jobs with native group $g-1$, g or $g+1$, and core jobs whose core group is one of these, as well as setups belonging to the latter jobs. The setup sizes have been rounded like the job sizes (see (P5)) and for each of the mentioned setup sizes s we have $s \geqslant \check{v}_{g-1}T$ and hence s is an integer multiple of $\varepsilon 2^{e(g-1)}$. The occurring loads are smaller than $\hat{v}_g T$ and we have $\hat{v}_g T = \varepsilon^{-1}\gamma^{-3}\check{v}_{g-1}T2^{e(g)+1}$. Hence, we set

$$L_g = \left\{ z\varepsilon 2^{e(g-1)} \,\middle|\, z \in \left\{0, 1, \ldots, 2/(\varepsilon^2\gamma^3)\right\} \right\}$$

and have $|L_g| = 2/(\varepsilon^2\gamma^3) + 1 = \mathcal{O}(1/\varepsilon^{11})$.

Next, we define a superset Λ of possible load values of fractional jobs and corresponding setup sizes in a relaxed schedule. Because of (P3), (P2), and (P1), each job and setup size is lower bounded by $\varepsilon/(n+K)$ and $v_{\min} \geqslant \varepsilon v_{\max}/m$. We set $e^* = \lfloor \log \varepsilon/(n+K) \rfloor$. Because of the rounding (see (P5)), each job and setup size is an integer multiple of $\varepsilon 2^{e^*}$. Furthermore, the overall load of all jobs together with one setup of each class without a fringe job can be bounded by $mv_{\max}T \leqslant m^2/\varepsilon$, or, more precisely, if this is not the case we can reject the current guess of the makespan. Hence, we can set $\Lambda = \{k\varepsilon 2^{e^*} \mid k \in \{0, 1, \ldots, 2m^2(n+K)/\varepsilon^3\}\}$, and get $|\Lambda| = \mathcal{O}(m^2(n+K)/\varepsilon^3)$.

Lastly, we bound the number of speeds $|V_g|$ that occur in group g. We have $\hat{v}_g = \check{v}_g/\gamma^2$ and applied geometric rounding on the speeds (see (P6)). Hence, $|V_g| = \mathcal{O}(\log_{1+\varepsilon}(1/\gamma^2)) = \mathcal{O}(1/\varepsilon \log 1/\varepsilon)$ (because $\varepsilon < 1$).

A state of the dynamic program is of the form

$$(g, k, \iota, \xi, \mu, \lambda)$$

with:

- $g \in \{0, \ldots, G\}$ is a group index.
- $k \in \{0, \ldots, K\}$ is a setup class index including a dummy class 0. The dummy class is included to deal with the fringe jobs with native group g.
- $\iota : B_g \to \{0, \ldots, n\}$ is a function mapping job sizes to multiplicities. Intuitively, $\iota(p)$ jobs of size p corresponding to the current class still have to be dealt with in the current group.
- $\xi \in \{0, 1\}$ is a flag that encodes whether a core job of the current class has been scheduled as a fractional job. Since this does not apply to fringe jobs, we essentially ignore the states with $k = 0$ and $\xi = 1$ in the following.
- $\mu : V_g \times L_g \times \{0, 1\} \to \{0, \ldots, m\}$ is a function mapping triples of machine speeds, load values, and flags to machine multiplicities. Intuitively, we have $\mu(v, \ell, \zeta)$ machines of speed v and with load

ℓ in the current machine group that already received the setup of the current class ($\zeta = 1$) or not ($\zeta = 0$). For fringe jobs, the setup is ignored in a relaxed schedule. We model this by essentially ignoring the states with $k = 0$ and μ with $\mu(\nu, \ell, 0) > 0$ for some $\nu \in V_g$ and $\ell \in L_g$ in the following. The intuition behind this is that we treat the machines as if they already received a setup for the fringe jobs.

- $\lambda \in \Lambda^3$ is a load vector. Its values λ_i corresponds to the load of fractional jobs together with the corresponding setups that have been pushed up to faster groups for the current ($i = 1$) or last group ($i = 2$), or the corresponding load for all previous groups offset against the remaining space on machines present only in slower groups ($i = 3$).

Let $\mathcal{S}$ be the set of states of the dynamic program. Because of the above considerations, we have:

$$|\mathcal{S}| = \mathcal{O}(GKn^{\max_g |B_g|} m^{\max_g 2|V_g||L_g|} |\Lambda^3|) = (nmK)^{\text{poly}(1/\varepsilon)}$$

Now, the idea is that the states form the vertices of a graph, and the relaxed schedules correspond to paths from a start to an end state. Some of the states do not make sense in this context, like e.g., the ones with $k = 0$ and $\xi = 1$, but we will make sure that these states cannot be reached. There are three types of edges:

(1.) The edges corresponding to scheduling decisions of the single jobs: For each $(g, k, \iota, \xi, \mu, \lambda)$ with $\iota \neq 0$ there are up to $2|V_g||L_g| + 1$ edges corresponding to the choices of scheduling some job on a machine with a certain speed and load that already received a setup or not, or treating the job as fractional. Let $p \in B_g$ be the biggest size with $\iota(p) > 0$. We define ι' as the function we get by decrementing $\iota(p)$. For each speed $\nu \in V_g$ and each load $\ell \in L_g$, we add up to two edges: If $\mu(\nu, \ell, 0) > 0$, $k > 0$, and $\ell + p + s_k \leqslant \nu T$; we add an edge to the state $(g, k, \iota', \xi, \mu', \lambda)$, where μ' is the function we get by decrementing $\mu(\nu, \ell, 0)$ and incrementing $\mu(\nu, \ell + p + s_k, 1)$. If $\mu(\nu, \ell, 1) > 0$ and $\ell + p \leqslant \nu T$, we add an edge to the state $(g, k, \iota', \xi, \mu'', \lambda)$, where μ'' is the function we get by decrementing $\mu(\nu, \ell, 1)$ and incrementing $\mu(\nu, \ell + p, 1)$. These two edges correspond to the decisions of scheduling a job with size p on a machine with speed ν and load ℓ that previously already had a setup for the current group or not (the latter case also covers fringe jobs, i.e., $k = 0$). Lastly, we add one edge to the state $(g, k, \iota', \xi', \mu, \lambda')$ with $\lambda'_2 = \lambda_2$ and $\lambda'_3 = \lambda_3$. If $k > 0$, k has no fringe job, and $\xi = 0$; we have $\xi' = 1$ and $\lambda'_1 = \lambda_1 + p + s_k$. Otherwise, $\xi' = \xi$ and $\lambda'_1 = \lambda_1 + p$. This edge corresponds to the decision of treating a job of size p and class k as a fractional job.

(2.) The edges marking the transition from one class to another: For each state $(g, k, \iota, \xi, \mu, \lambda) \in \mathcal{S}$ with $k < K$ and $\iota = 0$, there is an edge connecting the state with $(g, k+1, \iota', 0, \mu', \lambda)$, where ι' and μ' are defined as follows. If g is the core group of k, for each $p \in B_g$, the value $\iota'(p)$ is the number of core jobs of class k and size p, i.e., $\iota'(p) = |\{j \in \bar{J}_k \mid p_j = p\}|$. Otherwise, we set $\iota'(p) = 0$ for each $p \in B_g$. Furthermore, we have $\mu'(\nu, \ell, 0) = \mu(\nu, \ell, 0) + \mu(\nu, \ell, 1)$ and $\mu'(\nu, \ell, 1) = 0$ for each $\nu \in V_g$ and $\ell \in L_g$. If g is not the core group of $k+1$, the class is essentially skipped ($\iota' = 0$). Otherwise, these edges model that so far no core job of this class has been treated as fractional, no job of this class has been scheduled, and no machine has received a setup of this class yet.

(3.) The edges marking the transition from a group g to the next: For each state $(g, k, \iota, \xi, \mu, \lambda) \in \mathcal{S}$ with $g < G$, $k = K$, $\iota = 0$, and $\mu(\nu, \ell, \zeta) = 0$ for each $\ell \in L_g \setminus L_{g+1}$, there is an edge connecting the state with $(g+1, 0, \iota', 0, \mu', \lambda')$, where ι', μ' and λ' are defined as follows. For each $p \in B_{g+1}$ the value $\iota'(p)$ is the number of fringe jobs with native group g and size p, i.e., $\iota'(p) = |\{j \in \tilde{J}_g \mid p_j = p\}|$. We have $\lambda'_1 = 0$, $\lambda'_2 = \lambda_1$, and $\lambda'_3 = \lambda_2 + x$ with:

$$x = \max\Big\{0, \lambda_3 - \sum_{\nu \in V_g \cap V_{g-1}} \sum_{\ell \in L_g} (T\nu - \ell) \cdot (\mu(\nu, \ell, 0) + \mu(\nu, \ell, 1))\Big\}$$

At this point, we offset the remaining free space on the machines belonging to group g but not to group $g+1$ with the accumulated fractional load. Furthermore, $\mu'(\nu, \ell, \zeta)$ is given by $\mu(\nu, \ell, 0) + \mu(\nu, \ell, 1)$ if $\nu \in V_g \cap V_{g+1}$, $\ell \in L_g$, and $\zeta = 1$; by $|\{i \in M_g \mid \nu_i = \nu\}|$ if $\nu \in V_{g+1} \setminus V_g$, $\ell = 0$ and $\zeta = 1$; and by 0 otherwise. Hence, we carry over the loads of the machines present in both sets and consider the machines belonging to the new group but not to the old to be empty. Note that only triples with $\zeta = 1$ have values bigger than zero. This corresponds to the convention that we treat machines as if they already received a setup for the fringe jobs.

The start state of the dynamic program has the form $(0, 0, \iota, 0, \mu, \lambda)$, with ι, μ, and λ defined as follows. For each $p \in B_0$, the value $\iota(p)$ is the number of fringe jobs with native group 0 and size p; and for each speed $\nu \in V_0$, the value $\mu(\nu, 0, 1)$ is the number of machines with speed ν. Otherwise, we have $\mu(\nu, \ell, \zeta) = 0$. This can be understood as follows. In the beginning, no machine has received any load, and we start scheduling the fringe jobs with native group 0. Furthermore, we define $\lambda_1 = 0$, $\lambda_2 = W_{-1}$ and $\lambda_3 = \sum_{g \leqslant -2} W_g = R_{-1} + W_{g-2}$ (see Equation (7.1) for the definition of W_g). Jobs with core groups $g < 0$ do not have core machines and hence should all be treated as fractional. This motivates the latter definitions.

The end states have the form $(G, K, 0, 0, \mu', \lambda')$, where μ' and λ' have the following form. For each $v \in V_G$, we have $\mu'(v, \ell, \zeta) = 0$ if $\ell > vT$ and $\sum_{\ell \in L_G} \sum_{\zeta \in \{0,1\}} \mu'(v, \ell, \zeta) = |\{i \in M_G \mid v_i = v\}|$, that is, we have the correct number of machines of speed v and no machine is overloaded. Furthermore, $\lambda'_1 = \lambda'_2 = 0$, and $\lambda'_3 \leqslant \sum_{v \in V_G \setminus V_{G-1}} \sum_{\ell \in L_G} (Tv - \ell) \cdot (\mu(v, \ell, 0) + \mu(v, \ell, 1))$. This corresponds to the property $R_G = W_G = W_{G-1} = 0$ of a relaxed schedule with makespan T.

In the dynamic program, we search for a path from the start to an end state or correctly decide that there is no such path. This boils down to a reachability problem in a simple directed graph with $(nmK)^{\text{poly}(1/\varepsilon)}$ vertices and hence can be done in polynomial time in the input length.

CORRECTNESS. Next, we argue that there is a one to one correspondence between such paths and relaxed schedules with makespan T except for permutations of jobs and machines that are equivalent with respect to such a schedule, that is, machines with the same speed, fringe jobs with the same size, and core jobs of the same class and with the same size.

First note that a path from the start to an end state has to be of a certain form. Let $g \in \{0, \dots, G\}$ and $k \in \{0, \dots, K\}$. We define $S(g, k) \subseteq S$ as the set of states s with $s_1 = g$ and $s_2 = k$. Let

$$\iota_1^{(g,k)} : B_g \to \{0, \dots, n\}$$

be the function with

- $\iota_1^{(g,k)}(p) = |\{j \in \tilde{J}_g \mid p_j = p\}|$ for each $p \in B_g$ if $k = 0$,
- $\iota_1^{(g,k)}(p) = |\{j \in \bar{J}_k \mid p_j = p\}|$ for each $p \in B_g$ if $k > 0$ and g is the core group of class k, and
- $\iota_1^{(g,k)}(p) = 0$ for each $p \in B_g$ otherwise.

Moreover, let $r(g, k) = 1 + \sum_{p \in B_g} \iota_{g,k}^{(1)}(p)$. For each $1 < \ell \leqslant r(g, k)$, let $\iota_\ell^{(g,k)}$ be equal to $\iota_{\ell-1}^{(g,k)}$ except that the value corresponding to the largest job size p with $\iota_{\ell-1}^{(g,k)}(p) > 0$ has been decremented. This implies that $\iota_{r(g,k)}^{(g,k)}$ is the function mapping everything to 0. Let $s \in S(g, k)$. Due to the definitions of the edges, we have:

- If s has an incoming edge from a state $s' \in S(g', k') \neq S(g, k)$, then we have $s_3 = \iota_1^{(g,k)}$; $s'_3 = \iota_{r(g',k')}^{(g',k')}$; $g' = g$, $k' = k - 1$, and the edge is of type (2.) if $k > 0$; as well as $g' = g - 1$, $k' = K$, and the edge is of type (3.) if $k = 0$.
- If s has an incoming edge from a state $s' \in S(g, k)$ and $s_3 = \iota_\ell^{(g,k)}$, then we have $\ell > 1$, $s'_3 = \iota_{\ell-1}^{(g,k)}$, and the edge is of type (1.).

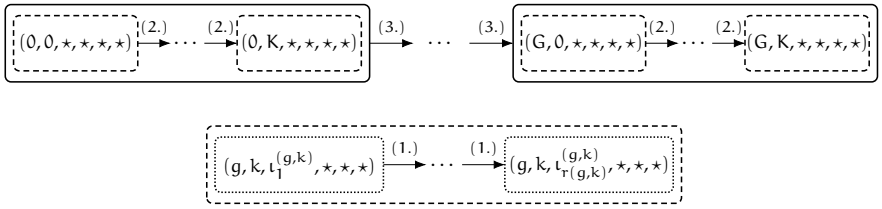


Figure 7.2: The structure of a path from the start to an end state. The outer rectangles above represent sets of states that have the same first value, and the dashed and dotted rectangles sets of states that share the first two or three values, respectively. The picture below illustrates the inner structure of each dashed set. The path includes exactly one state from each dotted set.

- If s has an outgoing edge to a state $s' \in \mathcal{S}(g', k') \neq \mathcal{S}(g, k)$, then we have $s_3 = \iota_{r(g,k)}^{(g,k)}$; $s'_3 = \iota_1^{(g',k')}$; $g' = g$, $k' = k+1$, and the edge is of type (2.) if $k < K$; as well as $g = g+1$, $k = 1$, and the edge is of type (3.) if $k = K$.
- If s has an outgoing edge to a state $s' \in \mathcal{S}(g, k)$ and $s_3 = \iota_\ell^{(g,k)}$, then we have $\ell < r(g, k)$, $s'_3 = \iota_{\ell+1}^{(g,k)}$, and the edge is of type (1.).

Let α be the start, and ω be an end state. Note that $\alpha \in \mathcal{S}(0, 0)$, $\alpha_3 = \iota_1^{(0,0)}$, $\omega \in \mathcal{S}(G, K)$, and $\omega_3 = \iota_{r(G,K)}^{(G,K)}$. This already determines the first three values on any path from the start to an end state and this is visualized in Figure 7.2.

Based on this structure, it is not very hard to verify the correspondence between paths from the start to an end state and relaxed schedules with makespan T for the given instance and how a relaxed schedule can be recovered from such a path. We list a few further observations on this subject but leave the details to the reader.

Let $g \in \{0, \dots, G\}$ and $k \in \{0, \dots, K\}$. Fix any permutation of the machines with speeds from V_g and the jobs corresponding to g and K, that is, the fringe jobs with native group g if $k = 0$, or the core jobs of class k if $k > 0$ and g is the core group of k. Then there is a unique edge of type (1.) in the path for each such job j corresponding to the scheduling decision for j. Either this edge corresponds to the placement on a machine with a speed from V_g and a load from L_g and the job is integral, or it is treated as fractional and the last value of the state is changed correspondingly. Note that in this decision, we maintain the correct number of machines for each speed, no machine is overloaded, and setups are assigned correctly. Lastly, we take a closer look at the last value of the states. Let $(g, k, \iota_\ell^{(g,k)}, \xi, \mu, \lambda)$ be a state on a path from the start to an end state. In the values λ_1 and λ_2, we keep track of the fractional load due to fringe jobs with native group g and core jobs of classes with core group g. This means that $\lambda_2 = W_{g-1}$ (see (7.1)) and $\lambda_1 = W_g$ if $\ell = r(g, k)$. Moreover, the value λ_3 corresponds to the overall remaining fractional load that occurred before offset

against the remaining free space on the machines belonging to former groups, that is, $\lambda_3 = R_{g-1} + W_{g-2}$. Note that the definition of edges of type (3.) makes sure that the latter holds.

7.3 OPEN PROBLEMS

We provided a PTAS for the studied problem, but it is unclear whether an EPTAS is possible. Note that EPTAS results for uniform machines like, e.g., the one by Jansen [72] for uniform scheduling, are usually obtained using mixed integer linear programs. The EPTAS result for machine scheduling with setup times discussed in the last chapter make use of n-fold integer programming. Hence, as a first step, the underlying techniques could to be considered like, for example, algorithms for mixed integer programs with n-fold structure.

Furthermore, one could study the other two models considered in Chapter 6, that is, the splittable and the preemptive cases, with respect to uniform machines as well.

8

MACHINE SCHEDULING WITH A SHARED RESOURCE

8.1 INTRODUCTION

In the present chapter, we consider two closely related scheduling problems in which a set $\mathcal{J}$ of n jobs has to be scheduled on $m \in \mathbb{N}$ identical parallel machines that share a renewable resource. For both problems, a fixed amount $R \in \mathbb{N}$ of the renewable resource is given and each job occupies a share of the resource while being executed. In the first problem, each job has a processing time and a resource requirement that has to be met in order to execute the job, and in the second problem the processing time of a job depends on the resource amount allocated to the job. For both problems the goal is to minimize the makespan, i.e., the length of the schedule. We formalize these notions.

In the first problem, called *single resource constrained scheduling*, each job $j \in \mathcal{J}$ has a processing time $p_j \in \mathbb{N}$. To be processed, it requires an amount of $r_j \in \mathbb{N}$ of the given resource and one machine. A schedule of these jobs is given by a mapping $\tau : \mathcal{J} \to \mathbb{N}_{\geqslant 0}$ from jobs to starting times. It is *feasible* if at each point in time $t \in \mathbb{N}$ there are enough machines to schedule the jobs and the total resource amount of jobs scheduled at t does not exceed the resource limit R, i.e.:

$$\forall t \in \mathbb{N} : \sum_{j : t \in [\tau(j), \tau(j)+p_j)} r_j \leqslant R \tag{8.1}$$

$$\forall t \in \mathbb{N} : |\{j \in \mathcal{J} | t \in [\tau(j), \tau(j) + p_j)\}| \leqslant m \tag{8.2}$$

The objective is to find a feasible schedule $\tau : \mathcal{J} \to \mathbb{N}_{\geqslant 0}$ minimizing the makespan $\max_{j \in \mathcal{J}}(\tau(j) + p_j)$.

In the problem of *scheduling with resource dependent processing times*, on the other hand, each job $j \in \mathcal{J}$ has a set $D_j \subseteq [R]$ of valid resource values and a processing time function $\pi_j : D_j \to \mathbb{N}_{>0}$. The objective is to find a resource assignment $\rho_j \in D_j$ and a starting time $\tau(j) \in \mathbb{N}$ for each job $j \in \mathcal{J}$ such that the resulting schedule is feasible, i.e., $\sum_{j : t \in [\tau(j), \tau(j)+\pi_j(\rho_j))} \rho_j \leqslant R$ and $|\{j \in \mathcal{J} | t \in [\tau(j), \tau(j) + \rho_j)\}| \leqslant m$ for each $t \in \mathbb{N}$, and such that the makespan $\max_{j \in \mathcal{J}}(\tau(j) + \pi_j(\rho_j))$ is as small as possible. It is easy to see that the first problem is a special case of the second.

These problems arise naturally in different contexts, e.g., in highly parallelized computing where simultaneously active jobs share common memory, or in production logistics where additional personnel may speed up certain tasks. From a theoretical perspective, on the

other hand, the problems are sensible generalizations of problems like scheduling on identical parallel machines or bin packing and have already been studied in 1975 [49].

Both single resource constrained scheduling and scheduling with resource dependent processing times are NP-hard and therefore there is little hope to find optimal solutions efficiently. Hence, approximation algorithms were studied for these problems. However, by a simple reduction from the Partition Problem[1], one can see that there is no algorithm with an approximation guarantee better than $3/2$ for the problems unless $P = NP$. Therefore, a PTAS is not possible, while an approximation scheme with respect to the *asymptotic* approximation ratio still can be achieved for both problems.

An algorithm for a minimization problem has an asymptotic approximation ratio of α if there is a constant c such that the objective value $\text{ALG}(I)$ computed by the algorithm is upper bounded by $\alpha \, \text{OPT}(I) + c$. Approximation schemes with respect to the asymptotic ratio are denoted as APTAS, AEPTAS, or AFPTAS. The regular approximation ratio is, in distinction from the asymptotic ratio, also called the absolute ratio.

Both problems have been studied in the context of the absolute approximation ratio. For instance, Niemeier and Wiese [120] presented a $(2+\varepsilon)$-approximation for single resource constraint scheduling, and Kellerer [93] achieved a ratio of $3.5 + \varepsilon$ for scheduling with resource dependent processing times. These are the best known ratios so far. Concerning the asymptotic ratio, Epstein and Levin presented an AFPTAS for *bin packing with cardinality constraints*. This problem is equivalent to single resource constrained scheduling with unit processing times. Furthermore, there are AFPTAS results known for several related problems like, e.g., strip packing [94] or scheduling of parallel [94] or moldable parallel tasks[2] [71].

RESULTS AND METHODOLOGY. In this chapter, we present AFPTAS results for both problems at hand:

THEOREM 8.1: Let I be an instance of single resource constraint scheduling and p_{max} the biggest occurring processing time in I. For each $\varepsilon \in (0, 1)$ there is an algorithm which computes a schedule

1 In the bin packing problem, a set of numbers with sizes between 0 and 1 has to be packed into as few unit sized bins as possible, that is, the set of items has to be partitioned into sets with summed up item size at most 1 minimizing the size of the partition. If there was an approximation algorithm with ratio smaller than $3/2$ for bin packing, it could be used to distinguish between instances with optimum 3 or 2, which is equivalent to solving the partition problem. Furthermore, bin packing is equivalent to single resource constrained scheduling with unit processing times and $m = n$.

