UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

On Scheduling with Setup Times

Dissertation

zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften (Dr. rer. nat.)
an der Fakultät für Elektrotechnik, Informatik und Mathematik
der Universität Paderborn

vorgelegt von

Alexander Mäcker

Paderborn, August 2019

Zusammenfassung

Das Auftreten von Setupzeiten für die Bereitstellung von Maschinen ist eine natürliche Annahme bei der Betrachtung von Schedulingproblemen. Derartige Setups tauchen z.B. als Startzeiten von Maschinen oder für die Rekonfiguration zwischen der Ausführung von Jobs unterschiedlicher Typen auf. Diese Arbeit beschäftigt sich mit zwei unterschiedlichen Modellen für Probleme mit Setupzeiten.

Im ersten Modell betrachten wir Jobs, die in verschiedene Klassen unterteilt sind. Sobald eine Maschine zwischen Jobs unterschiedlicher Klassen wechselt, ist ein Setup für die Rekonfiguration der Maschine notwendig. Wir betrachten dieses Problem für parallele Maschinen und die Minimierung des Makespan. Hierfür entwerfen und analysieren wir Approximationsalgorithmen für identische und heterogene Maschinen. Darüber hinaus verallgemeinern wir das Problem auf über die Zeit eintreffende Jobs und die Minimierung der maximalen Antwortzeit. Dabei betrachten wir Approximationen für den offline Fall auf einer Maschine und untersuchen die (smoothed) competitiveness eines einfachen online Algorithmus.

Im zweiten Modell befassen wir uns mit der Ausführung von Jobs auf gemieteten Maschinen und dem Ziel der Mietkostenminimierung. Wir betrachten zwei heterogene Maschinentypen, die für beliebige Zeitspannen gemietet werden können, bei denen allerdings Setupzeiten mit dem Starten einhergehen. Wir entwickeln und analysieren einen online Algorithmus für über die Zeit eintreffende Jobs mit Abarbeitungsfristen.

Abstract

The occurrence of setup times for the preparation of machines is a natural assumption when considering scheduling problems. They may occur, for example, as startup times for machines or for the (re-)configuration of machines between the processing of jobs of different types. In this thesis, we study two kinds of models for scheduling problems incorporating such setup times.

The first kind of model considers jobs that are partitioned into several classes. Whenever a machine switches between the processing of jobs of different classes, a setup for the reconfiguration of the machine needs to take place. We study this problem for parallel machines with the objective of minimizing the makespan. We design and analyze approximation algorithms for the case of identical and heterogeneous machines. Additionally, we generalize the problem by allowing jobs to arrive over time and by considering the minimization of the maximum flow time. We give an approximation algorithm for the offline case of a single machine and study the (smoothed) competitiveness of a simple online algorithm.

The second model deals with jobs that need to be processed on machines rented from the cloud and the minimization of the rental cost. We consider the availability of two heterogeneous types of machines that can be rented for arbitrary durations but incur a setup time for booting newly rented machines. We develop and analyze an online algorithm for jobs with deadlines arriving over time.

Acknowledgements

During the last years in which this thesis developed, various people were directly or indirectly involved and part of this period of my life. I would like to say thank you to all who have encouraged and supported me throughout this time.

I would like to express my appreciation to my supervisor Friedhelm Meyer auf der Heide for giving me the opportunity to write this thesis and especially, for the freedom he granted me when searching for and later working on my research topic.

Special thanks go to my co-authors Manuel Malatyali and Sören Riechers. I very much enjoyed our research sessions in front of the whiteboard discussing scheduling and streaming problems and I am thankful for all the counterexamples and questions that triggered thinking up new ideas. Also, I want to thank Klaus Jansen and Marten Maack from Kiel for their invitation for a cooperation and our joint work.

I also had great pleasure of working with all my current and former colleagues particularly (but not restricted to) those from the Algorithms & Complexity group. The good spirit in our office always made me enjoy coming to work and so I am thankful to my office mates of the last years: Andreas Cord-Landwehr, Sören Riechers, Johannes Schaefer, Till Knollmann and Jannik Castenow.

I am grateful to my family and to all of my friends. They all gave me encouragement and motivation to accomplish my personal goals. Finally, I want to thank Svenja for her moral support, her love and for being my companion.