2 In the literature moldable jobs are sometimes called malleable, but the term moldable is the one which established itself. Today the term malleable specifies jobs which can be scheduled preemptively.

with makespan at most $(1+\varepsilon)\,\mathrm{OPT}(I) + \mathcal{O}(p_{max}\log(1/\varepsilon)/\varepsilon)$, and has a running time that is polynomial in the input length and $1/\varepsilon$.

Note that this result directly improves the additive term of the AFPTAS for bin packing with cardinality constraints by Epstein and Levin [45] which has an additive terms that is exponential in $1/\varepsilon$.

THEOREM 8.2: Let I be an instance of scheduling with resource dependent processing times and π_{max} the biggest occurring processing time in I. For each $\varepsilon \in (0,1)$ there is an algorithm which computes a schedule with makespan at most $(1+\varepsilon)\,\mathrm{OPT}(I) + \mathcal{O}(\pi_{max}\log(1/\varepsilon)/\varepsilon)$ and has a running time that is polynomial in the input length and $1/\varepsilon$.

In the following, we give a brief overview on how we achieve these result, followed by a more detailed literature review. All our algorithms utilize a variety of linear programming and rounding techniques which are altered and recombined to serve our needs. To solve certain configuration LPs, we apply algorithms [59] for the Max-Min Resource Sharing problem as was done in [71]. This approach yields a better running time than the standard approach via the ellipsoid algorithm.

For the first problem, we partition the set of jobs into wide and narrow jobs based on their resource requirements and apply linear grouping for the wide jobs. To handle the narrow jobs, we adapt the notion of windows that was introduced by Epstein and Levin [45]. However, and this is crucial for both our algorithms, we manage to bound the number of windows to be in $\mathcal{O}(1/\varepsilon^2)$, via a second elaborate rounding step. This makes the small additive factor possible and is essential for the generalization to the resource dependent variant. By this approach we get an additive term of $\mathcal{O}(p_{max}/\varepsilon^2)$. Using geometric instead of linear grouping for the wide jobs and a new argument for the reduction of the windows, we are able to further improve the additive term to $\mathcal{O}(p_{max}\log(1/\varepsilon)/\varepsilon)$. The results concerning the first problem are presented in Section 8.2.

For scheduling with resource dependent processing times, we use a preemptive solution to define an instance for the fixed-resource problem in which the original jobs may be split into parts with different resource requirements. We then apply the same techniques we used for the fixed-resource case. Using a basic feasible solution of a specific LP, we change the fixed-resource instance such that all but a few jobs are no longer split. For these jobs, the schedule is provided by the fixed-resource AFPTAS, and the rest of the jobs are scheduled in the end causing only a small error. Furthermore, we discuss how the results can be generalized to variants of the second problem with a more succinct encoding of the processing time functions. The results concerning the second problems are presented in Section 8.3.

As a side note, we would like to point out that our techniques also work for the scenario where the resource needs to be allocated contiguously, that is, the resource is represented by an interval of length R and each job is assigned a subinterval whose length corresponds to

the resource amount assigned to the job. For the first problem, this scenario amounts to the problem of strip packing with cardinality constraints, where the cardinality constraint is given by the requirement that at most m jobs may executed at each time step. Hence, we provide an AFPTAS for this problem as well.

RELATED WORK. The first result for scheduling jobs on identical machines with additional resources was presented in 1975 by Garey and Graham [49]. Given m identical machines and s distinct resource limits such that each job requires an amount of each of the s distinct resources, they have shown that the greedy list algorithm delivers a schedule of length at most $(s+2-(2s+1)/m)\,\mathrm{OPT}$. This gives an absolute approximation ratio of $(3-3/m)$ for the case of $s=1$ which equates to the problem of single resource constraint scheduling. In the same year Garey and Johnson [50] showed that this general scheduling problem is NP-complete even if just one resource is given, i.e., $s=1$. Lately, Niemeier and Wiese [120] presented a $(2+\varepsilon)$-approximation for single resource constraint scheduling, and this is the best ratio known so far. By a simple reduction from the Partition Problem, one can see that there is no algorithm with an approximation guarantee better than $3/2$ for this problem unless $P = NP$.

The variant of single resource constraint scheduling where the resource has to be allocated contiguously is equivalent to the problem of strip packing with cardinality constraints. For strip packing there exists an AFPTAS with additive term $\mathcal{O}(h_{max}/\varepsilon^2)$ by Kenyon and Rémila [94], where h_{max} is the largest occurring height of an item and corresponds to the largest processing time in our scenario. The additive term was later improved to $\mathcal{O}(h_{max}\log(1/\varepsilon)/\varepsilon)$ by Sviredenko [138] and Bougeret et al. [21].

Moreover, in the case that we have less jobs than machines, i.e., $n \leqslant m$, the here considered problems are reducible to the problem of scheduling parallel or moldable parallel tasks, respectively. In the problem scheduling parallel tasks, we are given a set of m machines and a set of jobs such that each job needs a given number of machines to be processed on. If the number of machines m in single resource constrained scheduling is larger than n, we can never schedule more than m jobs at the same time, and this constraint is trivially fulfilled in each schedule. We then consider the resource requirement in single resource constrained scheduling as the needed number of machines in scheduling parallel tasks. Scheduling moldable parallel tasks is defined analogously to scheduling parallel tasks. However, in this variant jobs do not have a fixed number of machines they can be processed on. Instead each job has a set of allowed machine numbers and a processing time function similar to the case of scheduling with resource dependent processing times. Concerning scheduling parallel tasks, the AFPTAS results for strip packing work as well, and therefore

the smallest additive term of an AFPTAS has size $\mathcal{O}(p_{max} \log(1/\varepsilon)/\varepsilon)$ [21, 138]. On the other hand, for scheduling moldable parallel tasks the AFPTAS with the smallest additive term was presented by Jansen [71] and has size $\mathcal{O}(\pi_{max}/\varepsilon^2)$.

For scheduling with resource dependent processing times, the first result was achieved by Grigoriev et al. [60], who studied the unrelated scheduling variant in which the processing times depend on the machine as well as on the resource assignment. They achieved a 3.75-approximation algorithm. This was improved to $3.5 + \varepsilon$ by Kellerer [93] for the identical machine version. Grigoriev et al. [61] presented a $(3 + \varepsilon)$-approximation for a version of the problem where the jobs are preassigned to machines that also works when the processing time functions are encoded more succinctly. Rather recently, Kling et al. [98] considered a related problem, where for each job j a resource value $\rho_{j,t}$ has to be chosen for each timestep t it is processed in. Furthermore, each job j has a resource requirement r_j and a processing volume p_j, and if j receives a resource value of $\rho_{j,t}$ at timestep t, exactly $\min_{1,\rho_{j,t}/r_j}$ units of its processing volume are finished during t. They provide a $(2 + 1/(m-2))$-approximation for this case and show that the problem is NP-hard.

8.2 SINGLE RESOURCE CONSTRAINED SCHEDULING

In this section, we will prove Theorem 8.1. First, we introduce an algorithm with additive term $\mathcal{O}(p_{max}/\varepsilon^2)$ and discuss the modifications we make to achieve the improved additive term afterwards. In the following, we will call the resource requirement of a job its width and its processing time its height. We use similar notions of height and width for all introduced structures as (generalized) configurations and windows.

First, we will give a slightly more detailed high-level view of the algorithm and present a short overview. The steps of the algorithms are described in detail in the following sub sections. We differentiate two cases: first $1/\varepsilon < m$ and second $1/\varepsilon \geqslant m$, where the case $1/\varepsilon < m$ is more involved.

In the first case, the set of jobs $\mathcal{J}$ is partitioned into wide and narrow jobs where wide jobs have a larger resource amount than narrow jobs. The resource amount of the wide jobs is rounded by linear grouping such that we have to deal with just $\mathcal{O}(1/\varepsilon^2)$ different sizes. For this rounded instance, the algorithm computes a preemptive schedule using a configuration LP. Broadly speaking, a configuration is a selection of jobs that can be processed at the same time. After that, each configuration is partitioned in the wide job part and the narrow job part. The spare area (resource amount and used machines) not used by the wide jobs will be called window following the notation in [45]. In simplified terms, a window can be seen as the residual space

(resource and machine number) that is left by a configuration of wide jobs. By constructing a preemptive schedule first, instead of solving the LP by Levin and Epstein [45] directly, we manage to choose the width of each window more adjusted to the given instance. This adjustment improves the number of operations and the makespan of the solution slightly. However, the crucial step is to reduce the number of different window sizes such that it just depends on ε by simultaneously adding (not too much) processing time to the schedule. This is achieved by a further grouping step. After a solution with reduced number of windows is found, an integral schedule can be computed by adding some processing time to the schedule.

In the second case, partitioning the jobs by resource amount is not required and all job resource amounts can be rounded to few sizes. This simplifies the problem such that we can generate a schedule directly after generating the preemptive solution.

8.2.1 *First Case:* $1/\varepsilon < m$

Let an instance $I = (\mathcal{J}, m, R)$ and $\varepsilon > 0$ with $1/\varepsilon < m$ be given. In the following, we sometimes write $\mathcal{J}(I)$ to refer to the set of jobs in the instance. We assume w.l.o.g. that $1/\varepsilon \in \mathbb{N}$ and $m < n$ since otherwise we have the problem of scheduling parallel tasks for which an AFPTAS is already known [21, 138]. The algorithm can be summarized as follows:

(i) Define ε' such that it is the largest value with $1/\varepsilon' \in \mathbb{N}$ and $\varepsilon' \leqslant \varepsilon/6$. Compute $p_{max} = \max\{p_j | j \in \mathcal{J}(I)\}$.

(ii) Construct a rounded instance $I_{sup,\varepsilon'}$ with at most $1/\varepsilon'^2$ wide jobs using linear grouping.

(iii) Solve a configuration linear program LP_{pre} to find a preemptive schedule x_{pre} for $I_{sup,\varepsilon'}$ which uses at most $|\mathcal{J}(I_{sup,\varepsilon'})| + 1$ configurations and has a makespan of at most $(1+\varepsilon')^2 \textsc{Opt}(I)$.

(iv) Transform the obtained configurations into generalized configurations which are composed of a configuration part for wide and a window part for narrow jobs. This reduces the number of used configurations to $\min\{|\mathcal{J}(I_{sup,\varepsilon'})| + 1, (\frac{1}{\varepsilon'^2})^{1/\varepsilon'}\}$.

(v) Reduce the number of different windows further using a grouping step which lengthens the solution by a factor of at most $(1+\varepsilon')$. Generate a solution for $I_{sup,\varepsilon'}$ and a generalized configuration linear program LP_W which uses at most $5 + 3/\varepsilon'^2$ configurations and has a makespan of at most $(1+\varepsilon')^3 \textsc{Opt}(I)$.

(vi) Given this solution for $I_{sup,\varepsilon'}$ and LP_W, generate an integral schedule for I obtaining an overall makespan of at most $(1+\varepsilon')^4 \textsc{Opt}_{pre}(I) + (3/\varepsilon'^2 + 1/\varepsilon' + 6)p_{max}$.

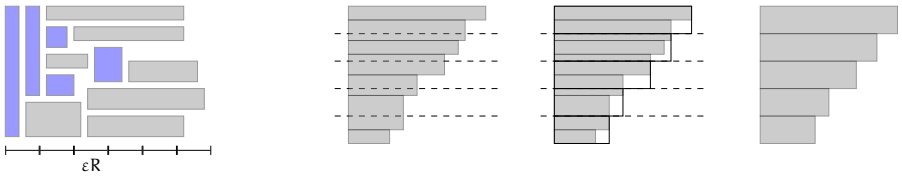


Figure 8.1: Rounding the Instance. Blue items represent narrow jobs, while gray rectangles represent wide jobs. On the right, one can see the steps of the linear grouping.

Rounded Instance – Step (ii)

In this step, we describe how to generate a rounded instance $I_{sup,\varepsilon}$ for a given instance I and an $\varepsilon \in \mathbb{R}$ with $1/\varepsilon \in \mathbb{N}$ using standard techniques. The algorithm will use the rounded instance $I_{sup,\varepsilon'}$.

We partition the given set of jobs $\mathcal{J}$ into a set of wide jobs $\mathcal{J}_{W,\varepsilon} := \{j \in \mathcal{J} | r_j \geqslant \varepsilon R\}$ with resource amount at least εR and a set of narrow jobs $\mathcal{J}_{N,\varepsilon} := \mathcal{J} \setminus \mathcal{J}_{W,\varepsilon}$. We round the resource amount of the wide jobs by linear grouping—a method introduced by Fernandez de la Vega and Lueker [142] for bin packing and extended by Kenyon and Rémila [94] to strip packing. To this end, we interpret each job j as a rectangle with width r_j and height p_j and place these rectangles on top of each other such that they form a vertical stack ordered by increasing resource amount from bottom to top (see Figure 8.1). Let $P_W := P(\mathcal{J}_{W,\varepsilon})$ be the height of the stack. Consider the horizontal lines at the heights $i\varepsilon^2 P_W$. Each job intersecting with one of these lines is divided at the intersection and split into two new jobs. We denote by $\mathcal{J}_{W,\varepsilon,i}$ the set of possibly split jobs which lie between the lines $(i-1)\varepsilon^2 P_W$ and $i\varepsilon^2 P_W$. We have $G := 1/\varepsilon^2$ many sets $\mathcal{J}_{W,\varepsilon,i}$, called groups, where $\mathcal{J}_{W,\varepsilon,G}$ is the group containing the jobs with the largest resource amount. Define by $R_i := \max\{r_j | j \in \mathcal{J}_{W,\varepsilon,i}\}$ the largest resource amount in group i. Note that $R_i \leqslant R_{i+1}$ for all $i < G$.

The rounded instance $I_{sup,\varepsilon}$ is generated as follows: Let $\mathcal{J}_{sup,\varepsilon}$ be the set containing all jobs from $\mathcal{J}_{N,\varepsilon}$ and one additional job for each $i \in \{1, \dots G\}$ with processing time $P(\mathcal{J}_{W,\varepsilon,i})$ and resource amount R_i and let be $I_{sup,\varepsilon} := (\mathcal{J}_{sup,\varepsilon}, m, R)$. We denote by $\mathcal{J}_{sup,W,\varepsilon}$ the set of wide jobs in $I_{sup,\varepsilon}$.

Preemptive Schedule – Step (iii)

The next step of the algorithm is to find a preemptive schedule for our rounded instance. Our definition of a preemptive schedule differs slightly from other definitions of preemptive schedules. In this section, we define the kind of preemptive schedule we speak of and prove that such a preemptive schedule for any given instance $I = (\mathcal{J}, m, R)$ and given ε can be found in time polynomial in n and $1/\varepsilon$ (see Lemma 8.3). Furthermore, in the end of this section, we prove that the makespan of an optimal preemptive solution of the rounded instance $I_{sup,\varepsilon}$ can

be bounded by the makespan of an optimal preemptive solution for I loosing a factor of $(1+\varepsilon)$ (see Lemma 8.5).

In a preemptive schedule, each job can be interrupted and restarted at no cost. Usually, in a preemptive schedule, parts of the same job are not allowed to be processed at the same time. However, since we have rounded the wide jobs, we allow the same wide job to be processed more than once at the same point in time. This is necessary, since in the rounded instance one wide job represents many different jobs of the original instance, and in an optimal solution of the original instance these jobs can be processed at the same time. If we do not allow this parallel processing of the same job, the optimal solutions of rounded and original instance are not comparable. We denote the optimum objective value of preemptive schedules in which jobs wider than εR are considered wide as $\mathrm{OPT}_{\mathrm{pre}}(I,\varepsilon)$.

A *configuration* $C \in \mathbb{N}^{\mathcal{J}}$ is a multiset of jobs which can be processed at the same point in time without violating the conditions of a feasible schedule. For a given configuration C the value $C(j)$ says how often the job j is contained in C. We allow $C(j) \in \{0,1\}$ if $j \in \mathcal{J}_{N,\varepsilon}$, and $C(j) \in \{0,\dots,1/\varepsilon\}$ if $j \in \mathcal{J}_{W,\varepsilon}$, since each job in $\mathcal{J}_{W,\varepsilon}$ has width at least εR and, therefore, it is not possible to schedule more than $1/\varepsilon$ of them at the same time. A configuration is *feasible* for a given instance I if $m(C) := \sum_{j\in\mathcal{J}} C(j) \leqslant m$ and $R(C) := \sum_{j\in\mathcal{J}} C(j) r_j \leqslant R$. Denote by $\mathcal{C}_{I,\varepsilon}$ the set of all feasible configurations of I. An optimal solution of the following linear program $\mathrm{LP}_{\mathrm{pre}}(I,\varepsilon)$ delivers an optimal preemptive schedule for any instance I.

$$\min \sum_{C\in\mathcal{C}_{I,\varepsilon}} x_C \tag{8.3}$$

$$\sum_{C\in\mathcal{C}_{I,\varepsilon}} C(j) x_C \geqslant p_j \quad \forall j \in \mathcal{J} \tag{8.4}$$

$$x_C \geqslant 0 \quad \forall C \in \mathcal{C}_{I,\varepsilon} \tag{8.5}$$

The variable x_C denotes the processing time of configuration $C \in \mathcal{C}_{I,\varepsilon}$. Inequality (8.4) ensures that each job is scheduled.

LEMMA 8.3. *For any Instance I and any $\varepsilon,\delta > 0$, we can find a solution $x_{\mathrm{pre},\varepsilon}$ to $\mathrm{LP}_{\mathrm{pre}}(I,\varepsilon)$ with*

$$\sum_{C\in\mathcal{C}_{I,\varepsilon}} (x_{pre,\varepsilon})_C \leqslant (1+\delta)\,\mathrm{OPT}_{\mathrm{pre}}(I,\varepsilon)$$

in $\mathcal{O}(|\mathcal{J}|(\ln(|\mathcal{J}|)+1/\delta^2)(|\mathcal{J}_W|z+|\mathcal{J}_N|+mz'^2/\delta^2))$ operations, where $z = \min\{1/\varepsilon, m\}$ and $z' = \min\{1/\delta, m\}$. Furthermore, a solution with the same objective value and at most $|\mathcal{J}|+1$ non-zero components can be found in $\mathcal{O}(|\mathcal{J}|^{2.5356}(\ln(|\mathcal{J}|)+\varepsilon^{-2}))$ further operations.

Proof. Our approach to find this preemptive solution is to transform I into an instance of a Max-Min-Resource-Sharing problem. Then we

use the algorithm by Grigoriadis et al. [59] to find a solution x for this transformed instance. This solution is transformed back to be a solution x' for I. Since this solution might not be a basic solution, we use an algorithm from [92] to transform x' to a basic solution $x_{pre,\varepsilon}$.

An instance of the Max-Min-Resource-Sharing problem consists of a nonempty convex compact set B and a function $f : B \to \mathbb{R}^M$ which consists of M nonnegative continuous concave functions $f_j : B \to \mathbb{R}$ with $j \in \{1, \ldots, M\}$. The objective is to find the value $\lambda^* := \max\{\lambda | \exists x \in B,\ f(x) \geqslant \lambda \mathbf{1}_M\}$ and a vector $x \in B$, for which $f(x) \geqslant \lambda^* \mathbf{1}_M$ holds. Above, $\mathbf{1}_M \in \mathbb{R}^M$ denotes the vector which is 1 at each position.

We transform LP_{pre} to a Max-Min-Resource-Sharing problem, by defining

$$B := \left\{ x \in \mathbb{R}_{\geqslant 0}^{|\mathcal{C}_{I,\varepsilon}|} \;\middle|\; \sum_{C \in \mathcal{C}_{I,\varepsilon}} x_C = 1 \right\}$$

as the set of selections of processing times for feasible configurations with overall makespan 1. Furthermore, for each $j \in \mathcal{J}$, we define

$$f_j : B \to \mathbb{R}_{\geqslant 0}, x \mapsto \sum_{C \in \mathcal{C}_{I,\varepsilon}} C(j) \frac{x_C}{p_j}$$

as the fraction of job j that is scheduled in a preemptive schedule derived from x. Note that if $f_j(x) \geqslant 1$ for each job, then each job is fully scheduled in the preemptive solution.

The algorithm by Grigoriadis et al. [59] computes an $x \in B$ satisfying $f_j(x) \geqslant (1-\rho)\lambda^*$ for each $j \in \mathcal{J}$ and a given ρ. We will choose $\rho \in \mathcal{O}(\delta)$. The algorithm iterates the following steps: Given an initial solution $\check{x}$, the algorithm computes a profit vector $q = q(\check{x}) \in \mathbb{R}^{|\mathcal{J}|}$ for the current value $\check{x}$. After that an $(1-\delta')$-approximative solution $\hat{x}$ of the problem $\max\{q^T f(x) | x \in B\}$ has to be computed where δ' depends linearly on ρ. This problem is called block-problem and a solver has to be provided. In the last step of an iteration, the new value of the vector $\check{x}$ is set to $(1-\tau)\check{x} + \tau\hat{x} \in B$, where $\tau \in (0,1)$ is an appropriate step length. One of these iterations where we update $\check{x}$ is called a coordination step and the algorithm needs at most $\mathcal{O}(|\mathcal{J}|(\ln(|\mathcal{J}|) + \rho^{-2}))$ of them.

Next, we describe how to solve the problem $\max\{q^T f(x) | x \in B\}$. Note that

$$q^T f(x) = \sum_{C \in \mathcal{C}_{I,\varepsilon}} x_C \sum_{j \in \mathcal{J}} \frac{q_j}{p_j} C(j).$$

Hence, to solve the problem, it suffices to compute a configuration C^* that maximizes $\sum_{j \in \mathcal{J}} \frac{q_j}{p_j} C(j)$ and set $x_{C^*} := 1$ and $x_C := 0$ for all $C \neq C^*$. Therefore, in each coordination step at most one new configuration is added to the solution. To find such a configuration C^*, we have to solve the following integer linear program $ILP_{kKP}(q, \mathcal{J}_{N,\varepsilon}, \mathcal{J}_{W,\varepsilon}, m, R)$:

$$\max \sum_{j \in \mathcal{J}} \frac{q_j}{p_j} a_j$$

$$\sum_{j \in \mathcal{J}} a_j \leqslant m$$
$$\sum_{j \in \mathcal{J}} r_j a_j \leqslant R$$
$$a_j \in \{0, 1\} \quad \forall j \in \mathcal{J}_{N,\varepsilon}$$
$$a_j \in \{0, \ldots, 1/\varepsilon\} \quad \forall j \in \mathcal{J}_{W,\varepsilon}$$

This ILP is equivalent to the ILP formulation of the knapsack problem with cardinality constraint (kKP). This problem is similar to the knapsack problem and has the additional constraint that at most k items are allowed to be put into the knapsack. In our case, we have that $k = m$ since we can schedule at most m machines at the same time. A difference in the formulation is that the wide jobs can be picked several times. There is an FPTAS by Mastrolilli and Hutter [114] to solve this problem. To use it, we have to duplicate each wide job $z = \min\{1/\varepsilon, m\}$ times since it can be scheduled up to z times at the same time without violating any constraint. The computation of a $(1+\delta')$-approximate solution for a kKP instance (I, k, ε) takes $\mathcal{O}(|I| + kz'^2/\delta'^2)$ operations with $z' := \min\{k, 1/\varepsilon\}$. In our case, we have $|I| = |\mathcal{J}_{W,\varepsilon}|z + |\mathcal{J}_{N,\varepsilon}|$, $k = m$ and $z' = \min m, 1\delta'$, and hence we need at most $\mathcal{O}(|\mathcal{J}_{W,\varepsilon}|z + |\mathcal{J}_{N,\varepsilon}| + mz'^2\delta'^{-2})$ operations to find a solution a^* to the ILP. To get a configuration C^* we define $C^*(j) := a^*_j$ for each $j \in \mathcal{J}$.