Alexander Mäcker
Paderborn, August 2019

Contents

1 Introduction **1**
- 1.1 Outline of the Thesis and Results . . . 3
- 1.2 Discussion of the Problems and Approaches . . . 6

2 Preliminaries **11**
- 2.1 Approximation and Online Algorithms . . . 11
- 2.2 Basic Techniques . . . 14
- 2.3 Some Probability Theory . . . 15

3 Scheduling on Machines with Setup Times **19**
- 3.1 A General Model . . . 20
- 3.2 Related Work . . . 21

4 Scheduling on Identical Machines with Setup Times **27**
- 4.1 Model & Notation . . . 28
- 4.2 Simple Approaches . . . 28
- 4.3 A $(3/2 + \varepsilon)$-Approximation Algorithm . . . 31
- 4.4 An Online Variant . . . 41

5 Scheduling on Heterogeneous Machines with Setup Times **43**
- 5.1 Model & Notation . . . 44
- 5.2 Unrelated Machines . . . 45
- 5.3 Special Cases of Unrelated Machines . . . 51
- 5.4 Uniformly Related Machines . . . 56

6 Offline Scheduling for Maximum Flow Time on a Machine with Setup Times **61**
- 6.1 Model & Notation . . . 62
- 6.2 Basic Properties and Observations . . . 62
- 6.3 Approximation Algorithm . . . 63

7 Online Scheduling for Maximum Flow Time on a Machine with Setup Times **69**
- 7.1 Model, Notation & Notions . . . 70

7.2 A Non-Clairvoyant Online Algorithm 71
7.3 Competitive Analysis 73
7.4 Smoothed Competitive Analysis 79

8 Cost-efficient Scheduling on Machines from the Cloud 89
8.1 Model & Notation 90
8.2 Related Work 92
8.3 Our Results 93
8.4 Simple Lower and Upper Bounds 94
8.5 A Batch-Style Competitive Algorithm 100

9 Conclusion & Outlook 117
9.1 Scheduling on Machines from the Cloud 117
9.2 Scheduling with Setup Times 118
9.3 Future Directions 121

Bibliography 123

CHAPTER 1

Introduction

Scheduling is a fundamental area of combinatorial optimization and operations research. It deals with the allocation of scarce resources so as to efficiently carry out work, perform activities or effectively utilize the resources. Scheduling has its applicability in various scenarios. As examples, take resources in form of machines in a production system that have to produce goods according to customers' orders; processors in a computing system that have to perform certain computations; or workers in the service industry who need to cater to customers' requests. In all such scenarios, there is work to be done – in the following called *jobs* – and resources – in the following called *machines* – that carry out this work or, more general, that are required by the jobs. A *schedule* is a central concept that defines which jobs are processed on which machines at which time. Given an *objective function* assessing the quality of schedules, it is crucial to optimize over the different options in order to efficiently make use of the resources and/or to optimize job-related objectives. As an example illustrating the kind and abstraction level of problems this thesis is concerned with, take one of the most basic and popular scheduling problems defined as follows: We are given a set of (identical) machines and a set of jobs, each completely defined by its length describing how long it takes to finish it. The goal is to distribute the jobs as evenly as possible among the machines so as to minimize the length of the schedule, that is, the time at which the latest job is completed (commonly called *makespan*).

Starting from this simple scheduling problem, numerous variants tailored to specific applications by special assumptions on the machines, jobs or objectives a schedule aims at have appeared in the past. As scheduling research has, with first works published in the 1950s starting with [Joh54], a long history within operations research and computer science, there is a vast literature on it. For an introduction with a presentation of basic concepts, models and algorithms the interested reader is referred to, for example, [Pin16; Bru07]. One specific thread in the scheduling

research literature, which also already exists since the 1960s starting with [GG64], is concerned with scheduling with setups. This line of research explicitly models setup times (or, in case setups incur a cost but involve no machine time, setup cost) as they can naturally occur for the preparation of machines. Even though setup times are observed to exist in many scenarios and taking them into account can be crucial for good scheduling solutions, they are not considered in the majority of studied scheduling models. Various situations where setup times occur are documented in the literature and we name a few of them here: They can occur in production systems as changeover, adjusting or cleaning times between the production of different goods on a machine [AGA99; All+08; All15; Pin05]; as times for making data being required for the processing of jobs available at a server [AS08]; or as times for the startup of not yet running machines [MH12]. Although there is a vast literature on such scheduling problems with setup times, it is, as Allahverdi et al. noticed in their survey [AGA99] from 1999, "often devoted to the development of heuristics and their empirical analysis" and "such efforts ... do not fully exploit the theoretical developments available for solving scheduling problems without setups". Therefore, "research efforts to develop a theoretical basis for scheduling models and worst-case analysis ... will be useful from an academic as well as a practical viewpoint.".