Let x be a solution to the described Max-Min-Resource-Sharing problem calculated by the algorithm in [59]. Note that the overall makspan of all used configurations in x is 1 at the moment and we do not necessarily have that $f_j(x) \geqslant 1$ for each job j. However, we are interested in a solution with makespan at most $(1+\delta)\,\mathrm{OPT}_{pre}(I, \varepsilon)$ and where $f_j(x) \geqslant 1$ for each job j (which is required to schedule every job entirely). Normally, we would expect to have to do a binary search to find the value of $(1+\delta)\,\mathrm{OPT}_{pre}(I, \varepsilon)$ and to solve the according Max-Min-Resource-Sharing instance. However, since the functions f are linear, we can avoid the binary search for the optimal makespan (see Claim 8.4). Instead of applying a binary search framework, we scale x such that $f(x) \geqslant \mathbf{1}_M$. As a result, we have $\sum_{C \in \mathcal{C}_{I,\varepsilon}} C(j)x_C \geqslant p_j$ for each $j \in \mathcal{J}$, which ensures that each job is entirely scheduled. Furthermore, we show how to choose ρ such that $\sum_{C \in \mathcal{C}_{I,\varepsilon}} x_C \leqslant (1+\delta)\,\mathrm{OPT}_{pre}(I, \varepsilon)$ holds.

Claim 8.4. Let $x \in B$ with $f(x) \geqslant (1-\rho)\lambda^* \mathbf{1}_M$ and $\hat{\lambda} := \min\{f_j(x) | j \in \mathcal{J}\}$. It holds that $f(x/\hat{\lambda}) \geqslant \mathbf{1}_M$ and

$$\sum_{C \in \mathcal{C}_{I,\varepsilon}} \frac{1}{\hat{\lambda}} x_C \leqslant \frac{1}{(1-\rho)}\,\mathrm{OPT}_{pre}(I, \varepsilon).$$

The first part of the claim $f(x/\hat{\lambda}) \geqslant \mathbf{1}_M$ is obvious since

$$f(x/\hat{\lambda}) = f_j(x)/\hat{\lambda} = f_j(x)/\min\{f_j(x) | j \in \mathcal{J}\} \geqslant 1$$

for all $j \in \mathcal{J}$. To prove the second part, we consider a more general definition of λ^* and B. Let $t \in \mathbb{R}_{\geqslant 0}$ be a target makespan and let

$$B_t := \left\{ x \in \mathbb{R}_{\geqslant 0}^{|\mathcal{C}_{I,\varepsilon}|} \;\middle|\; \sum_{C \in \mathcal{C}_{I,\varepsilon}} x_C = t \right\},$$

$$\lambda_t^* := \max\{\lambda | \exists x \in B_t \forall j \leqslant M : f_j(x) \geqslant \lambda\}.$$

Let $t' \in \mathbb{R}_{\geqslant 0}$. By making use of the linearity of f and $g : B \to \mathbb{R}, x \mapsto \sum_{C \in \mathcal{C}_{I,\varepsilon}} x_C$ it is easy to see that $B_t = \frac{t}{t'} B_{t'}$ and $\lambda_t^* = \frac{t}{t'} \lambda_{t'}^*$. Using $t = 1$, $\lambda_1^* = \lambda^*$, $t' = \mathrm{OPT}_{\mathrm{pre}}(I, \varepsilon)$, and $\lambda^*_{\mathrm{OPT}_{\mathrm{pre}}(I,\varepsilon)} = 1$, this yields

$$\sum_{C \in \mathcal{C}_{I,\varepsilon}} \frac{1}{\hat{\lambda}} x_C = \frac{1}{\hat{\lambda}} \leqslant \frac{1}{(1-\rho)} \frac{1}{\lambda^*} \leqslant \frac{1}{1-\rho} \mathrm{OPT}_{\mathrm{pre}}(I, \varepsilon)$$

and concludes the proof of the claim.

Therefore, to find a solution x' to $\mathrm{LPpre}(I, \varepsilon)$, we scale x by $1/\hat{\lambda}$ and set $\rho := \frac{\delta}{1+\delta} \in \mathcal{O}(\delta)$. Since $f(x') \geqslant \mathbf{1}_M$ holds, x' fulfills Equation (8.4) for each $j \in \mathcal{J}$. Hence, x' is a feasible solution to $\mathrm{LP}_{\mathrm{pre}}$. Since $\rho, \delta' \in \mathcal{O}(\delta)$, we need $\mathcal{O}(|\mathcal{J}|(\ln(|\mathcal{J}|) + \delta^{-2})(|\mathcal{J}_{W,\varepsilon}|z + |\mathcal{J}_{N,\varepsilon}| + mz'^2\delta^{-2}))$ operations to find x'. However, x' might have to many non-zero entries.

The algorithm performed $\mathcal{O}(|\mathcal{J}|(\ln(|\mathcal{J}|) + \delta^{-2}))$ coordination steps adding at most one configuration in every step. Therefore, x' has at most this amount of non-zero components since we add at most one configuration in each step. We reduce this number by computing a basic solution $x_{\mathrm{pre},\varepsilon}$ to $\mathrm{LP}_{\mathrm{pre}}$ by using x' as a start vector. Beling and Megiddo [14] described how to compute a basic solution for the problem $Ax = b$, $x \geqslant 0$, given a start solution $\bar{x}$, where $A \in \mathbb{Q}^{m \times n}$. Later Ke et al. [92] presented a faster way of rectangular matrix multiplication. Combined it is possible to find a basic solution in $\mathcal{O}(m^{1.5356}n)$ time. Since our linear program is not in standard form, we have to add $|\mathcal{J}|$ variables and the equation $\sum_{C \in \mathcal{C}_{I,\varepsilon}} x_C = \sum_{C \in \mathcal{C}_{I,\varepsilon}} (x_{\mathrm{pre}})_C$. Furthermore, we use just the configurations which have a non-zero component in x'. Therefore, we can compute a basic solution $x_{\mathrm{pre},\varepsilon}$ in $\mathcal{O}(|\mathcal{J}|^{1.5356}(|\mathcal{J}|(\ln(|\mathcal{J}|) + \delta^{-2}))) \leqslant \mathcal{O}(|\mathcal{J}|^{2.5356}(\ln(|\mathcal{J}|) + \delta^{-2}))$. In total, we need at most

$$\mathcal{O}(|\mathcal{J}|(\ln(|\mathcal{J}|) + \delta^{-2})(|\mathcal{J}_{W,\varepsilon}|z + |\mathcal{J}|^{1.5356} + mz'^2/\delta^2))$$

operations to find a basic solution $x_{\mathrm{pre},\varepsilon}$ to the linear program. □

We now look at the relation between optimal solutions to I and $I_{\mathrm{sup},\varepsilon}$:

LEMMA 8.5. *It holds that* $\mathrm{OPT}_{\mathrm{pre}}(I_{\mathrm{sup},\varepsilon}, \varepsilon) \leqslant (1+\varepsilon)\,\mathrm{OPT}_{\mathrm{pre}}(I, \varepsilon)$

Proof. Let a solution to $\mathrm{LP}_{\mathrm{pre}}(I, \varepsilon)$ and $\mathrm{LP}_{\mathrm{pre}}(I_{\mathrm{sup},\varepsilon}, \varepsilon)$ each be given. Since each group $\mathcal{J}_{W,\varepsilon,i}$ has the same summed up processing time, we can split that large job in $I_{\mathrm{sup},\varepsilon}$ with processing time $P(\mathcal{J}_{W,\varepsilon,i})$ and

resource amount R_i and schedule it instead of the jobs in $\mathcal{J}_{W,\varepsilon,i+1}$ in the solution to $LP_{pre}(I,\varepsilon)$. The widest group cannot be scheduled inside other jobs, but we can shift this group on top of the schedule. This group has a makespan of at most $\varepsilon^2 P_W \leqslant \varepsilon \operatorname{OPT}_{pre}(I,\varepsilon)$. So, for each solution to $LP_{pre}(I,\varepsilon)$, we can generate a solution to $LP_{pre}(I_{sup,\varepsilon},\varepsilon)$ which is lengthened by at most $\varepsilon \operatorname{OPT}_{pre}(I,\varepsilon)$, hence it holds that $\operatorname{OPT}_{pre}(I_{sup,\varepsilon},\varepsilon) \leqslant (1+\varepsilon)\operatorname{OPT}_{pre}(I,\varepsilon)$. □

Let us recapitulate the steps taken by the algorithm so far. In the first step, it computed $I_{sup,\varepsilon'}$. Next in (iii)—as described in this subsection—the algorithm computes $x_{pre,\varepsilon'}$ defined as the $(1+\varepsilon')$-approximate solution to $LP_{pre}(I_{sup,\varepsilon'},\varepsilon')$ with at most $|\mathcal{J}_{sup,\varepsilon'}|+1$ non-zero components. Since $|\mathcal{J}_{sup,W,\varepsilon'}| \leqslant \varepsilon'^{-2}$ and $1/\varepsilon < m$, we can generate this solution in $\mathcal{O}(|\mathcal{J}_{sup,\varepsilon'}|(\ln(|\mathcal{J}_{sup,\varepsilon'}|)+\varepsilon^{-2})(|\mathcal{J}_{sup,\varepsilon'}|^{1.5356}+m/\varepsilon^4))$ operations. By Lemma 8.5, we know that the solution $x_{pre,\varepsilon'}$ has a makespan of at most:

$$\sum_{C\in\mathcal{C}_{I_{sup,\varepsilon',\varepsilon'}}} (x_{pre,\varepsilon'})_C \overset{\text{L. 8.3}}{\leqslant} (1+\varepsilon')\operatorname{OPT}_{pre}(I_{sup,\varepsilon'},\varepsilon') \overset{\text{L. 8.5}}{\leqslant} (1+\varepsilon')^2\operatorname{OPT}_{pre}(I,\varepsilon') \leqslant (1+\varepsilon')^2\operatorname{OPT}(I)$$

Generalized Configurations – Step (iv)

Next, we will consider the solution $x_{pre,\varepsilon'}$ of $LP_{pre}(I_{sup,\varepsilon'},\varepsilon')$ generated in Step (iii) of the algorithm. Now the splitting point $\varepsilon' R$ between wide and narrow jobs is clear from the context and we will discard the index ε' in all occurring sets and definitions.

The number of configurations used in x_{pre} still depends on the input length, that is, n, because we did not round the sizes of the narrow jobs. For each used configuration, we have to pay one additional p_{max} when generating an integral schedule. Therefore, we achieve an additive term depending on n when computing an integral solution at this point. Since we aim for an additive term depending solely on $1/\varepsilon$, we give an additional structure to this solution. For this purpose, we use a set of containers for the narrow jobs, called windows, which were first introduced by Epstein and Levin [45].

A *window* $w = (w_r, w_m)$ is a pair consisting of a resource amount $R(w) = w_r$ and a number of machines $m(w) = w_m$. As for a configuration, the total time a window is processed is denoted as $p(w)$ and is called its height. At each point in time for a given window w, there $m(w)$ jobs with summed up resource amount $R(w)$ may be processed. For two windows w_1 and w_2, we write $w_1 \leqslant w_2$ if and only if $R(w_1) \leqslant R(w_2)$ and $m(w_1) \leqslant m(w_2)$.

For a given configuration $C \in \mathcal{C}_I$, we denote by $C|_{\mathcal{J}_W}$ the configuration consisting of all wide jobs in C; and for a given set of configurations $\mathcal{C} \subseteq \mathcal{C}_I$, we define $\mathcal{C}_W := \{C|_{\mathcal{J}_W} | C \in \mathcal{C}\}$. Note that

each configuration in $\mathcal{C}_W$ contains at most $1/\varepsilon'$ items since each of the wide jobs needs at least $\varepsilon' R$ resource. A *generalized configuration* (C, w) is a pair consisting of a configuration $C \in \mathcal{C}_W$ and a window w. (C, w) is valid for an instance I, if $m(w) \leqslant m - m(C)$ and $R(w) \leqslant R - R(C)$. For a configuration $C \in \mathcal{C}_W$ with $R(C) < R$ we define by $w(C) := (R - R(C), m - m(C))$ the *main window* for C.

Let $\mathcal{W}$ be a set of windows and $\mathcal{C}_W$ a set of configurations consisting exclusively of wide jobs. The following linear program LP_W describes the relation between generalized configurations and jobs assigned to them. This linear program was introduced by Epstein and Levin [45] but with a different set of generalized configurations.

$LP_W(I, \mathcal{C}_W, \mathcal{W})$:

$$\sum_{C \in \mathcal{C}_W} \sum_{\substack{w \in \mathcal{W} \\ w \leqslant w(C)}} C(j) x_{(C,w)} \geqslant p_j, \qquad \forall j \in \mathcal{J}_W \tag{8.6}$$

$$\sum_{w \in \mathcal{W}} y_{j,w} \geqslant p_j, \qquad \forall j \in \mathcal{J}_N \tag{8.7}$$

$$m(w) \sum_{\substack{C \in \mathcal{C}_W \\ w(C) \geqslant w}} x_{(C,w)} \geqslant \sum_{j \in \mathcal{J}_N} y_{j,w}, \qquad \forall w \in \mathcal{W} \tag{8.8}$$

$$R(w) \sum_{\substack{C \in \mathcal{C}_W \\ w(C) \geqslant w}} x_{(C,w)} \geqslant \sum_{j \in \mathcal{J}_N} r_j y_{j,w}, \qquad \forall w \in \mathcal{W} \tag{8.9}$$

$$x_{(C,w)} \geqslant 0, \qquad \forall C \in \mathcal{C}_W, \forall w \in \mathcal{W} \tag{8.10}$$

$$y_{j,w} \geqslant 0, \qquad \forall w \in \mathcal{W}, \forall j \in \mathcal{J}_N \tag{8.11}$$

The variable $x_{(C,w)}$ denotes the processing time of the generalized configuration (C, w), and the value $y_{j,w}$ indicates which amount of job j is processed in window w. Inequalities (8.6) and (8.7) ensure that for each job there is enough processing time reserved, while Equalities (8.8) and (8.9) ensure that in each window there is enough space to schedule the contained jobs. Given a solution (x, y) to LP_W, we define

$$P(x) := \sum_{C \in \mathcal{C}_W} \sum_{\substack{w \in \mathcal{W} \\ w \leqslant w(C)}} x_{(C,w)},$$

which is the makespan of (x, y), and

$$P(w, x) := \sum_{\substack{C \in \mathcal{C}_W \\ C(w) \geqslant w}} x_{(C,w)}$$

which is the summed up processing time of a window $w \in \mathcal{W}$ in x.

We denote by $\mathcal{C}_{pre} := \{C \in \mathcal{C}_I | (x_{pre})_C > 0\}$ the set of configurations with a non-zero component in x_{pre}, where $(x_{pre})_C$ denotes the processing time of configuration C in x_{pre}. Moreover, $\mathcal{C}_{pre,W}$ is the set of configurations from $\mathcal{C}_{pre}$ containing wide jobs exclusively. We define $\mathcal{W}_{pre} := \{w(C) | C \in \mathcal{C}_{pre,W}\}$ as the set of main windows for $\mathcal{C}_{pre,W}$ and $P_{pre} := \sum_{C \in \mathcal{C}_{pre}} (x_{pre})_C$ as the makespan of the preemptive schedule.

LEMMA 8.6. *Given a solution* x_{pre}*, we can generate a solution* $(\tilde{x}, \tilde{y})$ *to* $LP_W(I, \mathcal{C}_{pre,W}, \mathcal{W}_{pre})$ *such that*

$$P(\tilde{x}) = P_{pre} \tag{8.12}$$

Proof. To generate this solution, we simply look at each configuration $C \in \mathcal{C}_{pre,W}$ and sum up the processing time of each configuration $C' \in \mathcal{C}_{pre}$, which is reduced to C, meaning $C'|_{\mathcal{J}_W} = C$. Building a generalized configuration, we combine C with its main window $w(C) \in \mathcal{W}_{pre}$. Hence we define

$$\tilde{x}_{(C,w(C))} := \sum_{\substack{C' \in \mathcal{C}_{pre} \\ C'|_{\mathcal{J}_W} = C}} (x_{pre})_{C'}.$$

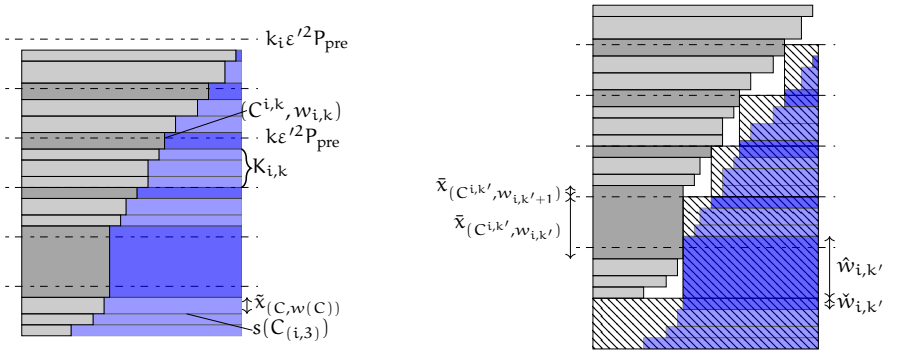
Equality (8.12) holds for this choice for $\tilde{x}$ since the procesing time of each configuration is added to exactly one generalized configuration. With a similar argument one can see that Inequality (8.6) holds because x_{pre} fulfills Inequality (8.4). Now, we have to ensure that Inequalities (8.7) to (8.9) hold. For this purpose, we look at each configuration $C \in \mathcal{C}_{pre}$ and consider the reduced configuration $C|_{\mathcal{J}_W}$ and its main window $w := w(C|_{\mathcal{J}_W})$. For each job $j \in \mathcal{J}_N$, we add its processing time in C, that is, $C(j)(x_{pre})_C$, to the window w. In total, we get for each window $w \in W$ and each job $j \in \mathcal{J}_N$

$$\tilde{y}_{j,w} := \sum_{\substack{C \in \mathcal{C}_{pre} \\ w(C|_{\mathcal{J}_W}) = w}} C(j)(x_{pre})_C.$$

Since the configuration C was valid and Equality (8.4) holds for x_{pre}, the Equalities (8.7) to (8.9) hold for $(\tilde{x}, \tilde{y})$. □

Reducing the Number of Configurations – Step (v)

At this point, we already have reduced the number of used (generalized) configurations: Each wide job has a width of at least $\varepsilon' R$ and they have at most $1/\varepsilon'^2$ different sizes. Therefore, the configuration part of each introduced generalized configuration has one of at most $(\frac{1}{\varepsilon'^2})^{1/\varepsilon'}$ different widths. Since in each introduced generalized configuration the configuration part is paired with the corresponding main window (which is the same for configuration parts with the same width and number of used machines), we used at most $(\frac{1}{\varepsilon'^2})^{1/\varepsilon'}$ different generalized configurations. This number is independent of the input size. However, this value is still to large since it is exponential in $1/\varepsilon'$, and if $|\mathcal{J}_{sup}| < (\frac{1}{\varepsilon'^2})^{1/\varepsilon'}$, not even a better bound on the number of generated configurations (the number of generalized configurations in $(\tilde{x}, \tilde{y})$ is bounded by $\min\{|\mathcal{J}_{sup}| + 1, (\frac{1}{\varepsilon'^2})^{1/\varepsilon'}\}$). To reduce the number of generalized configurations, we round the resource amounts of the windows, i.e., the widths of the windows, in the next step. We define $P_{pre}(C) := \tilde{x}_{C,w(C)}$ for each $C \in \mathcal{C}_W$ and $P_{pre}(K) := \sum_{C \in K} P_{pre}(C)$ for each $K \subseteq \mathcal{C}_W$.



(a) A stack of the generalized configurations in K_i. The grey rectangles represent configurations and the blue rectangles windows. The dashed lines are multiples of $\varepsilon'^2 P_{pre}$.

(b) The same stack of the generalized configurations as in Figure 8.2a. But this time the windows are shifted $\varepsilon'^2 P_{pre}$ downwards. One can see that their area fits into the new windows (hatched rectangles).

Figure 8.2: Shifting the windows

LEMMA 8.7. *Given a solution* $(\tilde{x}, \tilde{y})$ *to* $LP_W(I, \mathcal{C}_W, \mathcal{W}_{pre})$*, we can find a set* $\mathcal{W}' \subseteq \mathcal{W}_{pre}$ *with* $|\mathcal{W}'| \leqslant \varepsilon'^{-2} + 2$ *and a corresponding solution* $(\bar{x}, \bar{y})$ *to* $LP_W(I, \mathcal{C}_W, \mathcal{W}')$ *which fulfills*

$$P(\bar{x}) \leqslant (1 + \varepsilon')P_{pre} \tag{8.13}$$

and contains at most $|\mathcal{J}_N| + |\mathcal{J}_W| + 2|\mathcal{W}'| + 1$ *non-zero components with* $\mathcal{O}((|\mathcal{J}| + |\mathcal{W}'|)^{1.5356}|\mathcal{J}||\mathcal{W}'|)$ *operations.*

Proof. The steps to find $\mathcal{W}'$ and $(\bar{x}, \bar{y})$ are described in the following. The sets, configurations, and sizes defined for these steps can also be found in the Figures 8.2a and 8.2b.

To find the set $\mathcal{W}'$, we reduce the number of window resource amounts by a further grouping step. We partition the set of generalized configurations by the number of machines in the window. For each $i \in \{1, \ldots, m\}$, we define $K_i := \{C \in \mathcal{C}_W | m(C) = i\}$ to denote the set containing all configurations using exactly i machines. Since at most $1/\varepsilon'$ wide jobs can be part of one configuration, only the sets K_1 to $K_{1/\varepsilon'}$ are not empty. For each of these sets we apply linear grouping: We number the configurations in K_i such that $R(C_{i,1}) \leqslant R(C_{i,2}) \leqslant R(C_{i,3}) \ldots$ holds. Next, we stack the configurations defining start positions $s(C_{i,1}) := 0$ and $s(C_{i,j}) := s(C_{i,j-1}) + P_{pre}(C_{i,j-1})$ for each $j > 1$. Furthermore, we define a configurations end position $e(C_{i,j}) := s(C_{i,j+1})$. The summed up height of all stacks is P_{pre}. We want to get about ε'^{-2} windows. Hence, we split our stacks in ε'^{-2} pieces total and consider in each stack the multiples of $\varepsilon'^2 P_{pre}$ to group the windows.

For each $0 < i \leqslant 1/\varepsilon'$, we define $k_i := \lceil P_{pre}(K_i)/(\varepsilon'^2 P_{pre}) \rceil$ which is the first multiple of $\varepsilon'^2 P_{pre}$ that does not intersect a generalized

configuration in the i-th stack since it has a height of at most $P_{pre}(K_i)$. Note that k_i might touch the last configuration if

$$\lceil P_{pre}(K_i)/(\varepsilon'^2 P_{pre})\rceil = P_{pre}(K_i)/(\varepsilon'^2 P_{pre}),$$

but we allow this. For each $k \in \{1,\dots,k_i-1\}$, there is a configuration C which intersects with $k\varepsilon'^2 P_{pre}$, i.e., $s(C) < k\varepsilon'^2 P_{pre} \leqslant s(C) + P_{pre}(C)$. We denote this configuration by $C^{i,k}$ and define $w_{i,k} := w(C^{i,k})$ and $w_{i,k_i} := (0,0)$. Furthermore, we define the set $K_{i,k}$ as the set of configurations lying between the configurations $C^{i,k-1}$ and $C^{i,k}$. The set $\mathcal{W}'$ is defined such that it contains all the windows intersected by a multiple of $\varepsilon'^2 P_{pre}$ and two default windows. More precisely, we define

$$\mathcal{W}_{K_i} := \{w_{i,k} | k \in \{1,\dots,k_i-1\}\}$$

as the set of chosen windows from K_i and furthermore:

$$\mathcal{W}' := \bigcup_{i=1}^{1/\varepsilon'} \mathcal{W}_{K_i} \cup \{(0,0),(R,m)\}$$

It holds that $|\mathcal{W}_{K_i}| \leqslant \lfloor P_{pre}(K_i)/(\varepsilon'^2 P_{pre})\rfloor$ and therefore:

$$|\mathcal{W}'| \leqslant 2 + \sum_{i=1}^{1/\varepsilon'} \left\lfloor \frac{P_{pre}(K_i)}{\varepsilon'^2 P_{pre}} \right\rfloor \leqslant 2 + \frac{1}{\varepsilon'^2}$$

After choosing the set of windows $\mathcal{W}'$, the next step is to generate a solution to the $LP_{\mathcal{W}}$ which exclusively uses windows in $\mathcal{W}'$ and needs not much further processing time. The crucial step is to shift the windows in each stack exactly $\varepsilon'^2 P_{pre}$ downwards.

Let us consider the configurations $C^{i,k}$ and $C^{i,k-1}$. The resource amount of $C^{i,k-1}$ is less or equal to the resource amount of $C^{i,k}$. So, for each configuration $C \in K_{i,k}$, we have $w(C) \geqslant w_{i,k}$. If we shift the windows $\varepsilon'^2 P_{pre}$ downwards, the window $w_{i,k}$ is now processed alongside $C^{i,k-1}$. The windows above $w_{i,k}$ have a lesser resource amount. We now round up the resource amount of the windows alongside the configurations in $C \in K_{i,k}$ such that they have the same resource amount as $w_{i,k}$. We therefore define for each $1 \leqslant 1/\varepsilon'$, for all $k \leqslant k_i$, and for each $C \in K_{i,k}$: $\bar{x}_{(C,w_{i,k})} := \tilde{x}_{(C,w(C))}$

With configuration $C^{i,k}$ we have to be more careful. The lower part of this configuration should be alongside window $w_{i,k}$ and the other alongside $w_{i,k+1}$. We have to find the biggest multiple of $\varepsilon'^2 P_{pre}$ the configuration $C^{i,k}$ was intersected by. In case $P(C^{i,k}) > \varepsilon'^2 P_{pre}$, the configuration $C^{i,k}$ will be intersected by more than one multiple of $\varepsilon'^2 P_{pre}$. The biggest multiple is defined by $\varepsilon'^2 P_{pre} \lfloor \frac{e(C^{i,k})}{\varepsilon'^2 P_{pre}} \rfloor$. This multiple defines the height at which the configuration is split. We define for each $i \leqslant 1/\varepsilon'$ and for each $k \leqslant k_i$:

$$\bar{x}_{(C^{i,k},w_{i,k})} := \varepsilon'^2 P_{pre} \left\lfloor \frac{e(C^{i,k})}{\varepsilon'^2 P_{pre}} \right\rfloor - s(C^{i,k})$$

$$\bar{x}_{(C^{i,k},w_{i,k+1})} := \tilde{x}_{(C^{i,k},w_{i,k})} - \bar{x}_{(C^{i,k},w_{i,k})}$$

Finally, we need some extra space for the windows which were shifted below the lowest configuration. In each stack, it concerns window parts of total height $\varepsilon'^2 P_{pre}$. Since we have $1/\varepsilon'$ stacks, we need $\varepsilon' P_{pre}$ extra space. We round these windows up to the window (R, m). Hence, the configuration $(\emptyset, (R, m))$ is the configuration which is extended in height. We define $\bar{x}_{(\emptyset,(R,m))} := \tilde{x}_{(\emptyset,(R.m))} + \varepsilon' P_{pre}$.

In the following, we describe how to assign the narrow jobs to the round up windows. Let us consider a configuration $C \in K_{i,k+1}$. The window $w(C)$ was shifted down such that it can be rounded up to $w_{i,k}$. To round this window up, we have to know which amount of it was processed alongside C. This amount is given by $\varphi_C := P_{pre}(C)/P_{pre}(w(C))$. So, for each configuration $C \in K_{i,k+1}$ and each job $j \in \mathcal{J}_N$, we add $\varphi_C \tilde{y}_{j,w(C)}$ processing time to $\bar{y}_{j,w_{i,k}}$.

Next, we consider a window $w_{i,k}$. In general, this has to be split: as one can see in Figure 8.2b the upper part of window $w_{i,k}$ stays in this window while the lower part is put into window $w_{i,k-1}$. This time we have to split the window at the smallest multiple of $\varepsilon'^2 P_{pre}$. This multiple is defined by $\varepsilon'^2 P_{pre} \lceil s(C^{i,k})/\varepsilon'^2 P_{pre} \rceil$. Again, we have to know which amount of window $w_{i,k}$ has to be processed where. Let $\check{w}_{i,k} = \varepsilon'^2 P_{pre} \lceil s(C^{i,k})/\varepsilon'^2 P_{pre} \rceil - s(C^{i,k})$ be the processing time of window $w_{i,k}$ which hast to be scheduled in the window $w_{i,k-1}$, and let $\hat{w}_{i,k} = \tilde{x}_{(C^{i,k},w_{i,k})} - \check{w}_{i,k}$ be the processing time of window $w_{i,k}$ which can stay in $w_{i,k}$. Furthermore, we have to know which fraction of the total processing time of $w_{i,k}$ is represented by these two values. We define $\hat{\varphi}_{i,k} := \hat{w}_{i,k}/P_{pre}(w_{i,k})$ and $\check{\varphi}_{i,k} := \check{w}_{i,k}/P_{pre}(w_{i,k})$. We now know that we have to add $\hat{\varphi}_{i,k}\tilde{y}_{j,w_{i,k}}$ to $\bar{y}_{j,w_{i,k}}$ and $\check{\varphi}_{i,k}\tilde{y}_{j,w_{i,k}}$ to $\bar{y}_{j,w_{i,k-1}}$. In total we have

$$\bar{y}_{j,w_{i,k}} := \sum_{C \in K_{i,(k+1)}} \varphi_C \tilde{y}_{j,w(C)} + \hat{\varphi}_{i,k}\tilde{y}_{j,w_{i,k}} + \check{\varphi}_{i,k+1}\tilde{y}_{j,w_{i,k+1}}$$

for each $i \leqslant 1/\varepsilon'$, $k \leqslant k_i$ and $j \in \mathcal{J}_N$.

We consider the window (R, m) separately. First, we have to round the main windows from all configurations in $K_{i,1}$ up to (R, m). More precisely, for each $i \leqslant 1/\varepsilon'$, $C \in K_{i,1}$ and $j \in \mathcal{J}_N$, we add $\varphi_C \tilde{y}_{j,w(C)}$ to $\bar{y}_{j,(R,m)}$. Furthermore, we have to round up the lower part of window $w_{i,1}$ to (R, m). Additional jobs which were processed in window (R, m) before stay there. So, in total we have

$$\bar{y}_{j,(R,m)} := \tilde{y}_{j,(R,m)} + \sum_{i=1}^{1/\varepsilon'} \left(\check{\varphi}_{i,1}\tilde{y}_{j,w_{i,1}} + \sum_{C \in K_{i,1}} \varphi_C \tilde{y}_{j,w(C)} \right)$$

for each $j \in \mathcal{J}_N$. Therefore, $(\bar{x}, \bar{y})$ is a solution to $LP_W(\mathcal{W}')$. Using suitable data structures, $(\tilde{x}, \tilde{y})$ as well as $(\bar{x}, \bar{y})$ can be computed in $\mathcal{O}((m + \log(n)/\varepsilon)n)$ operations. This linear program has $|\mathcal{J}_N| +$

$|\mathcal{J}_{\sup \mathcal{W}}| + 2|\mathcal{W}'| \in \mathcal{O}(|\mathcal{J}| + |\mathcal{W}'|)$ constraints and at most $|\mathcal{C}_{\mathcal{W}}||\mathcal{W}'| + |\mathcal{J}_N||\mathcal{W}'| \in \mathcal{O}(|\mathcal{J}||\mathcal{W}'|)$ variables. So we can compute a solution with at most $|\mathcal{J}_N| + |\mathcal{J}_{\sup \mathcal{W}}| + 2|\mathcal{W}'| + 1$ non-zero components in $\mathcal{O}((|\mathcal{J}| + |\mathcal{W}'|)^{1.5356}|\mathcal{J}||\mathcal{W}'|)$ operations. □

In Step (v), the algorithm uses the results from Lemma 8.7 to find a solution $(\bar{x}, \bar{y})$ to $LP_{\mathcal{W}}$ which uses at most

$$|\mathcal{J}_N| + |\mathcal{J}_{\mathcal{W}}| + 2|\mathcal{W}'| + 1 \leqslant |\mathcal{J}_N| + 3/\varepsilon'^2 + 5$$

non-zero components in $\mathcal{O}((|\mathcal{J}_{\sup}| + |\mathcal{W}'|)^{1.5356}|\mathcal{J}_{\sup}||\mathcal{W}'|)$ operations. Since $|\mathcal{W}'| \in \mathcal{O}(1/\varepsilon^2)$, the total running time of the algorithms from Step (i) to (v) can be bounded by $\mathcal{O}(|\mathcal{J}_{\sup}|(\ln(|\mathcal{J}_{\sup}|) + \varepsilon^{-2})(|\mathcal{J}_{\sup}|^{1.5356} + m/\varepsilon^4))$.

Integral Solution – Step (vi)

We now generate an integral schedule of the jobs in $\mathcal{J}$. The used technique to schedule the wide jobs is similar to the technique used by Kenyon and Rémila [94] to place the wide rectangles into their fractional packing of rectangles. To place the narrow jobs, we use a similar argument as Epstein and Levin in [45].

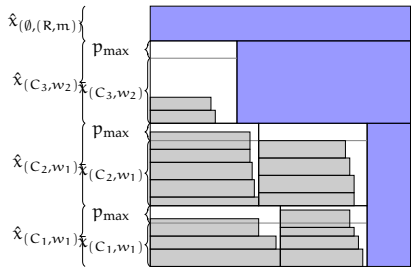
We say a narrow job in $\mathcal{J}_N$ is scheduled fractionally if it is assigned to more than one window Hence, each fractionally scheduled job corresponds to at least two non-zero components. Note that $(\bar{x}, \bar{y})$ has at most $|\mathcal{J}_N| + 3/\varepsilon'^2 + 5$ non-zero components. Since each job in $\mathcal{J}_N$ needs one none zero component to be scheduled, there are at most $3/\varepsilon'^2 + 5$ non-zero components left for configurations or narrow jobs. Hence, $(\bar{x}, \bar{y})$ contains at most $3/\varepsilon'^2 + 5$ fractionally scheduled jobs and configurations in total.

LEMMA 8.8. *Let $(\bar{x}, \bar{y})$ be a basic solution to $LP_{\mathcal{W}}(I_{\sup}, \mathcal{C}_{\mathcal{W}}, \mathcal{W}')$ such that the total number of fractionally scheduled narrow jobs and used configurations is bounded by K. There is a solution which places the jobs in $\mathcal{J}$ integrally and has a makespan of at most*

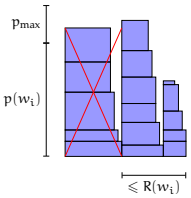
$$(1 + \varepsilon')P(\bar{x}) + (1 + \varepsilon'^{-1} + K)p_{\max}.$$

Proof. First, we modify the solution $(\bar{x}, \bar{y})$ to have enough space to schedule the jobs integrally: We define $\hat{x}_{(C,w)} := \bar{x}_{(C,w)} + p_{\max}$ if $\bar{x}_{(C,w)} > 0$, and $\hat{x}_{(C,w)} = 0$ otherwise. We denote by $\mathcal{J}_{frac} \subseteq \mathcal{J}_N$ the set of fractionally scheduled narrow jobs in $(\bar{x}, \bar{y})$. For each $j \in \mathcal{J}_{frac}$ and each window $w \in \mathcal{W}$, we set $\hat{y}_{j,w} := 0$ and $\hat{y}_{j,(R,m)} := p_j$. Furthermore, we add $P(\mathcal{J}_{frac}) \leqslant p_{\max}|\mathcal{J}_{frac}|$ to $\hat{x}_{(\emptyset,(R,m))}$. For each remaining $j \in \mathcal{J}_N \setminus \mathcal{J}_{frac}$, we set $\hat{y}_{j,w} = \bar{y}_{j,w}$. Moreover, for each fractional job and for each configuration, we add at most $p_{\max}$ processing time. Hence, we have $P(\hat{x}) \leqslant P(\bar{x}) + Kp_{\max}$.

Next, we place the wide jobs. We sort the generalized configurations such that configurations with the same window are scheduled consecutively and number them in ascending order. We iterate over each



(a) Three generalized configurations and the configuration $\hat{x}_{(\emptyset,(R,m))}$. Wide jobs are filled in the generalized configurations. The windows still need to be filled.



(b) Stacks of small jobs assigned to a window w_i. The first stack is discarded and positioned at the end of the schedule.

Figure 8.3: Integral placement of the jobs into the generalized configurations

$i < G$, i.e., over all groups, as well as all generalized configurations and fill the spaces for the job $i \in \mathcal{J}_{W,sup}$ which has a resource amount of R_i with jobs from $\mathcal{J}_{W,i} \cap \mathcal{J}_W$ until the height of the configuration $\bar{x}_{(C,w)}$ is reached. Each last job in a configuration is allowed to overlap the border $\bar{x}_{(C,w)}$ since we have added p_{max} extra space. All jobs from $\mathcal{J}_{W,i} \cap \mathcal{J}_W$ can be placed in the configurations. Furthermore, there is one generalized configuration (C, w) where the border $\bar{x}_{(C,w)}$ is not overlapped by a job from $\mathcal{J}_{W,i}$ since $\sum_{(C,w)\in\tilde{\mathcal{C}}} C(i)\bar{x}_{(C,w)} \geqslant P(\mathcal{J}_{W,i})$. In this particular configuration, we place the job which was intersected by the multiple of $\varepsilon'^2 P_W$ in the first linear grouping and which fractions were put in $\mathcal{J}_{W,i}$ as well as in $\mathcal{J}_{W,i+1}$. Since one of its fractions was in $\mathcal{J}_{W,i}$, it fits into the reserved space.

To place the small jobs, we consider each window $w \in \mathcal{W}$. Let $\mathcal{J}(w)$ be the set of jobs contained in w. We order the jobs in $\mathcal{J}(w)$ by decreasing resource amount. Since the generalized configurations containing window w are scheduled consecutively, we can build stacks of jobs form $\mathcal{J}(w)$ with total processing time between $P(w, \bar{x})$ and $P(w, \bar{x}) + p_{max}$ and schedule them in window w. Since $m(w)P(w, \bar{x}) \geqslant \sum_{j\in\mathcal{J}_N} \hat{y}_{j,w}$, we have to build at most $m(w)$ of these stacks. Because we have $m(w)$ free machines in each window, there is a free machine for each stack.

Consider the jobs $\mathcal{J}_{top}$ intersecting the bound $P(w, \bar{x})$. These are the jobs with the lowest resource amount of their stack. Hence, at each point in time, the small jobs in the stacks have a total resource amount of at least $R(\mathcal{J}_{top})$ and therefore $P(w, \bar{x})R(\mathcal{J}_{top}) \leqslant \sum_{j\in J(w)} r_j p_j \leqslant P(w, \bar{x})R(w)$. Thus, the jobs in $\mathcal{J}_{top}$ fit into the window w.

We remove the stack with the largest resource amount. Then the summed up resource amount of the jobs at the bottom of the stacks is less than $R(\mathcal{J}_{top})$ since each of these jobs was added right after a job from $\mathcal{J}_{top}$. Therefore, at each point in time, in window w it holds that the resource amount of small jobs is at most $R(\mathcal{J}_{top})$.

The removed stack has a total processing time of at most $P(w, \bar{x}) + p_{max}$. Since this stack is removed in each window, jobs with summed up processing time at most $\sum_{w \in \mathcal{W} \setminus \{(0,0),(R,m)\}} (P(w, \bar{x}) + p_{max}) \leqslant P(\bar{x}) + \varepsilon'^{-2} p_{max}$ have to be placed at the end of the schedule. Since each of these jobs has a resource amount of at most $\varepsilon' R$, we can schedule $1/\varepsilon'$ of them at the same time. Hence, we need an additional processing time of $\varepsilon' P(\bar{x}) + (1 + \varepsilon'^{-1}) p_{max}$. In total, we get a schedule of makespan:

$$\begin{aligned} &\varepsilon' P(\bar{x}) + (1 + \varepsilon'^{-1}) p_{max} + P(\bar{x}) + K p_{max} \\ &= (1 + \varepsilon') P(\bar{x}) + (1 + \varepsilon'^{-1} + K) p_{max} \end{aligned}$$

□

The solution we get from Lemma 8.7 has at most $|\mathcal{J}_N| + |\mathcal{J}_{\sup \mathcal{W}}| + 2|\mathcal{W}'| + 1 \leqslant |\mathcal{J}_N| + 3/\varepsilon'^2 + 5$ non-zero components. Since each job $j \in \mathcal{J}_N$ needs a non-zero component to be scheduled, we have at most $3/\varepsilon'^2 + 5$ configurations and fractional jobs. Therefore, the generated integral schedule has a makespan of:

$$\begin{aligned} &(1 + \varepsilon') P(\bar{x}) + (1 + \varepsilon'^{-1} + 3/\varepsilon'^2 + 5) p_{max} \\ &\leqslant (1 + \varepsilon')^3 \operatorname{OPT}_{pre}(I_{sup}) + \mathcal{O}(1/\varepsilon^2) p_{max} \\ &\leqslant (1 + \varepsilon')^4 \operatorname{OPT}_{pre}(I) + \mathcal{O}(1/\varepsilon^2) p_{max} \\ &\leqslant (1 + \varepsilon) \operatorname{OPT}(I) + \mathcal{O}(1/\varepsilon^2) p_{max} \end{aligned}$$

The number of operations to build the integral schedule is dominated by the number of operations to build the first preemptive schedule, that is, $\mathcal{O}(n(\ln(n) + \varepsilon^{-2})(n^{1.5356} + m\varepsilon^{-4}))$. Since each step we used to manipulate x_{pre} was dominated by this number, this is our total processing time. Note that in the above process resources can be allocated contiguously. Hence, this schedule is feasible for instances where contiguous resource is required as well.

8.2.2 *Second Case:* $m \leqslant 1/\varepsilon$

Let $\varepsilon > 0$ and an instance $I = (\mathcal{J}, m, R)$ with $1/\varepsilon \geqslant m$ be given. In this case, we do not partition the set of jobs into wide and narrow jobs. Instead we apply the linear grouping to all jobs. Again, we get $G \leqslant 1/\varepsilon'^2$ groups $\mathcal{J}_i$. We denote by $\mathcal{J}_{sup}$ the set of jobs which contains for each $i < G$ one job with processing time $P(\mathcal{J}_i)$ and resource amount $\max\{r_j | j \in \mathcal{J}_i\}$. We denote by $\mathcal{C}_{sup}$ the set of valid configurations of jobs in $\mathcal{J}_{sup}$. We denote by $I_{sup} := (\mathcal{J}_{sup}, m, R)$ the round up instance. To get a preemptive schedule for I_{sup} we have to solve $LP_{pre}(I_{sup})$ while interpreting each job in $\mathcal{J}_{sup}$ as a wide job. By Lemma 8.3, we get a solution x_{pre} with $\mathcal{O}(|\mathcal{J}_{sup}|(\ln(|\mathcal{J}_{sup}|) + \varepsilon'^{-2})) = \mathcal{O}(\varepsilon'^{-4})$ non-zero components and $\sum_{C \in \mathcal{C}_{sup}} (x_{pre})_C \leqslant (1 + \varepsilon') \operatorname{OPT}_{pre}$ in $\mathcal{O}(|\mathcal{J}_{sup}|(\ln(|\mathcal{J}_{sup}|) + 1/\varepsilon'^2)(|\mathcal{J}_W| z + |\mathcal{J}_N| + m z'^2/\varepsilon'^2)) = \mathcal{O}(m^3/\varepsilon'^6) = \mathcal{O}(1/\varepsilon'^9)$ operations.

Since we now have a polynomial number of variables, we can construct a solution $\tilde{x}$ which has at most $1/\varepsilon'^2$ non-zero components and for which it holds that $\sum_{C \in \mathcal{C}_{sup}} (x_{pre})_C = \sum_{C \in \mathcal{C}_{sup}} \tilde{x}_C$. This takes $\mathcal{O}(|\mathcal{J}_{sup}|^{2.5356}(\ln(|\mathcal{J}_{sup}|) + \varepsilon'^{-2})) = \mathcal{O}(1/\varepsilon'^{4.5356})$ operations.

To each of these components we add p_{max} processing time. Next, we place the jobs from $\mathcal{J}_i$ for each $i < G$ in the configurations as we placed the wide jobs in the first case. We get a schedule of length at most $(1+\varepsilon')\,\mathrm{OPT}_{pre} + \varepsilon'^{-2} p_{max}$. The jobs in $\mathcal{J}_G$ are added in the end of the schedule. These jobs have a total makespan of at most $\varepsilon'\,\mathrm{OPT}_{pre}(I)$ since $P(\mathcal{J}_G) \leqslant \varepsilon'^2 P(\mathcal{J})$ and $\mathrm{OPT}_{pre}(I) \geqslant P(\mathcal{J})/m \geqslant \varepsilon' P(\mathcal{J})$. So, in the end we have a schedule with a total makespan of at most

$$\begin{aligned}(1+\varepsilon')\,\mathrm{OPT}_{pre}(I) + \varepsilon'\,\mathrm{OPT}_{pre}(I) + 1/\varepsilon'^2 p_{max} \\ \leqslant (1+\varepsilon)\,\mathrm{OPT}(I) + \mathcal{O}(1/\varepsilon^2) p_{max}.\end{aligned}$$

8.2.3 *Improving the additive term*

It is possible to reduce the additive term using different rounding strategies. To do this, we have to consider other cases than before: $m > 1/\varepsilon'^2$ and $m \leqslant 1/\varepsilon'^2$.

Let $m > 1/\varepsilon'^2$. In the described algorithm, we have two rounding steps. One for the wide jobs and one for the windows for narrow jobs. To achieve a better additive term, we have to modify both of them. First, we describe how to round the wide jobs. Previously, we used linear grouping to round the wide jobs. This rounding yielded ε'^{-2} different item sizes. To improve the additive term, we have to reduce this number. Karmakar and Karp [90] have introduced a second rounding strategy called geometric grouping, and this technique was refined for strip packing in [21] and [138]. We describe how to apply the geometric grouping to the wide jobs:

LEMMA 8.9. *For every $\varepsilon > 0$ with $1/\varepsilon \in \mathbb{N}$ and every instance I with a set of wide jobs $\mathcal{J}_{W,\varepsilon}$, we can reduce the number of different resource amounts of the wide jobs to $\lceil \log(\varepsilon^{-1}) \rceil \varepsilon^{-1}$ by extending the optimal preemptive schedule $\mathrm{OPT}_{pre}(I, \varepsilon)$ by at most $2\varepsilon\,\mathrm{OPT}_{pre}(I, \varepsilon)$. The resulting rounded instance is called $I'_{sup,\varepsilon'}$.*

Proof. The idea is to partition the set $\mathcal{J}_W$ into sets $\mathcal{J}'_{W,i} := \{j \in \mathcal{J}_W | R/2^{i+1} \leqslant r_j < R/2^i\}$. Since each item in $\mathcal{J}_W$ has a resource amount of at least εR, we have at most $\lceil \log(\varepsilon^{-1}) \rceil$ of these sets. Next, we round the items in each of these sets by linear grouping into at most ε^{-1} different sizes. This is done in the following way: We stack the items of the set $\mathcal{J}'_{W,i}$ in ascending order of their width. We then partition the stack in sets of height $\varepsilon P(\mathcal{J}'_{W,i})$ and split the items cut by a horizontal line. Let $\mathcal{J}'_{W,i,j}$ be the set containing the items between the lines $\varepsilon P(\mathcal{J}'_{W,i-1})$ and $\varepsilon P(\mathcal{J}'_{W,i})$. We round the resource amount of all jobs in $\mathcal{J}'_{W,i,j}$ to the widest resource amount which can be found in this set.

Consider a solution of $\text{OPT}_{\text{pre}}(I, \varepsilon)$. We can replace each job from group $\mathcal{J}'_{W,i,j}$ by rounded jobs from group $\mathcal{J}'_{W,i,j+1}$. The widest group in each stack is scheduled at the end of the solution. Since the resource amount of each job in this group is smaller than $R/2^i$, we are able to schedule 2^i of them at the same point in time. Since the height of this group is at most $\varepsilon P(\mathcal{J}'_{W,i})$, we need at most $(1/2^i) \cdot \varepsilon P(\mathcal{J}'_{W,i})$ processing time to schedule this set of items fractionally.

On the other hand, we know that we can schedule at most 2^{i+1} items from $\mathcal{J}'_{W,i}$ at the same time since their resource amount is at least $R/2^{i+1}$. So it holds that:

$$\sum_{i=0}^{\lceil \log(\varepsilon'^{-1}) \rceil - 1} P(\mathcal{J}'_{W,i})/2^{i+1} \leqslant \text{OPT}_{\text{pre}}(I)$$

Let us now consider the total processing time we have to add at the end of the schedule: For each of the $\lceil \log(\varepsilon'^{-1}) \rceil$ sets, we add the last group at the end of the schedule, i.e., at most $(1/2^i) \cdot \varepsilon' P(\mathcal{J}'_{W,i})$ processing time per set. Hence, we add at most:

$$\begin{aligned} &\sum_{i=0}^{\lceil \log(\varepsilon'^{-1}) \rceil - 1} (1/2^i) \cdot \varepsilon' P(\mathcal{J}'_{W,i}) \\ &= 2\varepsilon' \sum_{i=0}^{\lceil \log(\varepsilon'^{-1}) \rceil - 1} \frac{1}{2^{i+1}} P(\mathcal{J}'_{W,i}) \leqslant 2\varepsilon' \,\text{OPT}_{\text{pre}}(I) \end{aligned}$$

In each of the sets we have ε'^{-1} groups. So in total we now have $\lceil \log(\varepsilon'^{-1}) \rceil \varepsilon'^{-1}$ different resource amounts for wide jobs and we have lengthened the preemptive schedule by at most $2\varepsilon' \,\text{OPT}_{\text{pre}}(I)$. □

In the second step, we modify the rounding of the resource amounts of the windows. We are able to reduce the number of different windows to $1/\varepsilon' + 1$:

LEMMA 8.10. *Let $m \geqslant 1/\varepsilon'^2$. Given a solution $(\tilde{x}, \tilde{y})$ to the linear program $\text{LP}_W(I, \mathcal{C}_W, \mathcal{W}_{\text{pre}})$, we can find a set $\mathcal{W}' \subseteq \mathcal{W}_{\text{pre}}$ with $|\mathcal{W}'| \leqslant 1/\varepsilon' + 1$ and a solution $(\bar{x}, \bar{y})$ to $\text{LP}_W(I, \mathcal{C}_W, \mathcal{W}')$ with*

$$P(\bar{x}) \leqslant (1 + 2\varepsilon')P(\tilde{x}), \tag{8.14}$$

and that contains at most $|\mathcal{J}_N| + |\mathcal{J}_W| + 2|\mathcal{W}'| + 1$ non-zero components. This can be done with $\mathcal{O}((|\mathcal{J}| + |\mathcal{W}'|)^{1.5356}|\mathcal{J}||\mathcal{W}'|)$ operations.

Proof. In the first step, we modify all windows except the window (R, m) such that they use at most $m - 1/\varepsilon'$ machines. This allows us to neglect the number of machines when we are rounding the windows resource amount since each configuration of wide jobs needs at most $1/\varepsilon'$ machines. Let us look at a window w. Let J_w be the set of job parts scheduled in w, more precisely, for each job in $\mathcal{J}_N$ with $y_{j,w} > 0$ there is a job in J_w with processing time $y_{j,w}$ and resource amount

r_j. We sort the jobs in J_w by decreasing resource amount and build stacks of height $P(w)$ exactly, similar as in the generation of an integral solution, but this time we allow jobs to be split horizontally. We now look at the number of stacks we got. If we have less than $m - 1/\varepsilon'$ stacks, we set w_m to $m - 1/\varepsilon'$ without violating the Equation (8.8). If we have $m - x$ stacks with $x \in \{0, \dots, 1/\varepsilon' - 1\}$, we remove the $1/\varepsilon' - x$ stacks with the smallest resource amount, set w_m to $m - 1/\varepsilon'$ without violating the Equation (8.8), and place them into window (R, m). Let $\mathcal{J}_{\text{shift}}$ be the set of (fractional) jobs we moved to window (R, m).

Claim 8.11. If we schedule the jobs $\mathcal{J}_{\text{shift}}$ in window (R, m), we have to add at most $\varepsilon' P(\tilde{x})$ processing time to $\tilde{x}_{(\emptyset,(R,m))}$ to get a feasible solution of LP_W.

The window (R, m) is the only one where we have added jobs. We have to show $m\varepsilon' P(\tilde{x}) \geqslant P(\mathcal{J}_{\text{shift}})$ and that $R\varepsilon' P(\tilde{x}) \geqslant \sum_{j \in \mathcal{J}_{\text{shift}}} r_j p_j$ to prove that the Inequalities (8.8) and (8.9) hold which then proves the claim.

Let us look at the stacks of jobs we removed in window w. We show that the summed up resource amount of the jobs at the bottom of these stacks is at most $\varepsilon' R$. To see this, notice that the summed up resource amount of the widest jobs in each stack except for the widest stack has a resource amount of at most $R(w)$. Otherwise, the resource amount of all the jobs in the window would be larger than $R(w)$ which would contradict the fact that Equality (8.9) holds for the considered solution. Furthermore, we know that $R(w) \leqslant R - \varepsilon' R$ since the considered window is not the window (R, m), and hence there has to be a wide job scheduled at the same time as the window.

Since the job at the bottom of the widest stack has a resource amount of less than $\varepsilon' R$, the summed resource amount of all stacks is strictly smaller than R. Assume that the jobs at the bottom of the $1/\varepsilon' - x$ smallest stacks have a summed resource amount of more than $\varepsilon' R$. In this case, one of these jobs has a resource amount of at least $\varepsilon' R/(\varepsilon'^{-1} - x) \geqslant \varepsilon'^2 R$. All of the at least $m - 1/\varepsilon'$ other wider stacks have a job with resource amount of at least $\varepsilon'^2 R$ at the bottom. Therefore, the total resource amount of the jobs at the bottom of the stacks is at least $(m - 1/\varepsilon')\varepsilon'^2 R + \varepsilon' R \geqslant (1/\varepsilon'^2 - 1/\varepsilon')\varepsilon'^2 R + \varepsilon' R = R$ which is a contradiction since the total resource amount of these stacks is strictly less than R.

Let $\mathcal{J}_{\text{shift},w}$ be the set of jobs in the stacks we remove from w. Since the summed up resource amount of the jobs at the bottom of these stacks is at most $\varepsilon' R$, we know that $\varepsilon' R \cdot P(w) \geqslant \sum_{j \in \mathcal{J}_{\text{shift},w}} r_j p_j$. The total processing time of these stacks is at most $\sum_{j \in \mathcal{J}_{\text{shift},w}} p_j = P(\mathcal{J}_{\text{shift},w}) \leqslant x \cdot P(w) \leqslant 1/\varepsilon' \cdot P(w) \leqslant m\varepsilon' P(w)$ since $m \geqslant 1/\varepsilon'^2$. If we sum up the changes for all windows, we get:

$$R\varepsilon' P(\tilde{x}) \geqslant \sum_{w \in \mathcal{W} \setminus (R,m)} R\varepsilon' P(w) \geqslant \sum_{w \in \mathcal{W} \setminus (R,m)} \sum_{j \in \mathcal{J}_{\text{shift},w}} r_j p_j = \sum_{j \in \mathcal{J}_{\text{shift}}} r_j p_j$$

and

$$m\varepsilon' P(\tilde{x}) \geqslant \sum_{w \in \mathcal{W} \setminus (R,m)} m\varepsilon' P(w) \geqslant \sum_{w \in \mathcal{W} \setminus (R,m)} P(\mathcal{J}_{\text{shift},w}) = P(\mathcal{J}_{\text{shift}}),$$

which proves the claim.

For each window $w \in \mathcal{W}$, we move each job $j \in \mathcal{J}_{\text{shift},w}$ to the window (R, m), by setting $\bar{x}_{(\emptyset,(R,m))} := \tilde{x}_{(\emptyset,(R,m))} + \varepsilon' P_{\text{pre}}$ and $\bar{y}_{j,w} := 0$ and $\bar{y}_{j,(R,m)} := \bar{y}_{j,w}$. Now, we can set $m(w) := m - 1/\varepsilon'$ for all windows in $\mathcal{W}_{\text{pre}}$ and adjust the solution accordingly.

We look again at the resource amount of the windows. Since all windows have the same number of machines they use, we do not need to partition the set of generalized configurations by the number of wide jobs they contain because the number of jobs is no longer a restriction to the possibility to schedule configuration and window at the same time. So, we build just one stack of generalized configurations, in increasing order of resource amounts of the configurations. We now shift the windows exactly $\varepsilon' P_{\text{pre}}$ downwards like in the original rounding step and round the windows like before. By this rounding, we get $\varepsilon'^{-1} + 1$ different window sizes and have to extend the window (R, m) by $\varepsilon' P_{\text{pre}}$. The solution $(\bar{x}, \bar{y})$ and the set $\mathcal{W}'$ can be constructed analogously to the previous rounding. The number of operations is the same as in Lemma 8.7. □

The modified algorithm first computes the rounded instance $I'_{\text{sup},\varepsilon'}$. Then it uses the algorithm described in the proof of Lemma 8.3 to find an approximate solution to LP_{pre} with value $(1+\varepsilon')\,\text{OPT}_{\text{pre}}(I'_{\text{sup},\varepsilon'}, \varepsilon')$. By Lemma 8.9 we know that $(1+\varepsilon')\,\text{OPT}_{\text{pre}}(I'_{\text{sup},\varepsilon'}, \varepsilon') \leqslant (1+\varepsilon')(1+2\varepsilon')\,\text{OPT}(I)$. This solution is then transformed to a solution of $LP_{\mathcal{W}}$ without losing any factor in the approximation utilizing the techniques from Lemma 8.6. In the next step, the algorithm reduces the number of widows with the techniques from Lemma 8.10: Since $|\mathcal{W}'| \leqslant \varepsilon'^{-1} + 1$ and $|\mathcal{J}_W| \leqslant \varepsilon'^{-1}\lceil\log(\varepsilon'^{-1})\rceil$, the linear program has $|\mathcal{J}_{\sup W}| + |\mathcal{J}_N| + 2|\mathcal{W}'| \in \mathcal{O}(n + \varepsilon'^{-1})$ constraints and at most $|\mathcal{C}_W||\mathcal{W}'| + |\mathcal{J}_N||\mathcal{W}'| \in \mathcal{O}(n\varepsilon^{-1})$ variables. Hence, we can compute a basic solution with at most $|\mathcal{J}_{\sup W}| + |\mathcal{J}_N| + 2|\mathcal{W}'| + 1$ non-zero components in $\mathcal{O}((n + \varepsilon'^{-1})^{1.5356} n\varepsilon^{-1}) \leqslant \mathcal{O}(n^{2.5356}\varepsilon^{-1} + n\varepsilon^{-3})$ operations. Since we have at most $|\mathcal{J}_{\sup W}| + |\mathcal{J}_N| + 2|\mathcal{W}'| + 1$ non-zero components, the number of configurations and fractionally scheduled narrow jobs is bounded by $\varepsilon'^{-1}\lceil\log(\varepsilon'^{-1})\rceil + 2\varepsilon^{-1} + 3$. The makespan of this solution is bounded by $(1+\varepsilon')(1+2\varepsilon')^2\,\text{OPT}(I)$. Using Lemma 8.8, we get an integral schedule for the jobs in I_{sup} with makespan at most $(1+\varepsilon')^2(1+2\varepsilon')^2\,\text{OPT}(I) + (1 + 1/\varepsilon' + 1/\varepsilon'(\log(1/\varepsilon') + 1) + 2/\varepsilon' + 3)p_{\max}$. If we set $\varepsilon' := \varepsilon/8$, we get a schedule with makespan of at most:

$$(1+\varepsilon)\,\text{OPT} + \mathcal{O}(p_{\max}\log(1/\varepsilon)/\varepsilon)$$

Let $m \leqslant 1/\varepsilon'^2$. Again, this case is the simpler one. We redefine the sets of wide and narrow jobs. Let $\mathcal{J}_W := \{j \in \mathcal{J} | r_j > R/m\}$ and

$\mathcal{J}_N := \{j \in \mathcal{J} | r_j \leqslant R/m\}$. Let us first consider the items in $\mathcal{J}_N$. Since we can schedule exactly m of them at the same time, we know that $P(\mathcal{J}_N) \leqslant mP_{pre}$. We round the items in $\mathcal{J}_N$ by a linear grouping step into at most $1/\varepsilon'$ different sizes. The total processing time of each group is given by $\varepsilon' P(\mathcal{J}_N) \leqslant \varepsilon' m P_{pre}$. Hence, the group containing the widest items can be scheduled at the end of the schedule adding at most $\varepsilon' P_{pre}$ processing time to the makespan.

Let us now consider the items in $\mathcal{J}_W$. We round this set of items via geometric grouping. Since the narrowest item has a width of at least R/m, we partition $\mathcal{J}_W$ into $\lceil \log(m) \rceil$ sets. In each of these stacks, we do linear grouping building $1/\varepsilon'$ groups per stack. For each stack i, we add at most $(\varepsilon'/2^i) \cdot P(\mathcal{J}_{W,i}) + p_{max}$ extra processing time to the makespan. So in total we add at most

$$\sum_{i=0}^{\lceil \log(m) \rceil - 1} ((1/2^i) \cdot \varepsilon' P(\mathcal{J}'_{W,i}) + p_{max}) \leqslant 2\lceil \log(1/\varepsilon') \rceil p_{max} + 2\varepsilon' P_{pre}$$

to the makespan if we round the wide jobs this way.

Now, we have at most $\lceil \log(m) \rceil / \varepsilon' \leqslant 2\lceil \log(1/\varepsilon') \rceil / \varepsilon'$ sizes for wide jobs and $1/\varepsilon'$ sizes for narrow jobs. By Lemma 8.3, we can find a preemptive schedule with at most $2\lceil \log(1/\varepsilon') \rceil / \varepsilon' + 1/\varepsilon' + 1$ non-zero components and makespan of at most $(1+\varepsilon')\,\mathrm{OPT}_{pre}$ with $\mathcal{O}(n(\ln(n) + \varepsilon^{-2})(|\mathcal{J}_W|/\varepsilon' + n + m/\varepsilon^4)) \leqslant \mathcal{O}((m/\varepsilon'^7) \cdot \log(1/\varepsilon'))$ operations. Utilizing Lemma 8.8, we can find an integral schedule with makespan at most

$$(1 + 5\varepsilon' + \varepsilon'^2)P_{pre} + 2(1 + \lceil \log(1/\varepsilon') \rceil + 1/\varepsilon' + \lceil \log(1/\varepsilon') \rceil / \varepsilon') p_{max}$$

If we set $\varepsilon' := \varepsilon/6$, we get a schedule with makespan of at most:

$$(1+\varepsilon)\,\mathrm{OPT} + \mathcal{O}(\varepsilon^{-1} \log(\varepsilon^{-1}))p_{max}$$

This proves Theorem 8.1.

8.3 RESOURCE DEPENDENT PROCESSING TIMES

An instance $\tilde{I}$ for scheduling with resource dependent processing times is given by a set $\tilde{\mathcal{J}}$ of n jobs, a number of machines m and a resource bound $R \in \mathbb{Z}_{\geqslant 0}$. Furthermore, for each job $j \in \tilde{\mathcal{J}}$ there is a set $D_j \subseteq [R]$ of valid resource values and a processing time function $\pi_j : D_j \to \mathbb{Q}_{>0}$. The goal is to find for each job j a resource assignment $\rho_j \in D_j$ and a starting time $t_j \in \mathbb{N}$ such that $\sum_{j : t \in [t_j, t_j + \pi_j(\rho_j))} \rho_j \leqslant R$ and $\sum_{j : t \in [t_j, t_j + \pi_j(\rho_j))} 1 \leqslant m$ for each $t \in \mathbb{N}$ and such that the makespan $\max_{j \in \mathcal{J}}(t_j + \pi_j(\rho_j))$ is minimized.

We set $\pi_{max} := \max\{\pi_j(\rho) | j \in \tilde{\mathcal{J}}, \rho \in D_j\}$ and denote the makespan of an optimal schedule by $\mathrm{OPT}(\tilde{I})$. For the most part of this section, we assume that the processing time functions π_j are explicitly given as a list. Later, we discuss cases with a more compact encoding.

Like before, we first present an AFPTAS with additive term $\mathcal{O}(1/\varepsilon^2)$. The basic idea of the algorithm is to find a resource allotment for the jobs and to utilize the AFPTAS for the fixed-resource variant. Finding a resource allotment that allows analysis is the main difficulty here. W.l.o.g., we assume that $1/\varepsilon \in \mathbb{Z}_{>0}$ and set $\varepsilon' := \varepsilon/8$. Like in the fixed-resource variant, the AFPTAS works differently in the two cases $1/\varepsilon < m$ and $1/\varepsilon \geqslant m$ and again the second case is much simpler. We give a brief overview of the algorithm for the first case, followed by a detailed description and analysis for both cases, and lastly show that the improved additive term can be extended as well. After that, we discuss cases in which the processing time functions have a more compact encoding. We show that for a wide range of cases our techniques can be used to achieve AFTPAS results as well.

8.3.1 *Explicitly Given Processing Time Functions*

Algorithm

Given an instance $\tilde{I}$ and $\varepsilon > 1/m$ the algorithm can be summarized as follows:

(i) Compute via max-min resource sharing a preemptive schedule with length at most $(1+\varepsilon')\,\mathrm{OPT}_{\mathrm{pre}}(\tilde{I})$.
(ii) Using the preemptive schedule, define an instance I for the fixed-resource variant, for which $\mathrm{OPT}_{\mathrm{pre}}(I) \leqslant (1+\varepsilon')\,\mathrm{OPT}_{\mathrm{pre}}(\tilde{I})$ holds.
(iii) Apply Steps (ii) to (v) of the fixed-resource AFPTAS yielding a solution $(\bar{x}, \bar{y})$ for the window LP for the rounded instance I' that uses at most $1/\varepsilon'^2 + 2$ windows.
(iv) Compute a unique resource allotment for all but at most $3/\varepsilon'^2 + 4$ of the original jobs in $\tilde{\mathcal{J}}$, which is in some sense compatible with both the linear grouping for I and the windows.
(v) Use the unique resource allotment to define a new instance for the fixed-resource variant and a new solution $(\check{x}, \check{y})$ of the window LP that has the same length as $(\bar{x}, \bar{y})$.
(vi) Apply Step (vi) of the fixed-resource AFPTAS, yielding a schedule for almost all of the jobs.
(vii) Put the jobs that did not get a unique resource allotment on top of the schedule using at most $(3/\varepsilon'^2 + 4)\pi_{\max}$ extra height.

Preemptive Schedule

Unlike before, in this section, we mean by *preemptive* a schedule where the jobs can be interrupted at any time at no cost and restarted *later* possibly on another processor with a different resource-allotment. Note that in Section 8.2 it was possible for pieces of the same job to be executed in parallel which now is forbidden. We denote the length of an optimal preemptive schedule for $\tilde{I}$ by $\mathrm{OPT}_{\mathrm{pre}}(\tilde{I})$. Like

before, a configuration C is formally defined as a multiset of jobs from $\tilde{J}$, and we denote the multiplicity of job j in configuration C by $\rho_{C,j}$. In this context, the multiplicity of a job encodes the number of resource-units that are assigned to it. A configuration C is called *feasible* if $\sum_{j:\rho_{C,j}>0} 1 \leqslant m$, $\sum_{j\in\tilde{J}} \rho_{C,j} \leqslant R$, and $\rho_{C,j} \in D_j \cup \{0\}$ for each $j \in \tilde{J}$. Note that $\rho_{C,j} = 0$ corresponds to the case that the job j is not a part of the configuration. The set of feasible configurations is denoted by $\tilde{\mathcal{C}}$.

The computation of the schedule is done via the following LP:

$$\begin{aligned} \min \sum_{C\in\tilde{\mathcal{C}}} x_C & \\ \sum_{\rho\in D_j} \frac{1}{\pi_j(\rho)} \sum_{C:\rho_{C,j}=\rho} x_C \geqslant 1 & \qquad \forall j \in \tilde{J} \\ x_C \geqslant 0 & \qquad \forall C \in \tilde{\mathcal{C}} \end{aligned}$$

In the following, this LP is denoted by $LP_{pre}(\tilde{I})$. It is easy to see that a solution of $LP_{pre}(\tilde{I})$ corresponds to a preemptive schedule and vice-versa. Note that $LP_{pre}(\tilde{I})$ is closely related to the LP used to find a preemptive solution for the fixed-resource variant, and indeed we will follow the same approach to find an approximate solution as in Lemma 8.3.

LEMMA 8.12. *A solution x to $LP_{pre}(\tilde{I})$ that satisfies*

$$\sum_{C\in\tilde{\mathcal{C}}} x_C \leqslant (1+\varepsilon')\,\mathrm{OPT}_{pre}(\tilde{I})$$

can be found in time $\mathcal{O}(n^2 \ln(\ln(n)) m^3 \max_j |D_j| \varepsilon^{-1}(\varepsilon^{-2} + \ln n))$. Furthermore, a solution with the same objective value and at most $n+1$ non-zero components can be found using $\mathcal{O}(n^{2.5356}(\varepsilon^{-2} + \ln n))$ operations.

Proof. We present the adaptations that have to be made in the proof of Lemma 8.3. First, we define a (non-empty convex compact) set $B = \{x \in \mathbb{R}^{\tilde{\mathcal{C}}}_{\geqslant 0} | \sum_{C\in\tilde{\mathcal{C}}} x_C = 1\}$ along with non-negative linear functions $f_j : \mathbb{R}^{|\tilde{\mathcal{C}}|} \to \mathbb{R}_{>0}, x \mapsto \sum_{\rho\in D_j} \frac{1}{\pi_j(\rho)} \sum_{C:\rho_{C,j}=\rho} x_C$ for each $j \in \tilde{J}$. Moreover, we set $\lambda^* = \max\{\lambda \mid f_j(x) \geqslant \lambda, j \in \tilde{J}, x \in B\}$. The algorithm by Grigoriades et al. [59] can be applied to find an $x \in B$ that satisfies $f_j(x) \geqslant (1-\gamma)\lambda^*$ for each $j \in \tilde{J}$. In each iteration of the algorithm, a price vector $q \in \mathbb{R}^n$ is obtained and the block problem $\max\{\sum_{j\in\tilde{J}} q_j f_j(x) | x \in B\}$ has to be solved approximately with an accuracy γ' that depends linearly on γ. It can be easily seen that an optimum to this problem is obtained at a point $x \in B$ with $x_{C^*} = 1$ for a single configuration C^* and $x_C = 0$ for $C \neq C^*$. The problem to find such a configuration can be described by the following ILP:

$$\max \sum_{j\in\tilde{J}} \sum_{\rho\in D_j} \frac{q_j}{\pi_j(\rho)} x_{j,\rho} \tag{8.15}$$

$$\sum_{j\in\tilde{\mathcal{J}}}\sum_{\rho\in D_j}\rho x_{j,\rho} \leqslant R \tag{8.16}$$

$$\sum_{j\in\tilde{\mathcal{J}}}\sum_{\rho\in D_j} x_{j,\rho} \leqslant m \tag{8.17}$$

$$\sum_{\rho\in D_j} x_{j,\rho} \leqslant 1 \qquad \forall j \in \tilde{\mathcal{J}} \tag{8.18}$$

$$x_{j,\rho} \in \{0,1\} \qquad \forall j \in \tilde{\mathcal{J}},\ \rho \in D_j \tag{8.19}$$

The variables $x_{j,\rho}$ express whether ρ units of the resource are assigned to a job j in the configuration. The Constraint (8.17) and (8.16) guarantee that there are at most m jobs in the configuration with summed up resource allocation at most R; and due to (8.18) every job is scheduled at most once. This problem can be seen as a multiple-choice knapsack problem with an additional capacity constraint (kMCKP): A variant of the knapsack problem where the items are partitioned into equivalence classes and only one item from every class may be packed. Moreover, the capacity constraint bounds the *number* of items that can be packed. A naive application of basic techniques by Lawler [105] provides an FPTAS for this problem with running time $\mathcal{O}(tk^3\delta^{-1})$, where t is the number of items to be packed, k the maximum number of items that may be packed, and δ the accuracy. Applied to this case we get a running time of $\mathcal{O}(n\max_j|D_j|m^3\gamma'^{-1})$.

Scaling x and setting $\gamma = \varepsilon'/(1+\varepsilon')$, we get a solution with the required makespan after $\mathcal{O}(n(\varepsilon^{-2}+\ln n))$ iterations. In each iteration, there is a numerical overhead of $\mathcal{O}(n\ln\ln(n\gamma^{-1}))$ and the knapsack problem has to be solved. The running time of the FPTAS can be bounded by $\mathcal{O}(n\ln(\ln(n))m^2\max_j|D_j|\gamma'^{-1})$. Since $\varepsilon'/2 \leqslant \gamma \leqslant \varepsilon'$ and $\varepsilon' = \varepsilon/8$, we get the asserted running time. We turn this solution into a basic feasible one in time $\mathcal{O}(n^{2.5356}(\varepsilon^{-2}+\ln n))$. □

Fixed-Resource Instance

Based on the preemptive schedule, we now define an instance I of the problem with fixed resources. Let $x_{(j,\rho)} := 1/\pi_j(\rho)\sum_{C:\rho_{C,j}=\rho} x_C$ be the fraction of job $j \in \tilde{\mathcal{J}}$ that is scheduled with a resource amount of $\rho \in D_j$ according to the preemptive solution. W.l.o.g., we may assume that:

$$\forall j \in \tilde{\mathcal{J}} : \sum_{\rho\in D_j} x_{(j,\rho)} = 1 \tag{8.20}$$

Note that if the left side should be larger than one, we can just scale down the $x_{(j,\rho)}$ values. The new set of jobs is defined as $\mathcal{J} = \{(j,\rho)\,|\,j \in \tilde{\mathcal{J}}, \rho \in D_j, x_{j,\rho} > 0\}$ with processing time $p_{(j,\rho)} := x_{(j,\rho)}\pi_j(\rho)$ and resource requirement $r_{(j,\rho)} = \rho$. The machine set and the resource bound stay the same.

This instance I has at most $(n+1)m$ jobs because in the preemptive solution at most $n+1$ configurations are used each containing at most m jobs. Furthermore note that:

$$\mathrm{OPT}_{\mathrm{pre}}(I) \leqslant (1+\varepsilon')\,\mathrm{OPT}_{\mathrm{pre}}(\tilde{I}) \leqslant (1+\varepsilon')\,\mathrm{OPT}(\tilde{I}) \tag{8.21}$$

The first inequality holds because (x_C) yields a solution for the preemptive version of the fixed-resource problem and the second is obvious.

From here on we have to distinguish the two cases $1/\varepsilon < m$ and $1/\varepsilon \geqslant m$. In both cases, we will make use of results for the fixed-resource problem. However, the second case is much simpler since a unique resource allotment can be found prior to the application of the AFPTAS for the fixed-resource variant. Therefore, the whole algorithm can be applied, while in the first case some steps are conducted before finding a unique resource allotment for all the jobs and some afterwards.

FIRST CASE: $1/\varepsilon < m$. For instance I and ε', we now use Steps (ii) to (v) of the AFPTAS for the fixed-resource problem. By application of linear grouping, the set of jobs is divided into wide $\mathcal{J}_W$ and narrow jobs $\mathcal{J}_N$. The wide jobs are split up into $G = 1/\varepsilon'^2$ groups $\mathcal{J}_{W,i}$. It may happen that jobs that are part of multiple groups are split up correspondingly. We formally handle this by introducing a factor $\delta_{(j,\rho),i}$ that denotes the fraction of a wide job $(j,\rho) \in \mathcal{J}_W$ that lies in the i-th group in the linear grouping stack. In the next step, a modified instance $I_{\mathrm{sup}} = (\mathcal{J}_{\mathrm{sup},W} \cup \mathcal{J}_N, m, R)$ is formed for which Lemma 8.3 and subsequently Lemma 8.7 can be applied. Summarizing, we get:

LEMMA 8.13. *We can obtain a set of windows $\mathcal{W}'$, a set of configurations $\mathcal{C}_W$, and a basic feasible solution $(\bar{x},\bar{y})$ to $\mathrm{LP}_W(I_{\mathrm{sup}},\mathcal{C}_W,\mathcal{W}')$ with the following properties:*

$$|\mathcal{W}'| \leqslant 1/\varepsilon'^2 + 2 \tag{8.22}$$

$$P(\bar{x}) \leqslant (1+\varepsilon')^2\,\mathrm{OPT}_{\mathrm{pre}}(I_{\mathrm{sup}}) \tag{8.23}$$

The running time can be bounded by $\mathcal{O}(nm(\ln(nm)+\varepsilon^{-2})((nm)^{1.5356} + m\varepsilon^{-4}))$.

Proof. The running time is due to Lemma 8.3 and $|\mathcal{J}| \leqslant (n+1)m$. All the remaining claims follow from Lemma 8.7. □

Unique Resource Allotment

The goal in this section is to find a unique resource allocation r_j for almost all the jobs $j \in \tilde{\mathcal{J}}$ of the original instance.

FIRST CASE: $1/\varepsilon < m$. Using a linear program, we will find a new instance for the fixed-resource variant in which the wide jobs still fit into the linear grouping stack given by I, and the narrow jobs fit into the windows given by $(\bar{x}, \bar{y})$. We set $u_{(j,\rho),i} := \delta_{(j,\rho),i} x_{(j,\rho)}$ for each $i \in [G]$ and $(j,\rho) \in \mathcal{J}_{W,i}$, and $v_{(j,\rho),w} := (\pi_j(\rho))^{-1} \bar{y}_{(j,\rho),w}$ for each $(j,\rho) \in \mathcal{J}_N$ and $w \in \mathcal{W}'$. This yields a solution to the following LP:

$$\sum_{(j,\rho)\in\mathcal{J}_{W,i}} \pi_j(\rho) u_{(j,\rho),i} \leqslant P(\mathcal{J}_{W,i}) \qquad \forall i \in [G] \quad (8.24)$$

$$\sum_{(j,\rho)\in\mathcal{J}_N} \pi_j(\rho) v_{(j,\rho),w} \leqslant m(w) \sum_{C\in\mathcal{C}(w)} \bar{x}_{C,w} \qquad \forall w \in \mathcal{W}' \quad (8.25)$$

$$\sum_{(j,\rho)\in\mathcal{J}_N} r_{(j,\rho)} \pi_j(\rho) v_{(j,\rho),w} \leqslant R(w) \sum_{C\in\mathcal{C}(w)} \bar{x}_{C,w} \qquad \forall w \in \mathcal{W}' \quad (8.26)$$

$$\sum_{i\in[G]} \sum_{(j,\rho)} u_{(j,\rho),i} + \sum_{w\in\mathcal{W}'} \sum_{(j,\rho)} v_{(j,\rho),w} = 1 \qquad \forall j \in \tilde{\mathcal{J}} \quad (8.27)$$

$$u_{(j,\rho),i} \geqslant 0 \qquad \forall i \in [G], (j,\rho) \in \mathcal{J}_{W,i} \quad (8.28)$$

$$v_{(j,\rho),w} \geqslant 0 \qquad \forall w \in \mathcal{W}', \forall (j,\rho) \in \mathcal{J}_N \quad (8.29)$$

The Inequality (8.24) holds due to the definition of the linear grouping and because $\pi_j(\rho) u_{(j,\rho),i} = \delta_{(j,\rho)} p_{(j,\rho)}$ is exactly the height that job (j,ρ) contributes to the height $P(\mathcal{J}_{W,i})$ of group i. The next two Constraints (8.25) and (8.26) are satisfied because $\pi_j(\rho) v_{(j,\rho),w} = \bar{y}_{(j,\rho),w}$ and $(\bar{x}, \bar{y})$ satisfies the Constraints (8.8) as well as (8.9) of LP_W. Furthermore, it holds that:

$$\begin{aligned}
&\sum_{i\in[G]} \sum_{(j,\rho)\in\mathcal{J}_{W,i}} u_{(j,\rho),i} + \sum_{w\in\mathcal{W}'} \sum_{(j,\rho)\in\mathcal{J}_N} v_{(j,\rho),w} \\
&= \sum_{(j,\rho)\in\mathcal{J}_W} x_{j,\rho} \sum_{i\in[G]} \delta_{(j,\rho),i} + \sum_{(j,\rho)\in\mathcal{J}_N} (\pi_j(\rho))^{-1} \sum_{w\in\mathcal{W}'} \bar{y}_{(j,\rho),w} \\
&= \sum_{(j,\rho)\in\mathcal{J}_W} x_{j,\rho} + \sum_{(j,\rho)\in\mathcal{J}_N} (\pi_j(\rho))^{-1} p_{(j,\rho)} \\
&= \sum_{(j,\rho)\in\mathcal{J}_W} x_{j,\rho} + \sum_{(j,\rho)\in\mathcal{J}_N} x_{j,\rho} = \sum_{(j,\rho)\in\mathcal{J}} x_{j,\rho} \underset{(8.20)}{=} 1
\end{aligned}$$

This yields (8.27), and this constraint guarantees that every job in $\tilde{I}$ is used exactly once.

Next, we convert (u, v) into a basic feasible solution of the LP. Afterwards, (u, v) has at most $G + 2|\mathcal{W}'| + |\tilde{\mathcal{J}}|$ non-zero variables. Furthermore, due to (8.27) and a simple counting argument, there are at most $G + 2|\mathcal{W}'| \leqslant 3/\varepsilon'^2 + 4$ jobs $j \in \tilde{\mathcal{J}}$ with more than one non-zero variable from the set $\{u_{(j,\rho),i}, v_{(j,\rho),w} \mid \rho \in D_j, i \in [G], w \in \mathcal{W}'\}$ of variables related to j. We will schedule the jobs with more than one non-zero variable separately in the end. The rest of the jobs $\bar{\mathcal{J}}$ yield a new instance $\bar{I}$ for the fixed-resource variant. For each $j \in \bar{\mathcal{J}}$, exactly one of the following holds: There are $\rho \in D_j$ and $i \in [G]$ with $u_{(j,\rho),i} = 1$, or there are $\rho \in D_j$ and $w \in \mathcal{W}'$ with $v_{(j,\rho),w} = 1$. In the

first case, j is a wide job, i.e., $j \in \bar{\mathcal{J}}_W$, and in the second it is narrow, i.e., $j \in \bar{\mathcal{J}}_N$, while in both cases, the processing time and resource requirement are given by $p_j := \pi_j(\rho)$ and $r_j := \rho$. Moreover, the wide jobs are uniquely assigned to groups $\bar{\mathcal{J}}_{W,i}$.

We define a modified rounded instance $\bar{I}_{sup} := (\mathcal{J}_{sup,W} \cup \bar{\mathcal{J}}_N, m, R)$ for $\bar{I}$ using the old rounded wide jobs and the new narrow jobs. Furthermore, let $\check{x}_{C,w} := \bar{x}_{C,w}$ for each $C \in \mathcal{C}_W$, and for each $j \in \bar{\mathcal{J}}_N$ we set:

$$\check{y}_{j,w} := \begin{cases} p_j & \text{, if } \nu_{(j,\rho),w} = 1 \\ 0 & \text{, otherwise} \end{cases}$$

LEMMA 8.14. *It holds that* $(\check{x}, \check{y})$ *is a solution to* $\mathrm{LP}_W(\bar{I}_{sup}, \mathcal{C}_W, \mathcal{W}')$ *that has the the same makespan as* $(\bar{x}, \bar{y})$ *and to which Lemma 8.8 can be applied. It can be computed using* $\mathcal{O}(nm\varepsilon^{-2}(nm + \varepsilon^{-2})^{1.5356})$ *operations.*

Proof. The generalized configurations keep the same height. Hence, the makespan stays the same, and, since also the same rounded wide jobs are used, the Constraint (8.6) of LP_W holds. The Constraints (8.8) and (8.9) are satisfied because ν satisfies (8.25) and (8.26), while Constraint (8.7) holds by the definition of $\check{y}$.

Because of (8.24), it holds that $P(\bar{\mathcal{J}}_{W,i}) \leqslant P(\mathcal{J}_{W,i})$ for each $i \in [G]$, i.e., the new wide jobs still fit into the linear grouping stack and can therefore be scheduled by the fixed-resource AFPTAS. Also the narrow jobs can be handled because (8.8) and (8.9) hold.

The running time is dominated by the computation of the basic feasible solution. Since the LP has at most $\mathcal{O}((nm + \varepsilon^{-2}))$ constraints and $\mathcal{O}(nm\varepsilon^{-2})$ variables, this can be done in time $\mathcal{O}(nm\varepsilon^{-2}(nm + \varepsilon^{-2})^{1.5356})$. □

SECOND CASE: $1/\varepsilon \geqslant m$. In this case, it is possible to treat all the jobs as wide jobs. We apply linear grouping and define u as above. This yields a solution for the LP that is given by (8.24), (8.28), and:

$$\sum_{i \in [G]} \sum_{(j,\rho) \in \mathcal{J}_{W,i}} u_{(j,\rho),i} = 1 \qquad \forall j \in \tilde{\mathcal{J}} \tag{8.30}$$

We transform u into a basic feasible solution, and, due to the same counting argument, there are at most $G = 1/\varepsilon'^2$ fractional variables. Next, the fractional jobs are removed and the wide jobs defined as above. Due to (8.24), they fit into the linear grouping stack.

The Integral Schedule

We apply the last steps of the fixed-resource AFPTAS.

FIRST CASE: $1/\varepsilon < m$. We can apply (the proof of) Lemma 8.8 to $(\check{x}, \check{y})$ yielding an integral schedule for almost all the jobs with makespan at most:

$$((1+\varepsilon')^4)\,\mathrm{OPT}_{\mathrm{pre}}(I) + (5 + \frac{1}{\varepsilon'} + \frac{3}{\varepsilon'^2})p_{\max}$$

Note that $p_{\max} \leqslant \pi_{\max}$ holds. We schedule the remaining jobs $\tilde{\mathcal{J}} \setminus \bar{\mathcal{J}}$ successively at the end of the schedule each with an optimal resource assignment. In doing so, we add at most

$$|\tilde{\mathcal{J}} \setminus \bar{\mathcal{J}}| \max_{j \in \tilde{\mathcal{J}}} \min_{\rho \in D_j} \pi_j(\rho) \leqslant (4 + \frac{3}{\varepsilon'^2})\pi_{\max}$$

length to the schedule. Summing up and applying (8.21), we get a makespan of at most:

$$(1+\varepsilon')^5\,\mathrm{OPT}_{\mathrm{pre}}(\tilde{I}) + (9 + \frac{1}{\varepsilon'} + \frac{6}{\varepsilon'^2})\pi_{\max} \leqslant (1+\varepsilon)\,\mathrm{OPT}(\tilde{I}) + \mathcal{O}(1/\varepsilon^2)\pi_{\max}$$

The overall running time is polynomial in n, $\max_j |D_j|$, and ε^{-1}. Due to the previous observations, it can be bounded by:

$$\mathcal{O}\Big(n\frac{m^2}{\varepsilon}\max_j |D_j|(\ln(nm) + \frac{1}{\varepsilon^2})((nm)^{1.5356} + \frac{m}{\varepsilon^4})\Big)$$

SECOND CASE: $1/\varepsilon \geqslant m$. In this case, we can apply all the steps of the fixed-resource AFPTAS that follow the linear grouping yielding a schedule for all but at most $G = 1/\varepsilon'^2$ jobs. The remaining jobs again are scheduled in the end. The corresponding schedule has a makespan of at most:

$$\begin{aligned}
&(1+2\varepsilon')\,\mathrm{OPT}_{\mathrm{pre}}(I) + 2/\varepsilon'^2\pi_{\max}\\
\leqslant&(1+2\varepsilon')(1+\varepsilon')\,\mathrm{OPT}(\tilde{I}) + \mathcal{O}(1/\varepsilon^2)\pi_{\max}\\
\leqslant&(1+\varepsilon)\,\mathrm{OPT}(\tilde{I}) + \mathcal{O}(1/\varepsilon^2)\pi_{\max}
\end{aligned}$$

The overall running time is obviously smaller than in the first case.

Better Additive Term

We can extend the improved additive term to the case of resource dependent processing times as well. To achieve this, the corresponding algorithm for the fixed-resource variant has to be used as a subprocedure, and, furthermore, the computation of the unique resource allotment has to be adjusted. The reduced number of groups and windows reduces the number of constraints needed in the LP which is used to find the unique resource allotment, and therefore a basic feasible solution has less non-zero variables. Hence, the number of jobs that do not get a unique resource allotment is reduced as well. The additive term of the algorithm is due to the additive term of

the algorithm for the fixed-resource variant and the jobs without a unique resource allotment. Since both are reduced, the overall additive term is reduced as well. We give some more details for the two cases $m > 1/\varepsilon'^2$ and $m \leqslant 1/\varepsilon'^2$.

The computation of the preemptive solution and the construction of the fixed-resource instance stays the same in both cases. Note that in the modified AFPTAS for the fixed-resource problem a distinction between wide and narrow jobs is made for both cases, and therefore we consider them simultaneously. Like before, we perform the first steps of the (modified) AFPTAS for the fixed-resource problem. With the new rounding procedure the wide jobs are partitioned into G groups with $G \leqslant 1/\varepsilon' \log(1/\varepsilon') + 1/\varepsilon'$ in the first and $G \leqslant 2/\varepsilon' \log(1/\varepsilon') + 1/\varepsilon'$ in the second case, and again there may be jobs that belong to multiple groups and are split accordingly. Next, a modified instance with replaced wide jobs is formed. In the first, case Lemma 8.3 and 8.10 can be applied yielding a solution $(\bar{x}, \bar{y})$ to LP_W with at most $|\mathcal{W}'| \leqslant 1/\varepsilon' + 1$ windows. In the second case, Lemma 8.3 directly yields a solution with at most $2/\varepsilon' \log(1/\varepsilon') + 2/\varepsilon' + 1$ non-zero variables. This can be transformed directly into a solution $(\bar{x}, \bar{y})$ to LP_W with at most $|\mathcal{W}'| \leqslant 2/\varepsilon' \log(1/\varepsilon') + 2/\varepsilon' + 1$ windows. We can build the LP for the unique resource allotment the same way as before. However, now we have at most $G + 2|\mathcal{W}'| \in \mathcal{O}(1/\varepsilon' \log(1/\varepsilon'))$ jobs without a unique resource allotment. Like before, we define a modified set of narrow jobs and a modified solution for LP_W to which we apply Lemma 8.8. In the first case, Lemma 8.9 together with (8.21) yields a solution for almost all the jobs with makespan at most:

$$(1+\varepsilon')^3(1+2\varepsilon')^2 \operatorname{OPT}(\tilde{I}) + (4 + 4\varepsilon^{-1} + \varepsilon'^{-1}\log(\varepsilon'^{-1}))\pi_{\max}$$

In the second case, we get a solution with makespan at most:

$$(1+\varepsilon')^2(1+5\varepsilon'+\varepsilon'^2) \operatorname{OPT}(\tilde{I}) + 2(2 + 2\varepsilon^{-1} + \varepsilon'^{-1}\log(\varepsilon'^{-1}))\pi_{\max}$$

We add the jobs without a unique resource allotment at the end of the schedule with an extra additive error of $\mathcal{O}(\varepsilon^{-1}\log(\varepsilon^{-1})\pi_{\max})$ in both cases. For $\varepsilon' = \varepsilon/10$, this yields a makespan of at most $(1+\varepsilon)\operatorname{OPT} + \mathcal{O}(\varepsilon^{-1}\log(\varepsilon^{-1}))\pi_{\max}$. Since the bound for the running time of the AFPTAS for the fixed-resource problem stays the same, we achieve the same running time in this case as well.

8.3.2 *Processing Time Functions with Compact Encoding*

The running time of our algorithm is linear in $\max_j |D_j|$ which might be as big as the resource bound R. As long as the processing time functions are explicitly given as a list, we have $|D_j| \in \mathcal{O}(|I|)$ and this is not a problem. In this section, we consider the case that the processing time functions have a more compact encoding, and therefore the dependence on $\max_j |D_j|$ in the running time should be avoided.

Note that the dependence on $\max_j |D_j|$ is due to the computation of the preemptive schedule. Hence, we have for any encoding of the processing times:

COROLLARY 8.15. *There is an AFPTAS for scheduling with resource dependent processing times if there is an FPTAS for the preemptive version of the problem.*

In the following, we argue that in many sensible cases the techniques presented so far together with an additional geometric rounding step are sufficient to get an AFPTAS.

Since we do not want to restrict ourselves to any particular encoding of π_j, we assume in the following that the values $\pi_j(r)$ can be accessed in constant time via an oracle. Furthermore, we assume that the functions π_j are non-increasing. Note that for any instance of the problem, there is an equivalent instance with non-increasing processing time functions, and therefore this is a very natural restriction. Lastly, we assume for the sake of simple presentation that the sets of feasible values D_j are discrete intervals $\{\lambda_j, \dots, \rho_j\}$. However, all the considerations in the following can be generalized to the case that the sets D_j are given as a collection of disjoint discrete intervals.

We consider a geometric rounding step for the processing times. For each job $j \in \tilde{\mathcal{J}}$ and resource value $r \in D_j$, let $\hat{\pi}_j(r) = \pi_j(\rho_j)(1+\varepsilon')^x$ with $x = \lceil \log_{1+\varepsilon'}(\pi_j(r)/\pi_j(\rho_j)) \rceil$. Moreover, let $\hat{P}_j = \{\hat{\pi}_j(r) | r \in D_j\}$. We get $|\hat{P}_j| \in \mathcal{O}(1/\varepsilon \log(\pi_j(\lambda_j)/\pi_j(\rho_j)))$ if we assume $\varepsilon' \leqslant 1$. For each $p \in \hat{P}_j$, let r_p be the minimum value r with $\hat{\pi}_j(r) = p$. We set $\hat{D}_j = \{r_p | p \in \hat{P}_j\}$ and $\hat{I}$ to be the modified instance with new sets of valid resource values $\hat{D}_j$ and processing time functions $\pi_j|_{\hat{D}_j}$. Note that $|\hat{D}_j| = |\hat{P}_j|$ and $\hat{D}_j$ can be efficiently computed for example using binary search in time $\mathcal{O}(|\hat{P}_j| \log |D_j|)$.

LEMMA 8.16. $\mathrm{OPT}(\hat{I}) \leqslant (1+\varepsilon')\,\mathrm{OPT}(\tilde{I})$ *and a schedule for* $\hat{I}$ *is a schedule for* $\tilde{I}$.

Proof. Consider a schedule for $\tilde{I}$ and the corresponding solution to LP_{pre}. We can increase the height of each configuration by a factor of $1+\varepsilon$. Then, for each job j, we can reduce the resource amount ρ that is assigned to j to $r_{\hat{\pi}_j(\rho)}$. In doing so, the processing time of the job is increased at most by a factor of $(1+\varepsilon)$ and we can still fit it into its corresponding configurations. Actually the configuration structure changes a little bit, but all emerging configurations are feasible because we only reduced the used resources. Hence, we get a new schedule with a makespan increased by (at most) a factor of $1+\varepsilon$ that is also a schedule for $\hat{I}$ because for each job j a resource amount from $\hat{D}_j$ is used. □

Therefore, we can apply the AFPTAS to $\hat{I}$ and get, with a slight adjustment in the definition of ε', a solution with the same quality as before. Concerning the running time, note that $|\hat{D}_j| = |\hat{P}_j| \in$

$\mathcal{O}(1/\varepsilon \log(\pi_j(\lambda_j)/\pi_j(\rho_j)))$. Hence, if the encoding size of the values $\pi_j(r)$ is polynomially bounded in the input size, we get an AFPTAS.

Monotone jobs

We now consider the case that the processing time functions are non-increasing and additionally $\check{r}\pi_j(\check{r}) \leqslant \hat{r}\pi_j(\hat{r})$ for each job j and $\check{r}, \hat{r} \in D_j$ with $\check{r} \leqslant \hat{r}$. In the context of the variant of parallel job scheduling in which the processing time of a job depends on the number of used processors, jobs of this kind are called *monotone* (see e.g. [71]). The intuition behind the notion is that investing more resource will decrease the running time but also the efficiency for example due to an increased overhead. In this case, we can apply geometric rounding directly to the resource values: For $r \in D_j$, let $\check{r} = \lceil \lambda_j(1+\varepsilon')^x \rceil$ with $x = \lfloor \log_{1+\varepsilon'}(r/\lambda_j)$. We get $\check{r} \leqslant r \leqslant (1+\varepsilon)\check{r}$ yielding $\check{r}\pi_j(\check{r}) \leqslant r\pi_j(r) \leqslant (1+\varepsilon')\check{r}\pi_j(r)$, and therefore $\pi_j(\check{r}) \leqslant (1+\varepsilon')\pi_j(r)$. Let $\check{D}_j = \{\check{r} | r \in D_j\}$ and $\check{I}$ be the modified instance given by the sets $\check{D}_j$ and the processing time functions $\pi_j|_{\check{D}_j}$. Analogously to Lemma 8.16, we get:

LEMMA 8.17. $\mathrm{OPT}(\check{I}) \leqslant (1+\varepsilon')\,\mathrm{OPT}(\tilde{I})$ *and a schedule for* $\check{I}$ *is a schedule for* $\tilde{I}$.

Note that $|\check{D}_j| \in \mathcal{O}(1/\varepsilon \log(\rho_j/\lambda_j))$ (again assuming $\varepsilon' \leqslant 1$), i.e., in this case we get an AFPTAS in which the factor $\max_j |D_j|$ in the running time can be replaced by $1/\varepsilon \log R$. In particular, the encoding length of the values of the processing time functions has no influence on the number of needed operations in this case.

8.4 OPEN PROBLEMS

Concerning the single resource constrained scheduling it is an open question whether we can find an approximation algorithm with absolute approximation ratio close to $3/2$ or an APTAS with an additive term p_{max}. The case when the number of given resources s is larger than one and $m \in \theta(n)$ is challenging. In this case, we cannot hope for an APTAS even if $s = 2$ since the underlying vector bin packing problem does not allow it. An interesting question is whether techniques used for vector bin packing can be used to find good approximations for the general resource constrained scheduling as well.

For the resource dependent case, one might note that there are many instances for which π_{max} is very big. Therefore, we want to point out that the above algorithm can be modified such that the π_{max}-factor can at least be bounded by $\mathrm{OPT}_{pre}(I)$. This can be done by changing the computation of the preemptive solution: We guess $\mathrm{OPT}_{pre}(I)$ via binary search and only use configurations in which every job has a processing time of at most $\mathrm{OPT}_{pre}(I)$. This increases the overall running time only slightly. However, providing an AFPTAS in which π_{max} is replaced by something smaller, like, e.g., $\max_{j \in \tilde{J}} \min_{\rho \in D_j} \pi_j(\rho)$ or

the maximum of the average processing times, is an interesting open problem. Furthermore, the design of approximation algorithms with improved absolute ratios remains an open problem.

BIBLIOGRAPHY

[1] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows - theory, algorithms and applications*. Prentice Hall, 1993. ISBN: 978-0-13-617549-0.

[2] Ali Allahverdi. "The third comprehensive survey on scheduling problems with setup times/costs." In: *European Journal of Operational Research* 246.2 (2015), pp. 345–378. DOI: 10.1016/j.ejor.2015.04.004.

[3] Ali Allahverdi, Jatinder ND Gupta, and Tariq Aldowaisan. "A review of scheduling research involving setup considerations." In: *Omega* 27.2 (1999), pp. 219–239. DOI: 10.1016/S0305-0483(98)00042-5.

[4] Ali Allahverdi, C. T. Ng, T. C. Edwin Cheng, and Mikhail Y. Kovalyov. "A survey of scheduling problems with setup times or costs." In: *European Journal of Operational Research* 187.3 (2008), pp. 985–1032. DOI: 10.1016/j.ejor.2006.06.060.

[5] Noga Alon, Yossi Azar, Gerhard J Woeginger, and Tal Yadid. "Approximation schemes for scheduling on parallel machines." In: *J. Scheduling* 1.1 (1998), pp. 55–66. DOI: 10.1002/(SICI)1099-1425(199806)1:1<55::AID-JOS2>3.0.CO;2-J.

[6] 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. DOI: 10.1145/3070694.

[7] Stefan Arnborg, Derek G Corneil, and Andrzej Proskurowski. "Complexity of finding embeddings in ak-tree." In: *SIAM J. Algebraic Discrete Methods* 8.2 (1987), pp. 277–284. DOI: 10.1137/0608024.

[8] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009. ISBN: 978-0-521-42426-4. URL: http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521424264.

[9] Arash Asadpour and Amin Saberi. "An Approximation Algorithm for Max-Min Fair Allocation of Indivisible Goods." In: *SIAM J. Comput.* 39.7 (2010), pp. 2970–2989. DOI: 10.1137/080723491.

[10] Yuichi Asahiro, Eiji Miyano, and Hirotaka Ono. "Graph classes and the complexity of the graph orientation minimizing the maximum weighted outdegree." In: *Discrete Applied Mathematics* 159.7 (2011), pp. 498–508. DOI: 10.1016/j.dam.2010.11.003.

[11] Yossi Azar and Amir Epstein. "Convex programming for scheduling unrelated parallel machines." In: *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*. 2005, pp. 331–337. DOI: 10.1145/1060590.1060639.

[12] Yossi Azar, Leah Epstein, Yossi Richter, and Gerhard J. Woeginger. "All-norm approximation algorithms." In: *J. Algorithms* 52.2 (2004), pp. 120–133. DOI: 10.1016/j.jalgor.2004.02.003.

[13] MohammadHossein Bateni, Moses Charikar, and Venkatesan Guruswami. "MaxMin allocation via degree lower-bounded arborescences." In: *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*. 2009, pp. 543–552. DOI: 10.1145/1536414.1536488.

[14] Peter A. Beling and Nimrod Megiddo. "Using Fast Matrix Multiplication to Find Basic Solutions." In: *Theor. Comput. Sci.* 205.1-2 (1998), pp. 307–316. DOI: 10.1016/S0304-3975(98)00003-6.

[15] Ivona Bezáková and Varsha Dani. "Allocating indivisible goods." In: *SIGecom Exchanges* 5.3 (2005), pp. 11–18. DOI: 10.1145/1120680.1120683.

[16] Aditya Bhaskara, Ravishankar Krishnaswamy, Kunal Talwar, and Udi Wieder. "Minimum Makespan Scheduling with Low Rank Processing Times." In: *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*. 2013, pp. 937–947. DOI: 10.1137/1.9781611973105.67.

[17] Raphaël Bleuse, Safia Kedad-Sidhoum, Florence Monna, Grégory Mounié, and Denis Trystram. "Scheduling independent tasks on multi-cores with GPU accelerators." In: *Concurrency and Computation: Practice and Experience* 27.6 (2015), pp. 1625–1638. DOI: 10.1002/cpe.3359.

[18] Hans L. Bodlaender. "A Linear-Time Algorithm for Finding Tree-Decompositions of Small Treewidth." In: *SIAM J. Comput.* 25.6 (1996), pp. 1305–1317. DOI: 10.1137/S0097539793251219.

[19] Hans L. Bodlaender. "A Partial k-Arboretum of Graphs with Bounded Treewidth." In: *Theor. Comput. Sci.* 209.1-2 (1998), pp. 1–45. DOI: 10.1016/S0304-3975(97)00228-4.

[20] Vincenzo Bonifaci and Andreas Wiese. "Scheduling Unrelated Machines of Few Different Types." In: *CoRR* abs/1205.0974 (2012). arXiv: 1205.0974.

[21] Marin Bougeret, Pierre-François Dutot, Klaus Jansen, Christina Robenek, and Denis Trystram. "Approximation Algorithms for Multiple Strip Packing and Scheduling Parallel Jobs in Platforms." In: *Discrete Math., Alg. and Appl.* 3.4 (2011), pp. 553–586. DOI: 10.1142/S1793830911001413.

[22] Andreas Brandstädt and Vadim V. Lozin. "On the linear structure and clique-width of bipartite permutation graphs." In: *Ars Comb.* 67 (2003).

[23] Deeparnab Chakrabarty, Julia Chuzhoy, and Sanjeev Khanna. "On Allocating Goods to Maximize Fairness." In: *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*. 2009, pp. 107–116. DOI: 10.1109/FOCS.2009.51.

[24] Deeparnab Chakrabarty, Sanjeev Khanna, and Shi Li. "On (1, ε)-Restricted Assignment Makespan Minimization." In: *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*. 2015, pp. 1087–1101. DOI: 10.1137/1.9781611973730.73.

[25] Deeparnab Chakrabarty and Kirankumar Shiragur. "Graph Balancing with Two Edge Types." In: *CoRR* abs/1604.06918 (2016). arXiv: 1604.06918.

[26] Chandra Chekuri and Sanjeev Khanna. "On Multidimensional Packing Problems." In: *SIAM J. Comput.* 33.4 (2004), pp. 837–851. DOI: 10.1137/S0097539799356265.

[27] Bo Chen. "A Better Heuristic for Preemptive Parallel Machine Scheduling with Batch Setup Times." In: *SIAM J. Comput.* 22.6 (1993), pp. 1303–1318. DOI: 10.1137/0222078.

[28] Bo Chen, Chris N. Potts, and Gerhard J. Woeginger. "A Review of Machine Scheduling: Complexity, Algorithms and Approximability." In: *Handbook of Combinatorial Pptimization*. 1998, pp. 1493–1641. DOI: 10.1007/978-1-4613-0303-9_25.

[29] Bo Chen, Yinyu Ye, and Jiawei Zhang. "Lot-sizing scheduling with batch setup times." In: *J. Scheduling* 9.3 (2006), pp. 299–310. DOI: 10.1007/s10951-006-8265-7. URL: https://doi.org/10.1007/s10951-006-8265-7.

[30] Lin Chen, Deshi Ye, and Guochuan Zhang. "An improved lower bound for rank four scheduling." In: *Oper. Res. Lett.* 42.5 (2014), pp. 348–350. DOI: 10.1016/j.orl.2014.06.003.

[31] Lin Chen, Dániel Marx, Deshi Ye, and Guochuan Zhang. "Parameterized and Approximation Results for Scheduling with a Low Rank Processing Time Matrix." In: *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*. 2017, 22:1–22:14. DOI: 10.4230/LIPIcs.STACS.2017.22.

[32] Siu-Wing Cheng and Yuchen Mao. "Restricted Max-Min Fair Allocation." In: *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*. 2018, 37:1–37:13. DOI: 10.4230/LIPIcs.ICALP.2018.37.

[33] Stephen A. Cook. "The Complexity of Theorem-Proving Procedures." In: *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*. 1971, pp. 151–158. DOI: 10.1145/800157.805047.

[34] Derek G. Corneil and Udi Rotics. "On the Relationship Between Clique-Width and Treewidth." In: *SIAM J. Comput.* 34.4 (2005), pp. 825–847. DOI: 10.1137/S0097539701385351.

[35] José R. Correa, Victor Verdugo, and José Verschae. "Splitting versus setup trade-offs for scheduling to minimize weighted completion time." In: *Oper. Res. Lett.* 44.4 (2016), pp. 469–473. DOI: 10.1016/j.orl.2016.04.011.

[36] José R. Correa, Alberto Marchetti-Spaccamela, Jannik Matuschke, Leen Stougie, Ola Svensson, Víctor Verdugo, and José Verschae. "Strong LP formulations for scheduling splittable jobs on unrelated machines." In: *Math. Program.* 154.1-2 (2015), pp. 305–328. DOI: 10.1007/s10107-014-0831-8.

[37] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. ISBN: 978-3-319-21274-6. DOI: 10.1007/978-3-319-21275-3.

[38] Sami Davies, Thomas Rothvoss, and Yihao Zhang. "A Tale of Santa Claus, Hypergraphs and Matroids." In: *CoRR* abs/1807.07189 (2018). arXiv: 1807.07189.

[39] Max A. Deppert and Klaus Jansen. "Near-Linear Approximation Algorithms for Scheduling Problems with Batch Setup Times." In: *The 31st ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2019, Phoenix, AZ, USA, June 22-24, 2019*. 2019, pp. 155–164. DOI: 10.1145/3323165.3323200.

[40] 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. DOI: 10.1007/s00453-012-9668-9.

[41] Jack Edmonds. "Paths, trees, and flowers." In: *Canadian Journal of mathematics* 17 (1965), pp. 449–467. DOI: 10.4153/CJM-1965-045-4.

[42] Friedrich Eisenbrand, Christoph Hunkenschröder, and Kim-Manuel Klein. "Faster Algorithms for Integer Programs with Block Structure." In: *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*. 2018, 49:1–49:13. DOI: 10.4230/LIPIcs.ICALP.2018.49.

[43] Friedrich Eisenbrand and Gennady Shmonin. "Carathéodory bounds for integer cones." In: *Oper. Res. Lett.* 34.5 (2006), pp. 564–568. DOI: 10.1016/j.orl.2005.09.008.

[44] Friedrich Eisenbrand, Christoph Hunkenschröder, Kim-Manuel Klein, Martin Koutecký, Asaf Levin, and Shmuel Onn. "An Algorithmic Theory of Integer Programming." In: *CoRR* abs/1904.01361 (2019). arXiv: 1904.01361.

[45] Leah Epstein and Asaf Levin. "AFPTAS Results for Common Variants of Bin Packing: A New Method for Handling the Small Items." In: *SIAM Journal on Optimization* 20.6 (2010), pp. 3121–3145. DOI: 10.1137/090767613.

[46] Leah Epstein and Asaf Levin. "Scheduling with processing set restrictions: PTAS results for several variants." In: *International Journal of Production Economics* 133.2 (2011), pp. 586–595. DOI: 10.1016/j.ijpe.2011.04.024.

[47] Leah Epstein and Jirí Sgall. "Approximation Schemes for Scheduling on Uniformly Related and Identical Parallel Machines." In: *Algorithmica* 39.1 (2004), pp. 43–57. DOI: 10.1007/s00453-003-1077-7.

[48] Waldo Gálvez, José A. Soto, and José Verschae. "Symmetry Exploitation for Online Machine Covering with Bounded Migration." In: *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*. 2018, 32:1–32:14. DOI: 10.4230/LIPIcs.ESA.2018.32.

[49] M. R. Garey and Ronald L. Graham. "Bounds for Multiprocessor Scheduling with Resource Constraints." In: *SIAM J. Comput.* 4.2 (1975), pp. 187–200. DOI: 10.1137/0204015.

[50] M. R. Garey and David S. Johnson. "Complexity Results for Multiprocessor Scheduling under Resource Constraints." In: *SIAM J. Comput.* 4.4 (1975), pp. 397–411. DOI: 10.1137/0204035.

[51] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. ISBN: 0-7167-1044-7.

[52] Jan Clemens Gehrke, Klaus Jansen, Stefan E. J. Kraft, and Jakob Schikowski. "A PTAS for Scheduling Unrelated Machines of Few Different Types." In: *Int. J. Found. Comput. Sci.* 29.4 (2018), pp. 591–621. DOI: 10.1142/S0129054118410071.

[53] Vassilis Giakoumakis and Jean-Marie Vanherpe. "Bi-complement reducible graphs." In: *Adv. Appl. Math.* 18.4 (1997), pp. 389–402. DOI: 10.1006/aama.1996.0519.

[54] Vassilis Giakoumakis and Jean-Marie Vanherpe. "Linear Time Recognition and Optimizations for Weak-Bisplit Graphs, Bi-Cographs and Bipartite P_6-Free Graphs." In: *Int. J. Found. Comput. Sci.* 14.1 (2003), pp. 107–136. DOI: 10.1142/S0129054103001625.

[55] Paul C Gilmore and Ralph E Gomory. "A linear programming approach to the cutting-stock problem." In: *Operations research* 9.6 (1961), pp. 849–859. DOI: 10.1287/opre.9.6.849.

[56] Michel X. Goemans and Thomas Rothvoß. "Polynomiality for Bin Packing with a Constant Number of Item Types." In: *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*. 2014, pp. 830–839. DOI: 10.1137/1.9781611973402.61.

[57] Ronald L. Graham. "Bounds for Certain Multiprocessing Anomalies." In: *Bell System Technical Journal* 45.9 (1966), pp. 1563–1581. DOI: 10.1002/j.1538-7305.1966.tb01709.x.

[58] Ronald L. Graham. "Bounds on Multiprocessing Timing Anomalies." In: *SIAM Journal of Applied Mathematics* 17.2 (1969), pp. 416–429. DOI: 10.1137/0117039.

[59] Michael D. Grigoriadis, Leonid G. Khachiyan, Lorant Porkolab, and J. Villavicencio. "Approximate Max-Min Resource Sharing for Structured Concave Optimization." In: *SIAM Journal on Optimization* 11.4 (2001), pp. 1081–1091. DOI: 10.1137/S1052623499358689.

[60] Alexander Grigoriev, Maxim Sviridenko, and Marc Uetz. "Machine scheduling with resource dependent processing times." In: *Math. Program.* 110.1 (2007), pp. 209–228. DOI: 10.1007/s10107-006-0059-3.

[61] Alexander Grigoriev and Marc Uetz. "Scheduling jobs with time-resource tradeoff via nonlinear programming." In: *Discrete Optimization* 6.4 (2009), pp. 414–419. DOI: 10.1016/j.disopt.2009.05.002.

[62] Pinar Heggernes and Dieter Kratsch. "Linear-time certifying recognition algorithms and forbidden induced subgraphs." In: *Nord. J. Comput.* 14.1-2 (2007), pp. 87–108.

[63] Raymond Hemmecke, Shmuel Onn, and Lyubov Romanchuk. "n-Fold integer programming in cubic time." In: *Math. Program.* 137.1-2 (2013), pp. 325–341. DOI: 10.1007/s10107-011-0490-y.

[64] Petr Hlinený and Sang-il Oum. "Finding Branch-Decompositions and Rank-Decompositions." In: *SIAM J. Comput.* 38.3 (2008), pp. 1012–1032. DOI: 10.1137/070685920.

[65] 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. DOI: 10.1145/7531.7535.

[66] Dorit S. Hochbaum and David B. Shmoys. "A Polynomial Approximation Scheme for Scheduling on Uniform Processors: Using the Dual Approximation Approach." In: *SIAM J. Comput.* 17.3 (1988), pp. 539–551. DOI: 10.1137/0217033.

[67] Ellis Horowitz and Sartaj Sahni. "Exact and Approximate Algorithms for Scheduling Nonidentical Processors." In: *J. ACM* 23.2 (1976), pp. 317–327. DOI: 10.1145/321941.321951.

[68] Chien-Chung Huang and Sebastian Ott. "A Combinatorial Approximation Algorithm for Graph Balancing with Light Hyper Edges." In: *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*. 2016, 49:1–49:15. DOI: 10.4230/LIPIcs.ESA.2016.49.

[69] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. "Which Problems Have Strongly Exponential Complexity?" In: *J. Comput. Syst. Sci.* 63.4 (2001), pp. 512–530. DOI: 10.1006/jcss.2001.1774.

[70] Csanád Imreh. "Scheduling Problems on Two Sets of Identical Machines." In: *Computing* 70.4 (2003), pp. 277–294. DOI: 10.1007/s00607-003-0011-9.

[71] Klaus Jansen. "Scheduling Malleable Parallel Tasks: An Asymptotic Fully Polynomial Time Approximation Scheme." In: *Algorithmica* 39.1 (2004), pp. 59–81. DOI: 10.1007/s00453-003-1078-6.

[72] Klaus Jansen. "An EPTAS for Scheduling Jobs on Uniform Processors: Using an MILP Relaxation with a Constant Number of Integral Variables." In: *SIAM J. Discrete Math.* 24.2 (2010), pp. 457–485. DOI: 10.1137/090749451.

[73] Klaus Jansen, Kim-Manuel Klein, and José Verschae. "Closing the Gap for Makespan Scheduling via Sparsification Techniques." In: *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*. 2016, 72:1–72:13. DOI: 10.4230/LIPIcs.ICALP.2016.72.

[74] Klaus Jansen and Felix Land. "Non-preemptive Scheduling with Setup Times: A PTAS." In: *Euro-Par 2016: Parallel Processing - 22nd International Conference on Parallel and Distributed Computing, Grenoble, France, August 24-26, 2016, Proceedings*. 2016, pp. 159–170. DOI: 10.1007/978-3-319-43659-3_12.

[75] Klaus Jansen, Alexandra Lassota, and Marten Maack. "Approximation Algorithms for Scheduling with Class Constraints." In: *CoRR* abs/1909.11970 (2019). arXiv: 1909.11970.

[76] Klaus Jansen, Alexandra Lassota, and Lars Rohwedder. "Near-Linear Time Algorithm for n-fold ILPs via Color Coding." In: *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*. 2019, 75:1–75:13. DOI: 10.4230/LIPIcs.ICALP.2019.75.

[77] Klaus Jansen and Marten Maack. "An EPTAS for Scheduling on Unrelated Machines of Few Different Types." In: *Algorithms and Data Structures - 15th International Symposium, WADS 2017, St. John's, NL, Canada, July 31 - August 2, 2017, Proceedings*. 2017, pp. 497–508. DOI: 10.1007/978-3-319-62127-2_42.

[78] Klaus Jansen and Marten Maack. "An EPTAS for Scheduling on Unrelated Machines of Few Different Types." In: *Algorithmica* 81.10 (2019), pp. 4134–4164. DOI: 10.1007/s00453-019-00581-w.

[79] Klaus Jansen, Marten Maack, and Alexander Mäcker. "Scheduling on (Un-)Related Machines with Setup Times." In: *2019 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2019, Rio de Janeiro, Brazil, May 20-24, 2019*. 2019, pp. 145–154. DOI: 10.1109/IPDPS.2019.00025.

[80] Klaus Jansen, Marten Maack, and Malin Rau. "Approximation schemes for machine scheduling with resource (in-)dependent processing times." In: *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*. 2016, pp. 1526–1542. DOI: 10.1137/1.9781611974331.ch104.

[81] Klaus Jansen, Marten Maack, and Malin Rau. "Approximation Schemes for Machine Scheduling with Resource (In-)dependent Processing Times." In: *ACM Trans. Algorithms* 15.3 (2019), 31:1–31:28. DOI: 10.1145/3302250.

[82] Klaus Jansen, Marten Maack, and Roberto Solis-Oba. "Structural Parameters for Scheduling with Assignment Restrictions." In: *Algorithms and Complexity - 10th International Conference, CIAC 2017, Athens, Greece, May 24-26, 2017, Proceedings*. 2017, pp. 357–368. DOI: 10.1007/978-3-319-57586-5_30.

[83] Klaus Jansen and Lars Rohwedder. "A Quasi-Polynomial Approximation for the Restricted Assignment Problem." In: *Integer Programming and Combinatorial Optimization - 19th International Conference, IPCO 2017, Waterloo, ON, Canada, June 26-28, 2017, Proceedings*. 2017, pp. 305–316. DOI: 10.1007/978-3-319-59250-3_25.

[84] Klaus Jansen and Lars Rohwedder. "On the Configuration-LP of the Restricted Assignment Problem." In: *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*. 2017, pp. 2670–2678. DOI: 10.1137/1.9781611974782.176.

[85] Klaus Jansen and Lars Rohwedder. "Local Search Breaks 1.75 for Graph Balancing." In: *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece.* 2019, 74:1–74:14. DOI: 10.4230/LIPIcs.ICALP.2019.74.

[86] Klaus Jansen, Kim-Manuel Klein, Marten Maack, and Malin Rau. "Empowering the Configuration-IP - New PTAS Results for Scheduling with Setups Times." In: *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*. 2019, 44:1–44:19. DOI: 10.4230/LIPIcs.ITCS.2019.44.

[87] David S. Johnson. "Approximation Algorithms for Combinatorial Problems." In: *J. Comput. Syst. Sci.* 9.3 (1974), pp. 256–278. DOI: 10.1016/S0022-0000(74)80044-9.

[88] Hendrik W. Lenstra Jr. "Integer Programming with a Fixed Number of Variables." In: *Math. Oper. Res.* 8.4 (1983), pp. 538–548. DOI: 10.1287/moor.8.4.538.

[89] Ravi Kannan. "Minkowski's Convex Body Theorem and Integer Programming." In: *Math. Oper. Res.* 12.3 (1987), pp. 415–440. DOI: 10.1287/moor.12.3.415.

[90] Narendra Karmarkar and Richard M. Karp. "An Efficient Approximation Scheme for the One-Dimensional Bin-Packing Problem." In: *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*. 1982, pp. 312–320. DOI: 10.1109/SFCS.1982.61.

[91] Richard M. Karp. "Reducibility Among Combinatorial Problems." In: *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*. 1972, pp. 85–103. DOI: 10.1007/978-1-4684-2001-2_9.

[92] ShanXue Ke, BenSheng Zeng, WenBao Han, and Victor Y Pan. "Fast rectangular matrix multiplication and some applications." In: *Science in China Series A: Mathematics* 51.3 (2008), pp. 389–406. DOI: 10.1007/s11425-007-0169-2.

[93] Hans Kellerer. "An approximation algorithm for identical parallel machine scheduling with resource dependent processing times." In: *Oper. Res. Lett.* 36.2 (2008), pp. 157–159. DOI: 10.1016/j.orl.2007.08.001.

[94] Claire Kenyon and Eric Rémila. "A Near-Optimal Solution to a Two-Dimensional Cutting Stock Problem." In: *Math. Oper. Res.* 25.4 (2000), pp. 645–656. DOI: 10.1287/moor.25.4.645.12118.

[95] Kamyar Khodamoradi. "Algorithms for Scheduling and Routing Problems." PhD thesis. Simon Fraser University, 2016.

[96] Kamyar Khodamoradi, Ramesh Krishnamurti, Arash Rafiey, and Georgios Stamoulis. "PTAS for Ordered Instances of Resource Allocation Problems." In: *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2013, December 12-14, 2013, Guwahati, India*. 2013, pp. 461–473. DOI: 10.4230/LIPIcs.FSTTCS.2013.461.

[97] Kamyar Khodamoradi, Ramesh Krishnamurti, Arash Rafiey, and Georgios Stamoulis. "PTAS for Ordered Instances of Resource Allocation Problems with Restrictions on Inclusions." In: *CoRR* abs/1610.00082 (2016). arXiv: 1610.00082.

[98] Peter Kling, Alexander Mäcker, Sören Riechers, and Alexander Skopalik. "Sharing is Caring: Multiprocessor Scheduling with a Sharable Resource." In: *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2017, Washington DC, USA, July 24-26, 2017*. 2017, pp. 123–132. DOI: 10.1145/3087556.3087578.

[99] Ton Kloks. *Treewidth, Computations and Approximations*. Vol. 842. Lecture Notes in Computer Science. Springer, 1994. ISBN: 3-540-58356-4. DOI: 10.1007/BFb0045375.

[100] Dusan Knop and Martin Koutecký. "Scheduling meets n-fold integer programming." In: *J. Scheduling* 21.5 (2018), pp. 493–503. DOI: 10.1007/s10951-017-0550-0.

[101] Dusan Knop, Martin Koutecký, and Matthias Mnich. "Combinatorial n-fold Integer Programming and Applications." In: *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*. 2017, 54:1–54:14. DOI: 10.4230/LIPIcs.ESA.2017.54.

[102] Phokion G. Kolaitis and Moshe Y. Vardi. "Conjunctive-Query Containment and Constraint Satisfaction." In: *J. Comput. Syst. Sci.* 61.2 (2000), pp. 302–332. DOI: 10.1006/jcss.2000.1713.

[103] Ishai Kones and Asaf Levin. "A Unified Framework for Designing EPTAS for Load Balancing on Parallel Machines." In: *Algorithmica* 81.7 (2019), pp. 3025–3046. DOI: 10.1007/s00453-019-00566-9.

[104] Martin Koutecký, Asaf Levin, and Shmuel Onn. "A Parameterized Strongly Polynomial Algorithm for Block Structured Integer Programs." In: *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague,*

Czech Republic. 2018, 85:1–85:14. DOI: 10.4230/LIPIcs.ICALP.2018.85.

[105] Eugene L. Lawler. "Fast Approximation Algorithms for Knapsack Problems." In: *Math. Oper. Res.* 4.4 (1979), pp. 339–356. DOI: 10.1287/moor.4.4.339.

[106] Kangbok Lee, Joseph Y.-T. Leung, and Michael L. Pinedo. "A note on graph balancing problems with restrictions." In: *Inf. Process. Lett.* 110.1 (2009), pp. 24–29. DOI: 10.1016/j.ipl.2009.09.015.

[107] Kangbok Lee, Joseph Y.-T. Leung, and Michael L. Pinedo. "Makespan minimization in online scheduling with machine eligibility." In: *Annals OR* 204.1 (2013), pp. 189–222. DOI: 10.1007/s10479-012-1271-6.

[108] Jan Karel Lenstra, David B. Shmoys, and Éva Tardos. "Approximation Algorithms for Scheduling Unrelated Parallel Machines." In: *Math. Program.* 46 (1990), pp. 259–271. DOI: 10.1007/BF01585745.

[109] Joseph Y-T Leung and Chung-Lun Li. "Scheduling with processing set restrictions: A survey." In: *International Journal of Production Economics* 116.2 (2008), pp. 251–262. DOI: 10.1016/j.ijpe.2008.09.003.

[110] Joseph Y-T Leung and Chung-Lun Li. "Scheduling with processing set restrictions: A literature update." In: *International Journal of Production Economics* 175 (2016), pp. 1–11. DOI: 10.1016/j.ijpe.2014.09.038.

[111] Marten Maack and Klaus Jansen. "Inapproximability Results for Scheduling with Interval and Resource Restrictions." In: *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France*. Vol. 154. 2020, 5:1–5:18. DOI: 10.4230/LIPIcs.STACS.2020.5.

[112] Alexander Mäcker, Manuel Malatyali, Friedhelm Meyer auf der Heide, and Sören Riechers. "Non-preemptive Scheduling on Machines with Setup Times." In: *Algorithms and Data Structures - 14th International Symposium, WADS 2015, Victoria, BC, Canada, August 5-7, 2015. Proceedings*. 2015, pp. 542–553. DOI: 10.1007/978-3-319-21840-3_45.

[113] Dániel Marx. "Parameterized Complexity and Approximation Algorithms." In: *Comput. J.* 51.1 (2008), pp. 60–78. DOI: 10.1093/comjnl/bxm048.

[114] Monaldo Mastrolilli and Marcus Hutter. "Hybrid rounding techniques for knapsack problems." In: *Discrete Applied Mathematics* 154.4 (2006), pp. 640–649. DOI: 10.1016/j.dam.2005.08.004.

[115] Matthias Mnich and René van Bevern. "Parameterized complexity of machine scheduling: 15 open problems." In: *Computers & OR* 100 (2018), pp. 254–261. DOI: 10.1016/j.cor.2018.07.020.

[116] Matthias Mnich and Andreas Wiese. "Scheduling and fixed-parameter tractability." In: *Math. Program.* 154.1-2 (2015), pp. 533–562. DOI: 10.1007/s10107-014-0830-9.

[117] Clyde L. Monma and Chris N. Potts. "Analysis of Heuristics for Preemptive Parallel Machine Scheduling with Batch Setup Times." In: *Operations Research* 41.5 (1993), pp. 981–993. DOI: 10.1287/opre.41.5.981.

[118] Cristopher Moore and Stephan Mertens. *The Nature of Computation*. Oxford University Press, 2011. ISBN: 978-0-19-923321-2. URL: http://ukcatalogue.oup.com/product/9780199233212.do.

[119] Gabriella Muratore, Ulrich M. Schwarz, and Gerhard J. Woeginger. "Parallel machine scheduling with nested job assignment restrictions." In: *Oper. Res. Lett.* 38.1 (2010), pp. 47–50. DOI: 10.1016/j.orl.2009.09.010.

[120] Martin Niemeier and Andreas Wiese. "Scheduling with an Orthogonal Resource Constraint." In: *Algorithmica* 71.4 (2015), pp. 837–858. DOI: 10.1007/s00453-013-9829-5.

[121] Shmuel Onn. *Nonlinear discrete optimization*. Zürich: European Mathematical Society (EMS), 2010. ISBN: 978-3-03719-093-7/pbk. DOI: 10.4171/093.

[122] Jinwen Ou, Joseph Y-T Leung, and Chung-Lun Li. "Scheduling parallel machines with inclusive processing set restrictions." In: *Naval Research Logistics (NRL)* 55.4 (2008), pp. 328–338. DOI: 10.1002/nav.20286.

[123] Sang-il Oum and Paul D. Seymour. "Approximating clique-width and branch-width." In: *J. Comb. Theory, Ser. B* 96.4 (2006), pp. 514–528. DOI: 10.1016/j.jctb.2005.10.006.

[124] Daniel R. Page and Roberto Solis-Oba. "A 3/2-Approximation Algorithm for the Graph Balancing Problem with Two Weights." In: *Algorithms* 9.2 (2016), p. 38. DOI: 10.3390/a9020038.

[125] Daniel R. Page, Roberto Solis-Oba, and Marten Maack. "Makespan Minimization on Unrelated Parallel Machines with Simple Job-Intersection Structure and Bounded Job Assignments." In: *Combinatorial Optimization and Applications - 12th International Conference, COCOA 2018, Atlanta, GA, USA, December 15-17, 2018, Proceedings*. 2018, pp. 341–356. DOI: 10.1007/978-3-030-04651-4_23.

[126] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982. ISBN: 0-13-152462-3.

[127] Michael Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, 2012. ISBN: 978-3-319-26580-3. DOI: 10.1007/978-3-319-26580-3.

[128] Gurulingesh Raravi and Vincent Nélis. "A PTAS for Assigning Sporadic Tasks on Two-type Heterogeneous Multiprocessors." In: *Proceedings of the 33rd IEEE Real-Time Systems Symposium, RTSS 2012, San Juan, PR, USA, December 4-7, 2012*. 2012, pp. 117–126. DOI: 10.1109/RTSS.2012.64.

[129] Marko Samer and Stefan Szeider. "Constraint satisfaction with bounded treewidth revisited." In: *J. Comput. Syst. Sci.* 76.2 (2010), pp. 103–114. DOI: 10.1016/j.jcss.2009.04.003.

[130] Thomas J. Schaefer. "The Complexity of Satisfiability Problems." In: *Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1-3, 1978, San Diego, California, USA*. 1978, pp. 216–226. DOI: 10.1145/800133.804350.

[131] Frans Schalekamp, René Sitters, Suzanne van der Ster, Leen Stougie, Víctor Verdugo, and Anke van Zuylen. "Split scheduling with uniform setup times." In: *J. Scheduling* 18.2 (2015), pp. 119–129. DOI: 10.1007/s10951-014-0370-4.

[132] 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. DOI: 10.1002/(SICI)1099-1425(199909/10)2:5<203::AID-JOS26>3.0.CO;2-5.

[133] Petra Schuurman and Gerhard J. Woeginger. "Preemptive Scheduling with Job-Dependent Setup Times." In: *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, 17-19 January 1999, Baltimore, Maryland, USA.* 1999, pp. 759–767. URL: http://dl.acm.org/citation.cfm?id=314500.314911.

[134] Ulrich M. Schwarz. "A PTAS for Scheduling with Tree Assignment Restrictions." In: *CoRR* abs/1009.4529 (2010). arXiv: 1009.4529.

[135] Ulrich M. Schwarz. "Approximation algorithms for scheduling and two-dimensional packing problems." PhD thesis. University of Kiel, 2010. URL: http://eldiss.uni-kiel.de/macau/receive/dissertation_diss_00005147.

[136] Georgios Stamoulis. Private communication. 2019.

[137] Ola Svensson. "Santa Claus Schedules Jobs on Unrelated Machines." In: *SIAM J. Comput.* 41.5 (2012), pp. 1318–1341. DOI: 10.1137/110851201.

[138] Maxim Sviridenko. "A note on the Kenyon-Remila strip-packing algorithm." In: *Inf. Process. Lett.* 112.1-2 (2012), pp. 10–12. DOI: 10.1016/j.ipl.2011.10.003.

[139] Stefan Szeider. "On Fixed-Parameter Tractable Parameterizations of SAT." In: *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*. 2003, pp. 188–202. DOI: 10.1007/978-3-540-24605-3_15.

[140] Stefan Szeider. "Not So Easy Problems for Tree Decomposable Graphs." In: *CoRR* abs/1107.1177 (2011). arXiv: 1107.1177.

[141] Craig A. Tovey. "A simplified NP-complete satisfiability problem." In: *Discrete Applied Mathematics* 8.1 (1984), pp. 85–89. DOI: 10.1016/0166-218X(84)90081-7.

[142] Wenceslas Fernandez de la Vega and George S. Lueker. "Bin packing can be solved within 1+epsilon in linear time." In: *Combinatorica* 1.4 (1981), pp. 349–355. DOI: 10.1007/BF02579456.

[143] Chao Wang and René Sitters. "On some special cases of the restricted assignment problem." In: *Inf. Process. Lett.* 116.11 (2016), pp. 723–728. DOI: 10.1016/j.ipl.2016.06.007.

[144] Andreas Wiese, Vincenzo Bonifaci, and Sanjoy K. Baruah. "Partitioned EDF scheduling on a few types of unrelated multiprocessors." In: *Real-Time Systems* 49.2 (2013), pp. 219–238. DOI: 10.1007/s11241-012-9164-y.

[145] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011. ISBN: 978-0-521-19527-0. URL: http://www.cambridge.org/de/knowledge/isbn/item5759340/?site_locale=de_DE.

[146] Gerhard J. Woeginger. "A polynomial-time approximation scheme for maximizing the minimum machine completion time." In: *Oper. Res. Lett.* 20.4 (1997), pp. 149–154. DOI: 10.1016/S0167-6377(96)00055-7.

[147] Gerhard J. Woeginger. "When Does a Dynamic Programming Formulation Guarantee the Existence of a Fully Polynomial Time Approximation Scheme (FPTAS)?" In: *INFORMS Journal on Computing* 12.1 (2000), pp. 57–74. DOI: 10.1287/ijoc.12.1.57.11901.

ERKLÄRUNG

Hiermit gebe ich folgende Erklärungen ab:

- Diese Abhandlung ist, abgesehen von der Beratung durch den Betreuer, nach Inhalt und Form meine eigene Arbeit. Ich habe sie eigenständig und nur mit den angegebenen Hilfsmitteln verfasst.
- Die Arbeit ist unter Einhaltung der Regeln guter wissenschaftlicher Praxis der Deutschen Forschungsgemeinschaft entstanden.
- Es wurde mir noch nie ein akademischer Grad entzogen.

Teile dieser Arbeit sind bereits an anderer Stelle im Rahmen eines Prüfungsverfahrens vorgelegt worden. Dies betrifft Teile, die auf den Arbeiten [86] und [81] basieren und zwar Kapitel 6, die Abschnitte 8.1 und 8.2 sowie vereinzelte Formulierungen in den einleitenden Kapiteln 1 und 2. Diese Teile sind in ähnlicher Form auch in der Dissertation meiner Co-Autorin Malin Rau enthalten. Die besagte Dissertation hat den Titel „Useful Structures and How to Find Them“ und wurde 2019 an der Technischen Fakultät der Christian-Albrechts-Universität zu Kiel eingereicht. Kein anderer Teil dieser Arbeit ist bereits an anderer Stelle im Rahmen eines Prüfungsverfahrens vorgelegt worden. Teile wurden, wie in der Arbeit gekennzeichnet, im Rahmen wissenschaftlicher Veröffentlichungen publiziert.

Kiel, November 2019

Marten Maack