As these observations do not seem to have become less true during the last twenty years, this thesis aims at contributing to the theoretical research in scheduling with setup times. To this end, it considers different problems from the perspective of worst-case analysis in terms of the development and analysis of *approximation* and *online* algorithms. For a first, high-level illustration of these two notions, recall the exemplary problem (without setups) from above. One perspective on this problem is to assume that a scheduling algorithm knows all the jobs to be scheduled in advance and, based on this complete information, needs to compute an assignment of jobs to machines. Because this problem (as well as the problems we consider in this thesis) is NP-hard, it is widely conjectured that an optimal solution cannot be computed efficiently. Due to this matter of fact, one is interested in approximation algorithms and heuristics. Both kinds of algorithms are efficient but, on the other hand, possibly do not provide optimal schedules. While approximation algorithms come with formally proven performance guarantees, this is usually not required for heuristics. (It is noteworthy that in the literature, efficient but suboptimal algorithms is the method of choice even in settings that are mainly motivated within the context of production systems where the size of input instances to solve tends to be much smaller than in settings motivated by computer systems. For example, roughly 75% of the papers on scheduling with setup times surveyed in [All15] consider heuristics compared to 25% considering exact solution methods.) Therefore, in the domain of approximation algorithms, the main concern is the design of algorithms whose suboptimality can be mathematically proven to be bounded by some small factor; that is, given any set of jobs, the schedule computed by the approximation algorithm is only by a small factor longer than an optimal solution. To achieve this, one could, for example, sort the jobs by non-increasing length and then assign one job after the other to the machine which (currently) completes first. This rule called

LPT (Longest Processing Time) was proven to provide schedules having a length of at most 4/3 times the length of an optimal one [Gra69]. A different perspective on the problem is to consider competitive online algorithms. Here it is assumed that the jobs are revealed to the scheduling algorithm one by one and a job needs to be assigned to a machine before the next one is presented. In this case, the online algorithm lacks knowledge of the future, which certainly may lead to suboptimal decisions. Again, to benchmark the performance of an online algorithm and measure its suboptimality, the online algorithm is compared to an optimal algorithm that knows all the jobs in advance. While it is no longer possible to use LPT (as we cannot sort the jobs by their length in the beginning), dispatching the jobs in the given order and simply assigning the current job to the machine which currently completes first is known to give schedules with length at most 2 times the length of an optimal schedule [Gra69]. We provide a brief but more formal introduction to both of these concepts, which are central in the course of this thesis, in Chapter 2 after starting with a broad overview of the problems considered and results achieved in this thesis.

1.1 Outline of the Thesis and Results

This thesis considers two kinds of models that reflect several scheduling problems focusing on the explicit incorporation of setup times into classical scheduling problems.

In Chapters 4 to 7, we study different but closely related problems based on a common model in which each job belongs to one of several given classes and a machine needs to be setup whenever it switches from processing jobs of one class to a job of another class. We formally define this model and discuss related work relevant to the problems of Chapters 4 to 7 in Chapter 3. We then start with the case of minimizing the makespan on identical parallel machines in Chapter 4 and then consider a generalized variant for non-identical machines in Chapter 5. For jobs arriving over time, a natural extension of the makespan objective is investigated in Chapter 6 and Chapter 7. There we are interested in the minimization of the maximum flow time (also called response time) on a single machine and we develop and analyze offline and online algorithms.

In Chapter 8, we consider a different setting and the problem of renting machines from the cloud and scheduling jobs so as to minimize the rental cost. Here, setup times naturally occur as times for starting (virtual) machines and acquiring resources from the cloud as soon as the scheduler issues a request for a new machine.

Finally, we conclude in Chapter 9 with some open questions and possible future work. In what follows, we give a more detailed summary of the considered models and an explicit presentation of the results achieved in the respective chapters of the thesis.

Scheduling on Identical Machines with Setup Times. In Chapter 4, we consider the problem in which a set of n jobs that is partitioned into K classes is to be scheduled on m identical machines. A machine requires a proper setup taking s time units before processing jobs of a given class. During such a setup, a machine cannot process jobs. The objective is to minimize the makespan of the resulting schedule. We design and analyze a simple and a much more involved algorithm, both running in time polynomial in n, m and K. They compute solutions with an approximation factor of 2 and one that can be made arbitrarily close to $3/2$, respectively. For constant m, we provide a simple fully polynomial-time approximation scheme. The results and presentation of this chapter are based on the following publication:

> **2015** (with M. Malatyali, F. Meyer auf der Heide and S. Riechers). "Non-preemptive Scheduling on Machines with Setup Times". In: *Proceedings of the 14th International Symposium on Algorithms and Data Structures (WADS)*, cf. [Mäc+15].

Scheduling on Heterogeneous Machines with Setup Times. Chapter 5 generalizes the problem from Chapter 4 by considering n jobs that need to be scheduled on m parallel *non-identical* machines so as to minimize the makespan. The set of jobs is again partitioned into several classes and a machine requires a setup whenever it switches from processing jobs of one class to jobs of a different class. Here, the duration of a setup may depend on the machine as well as the class of the job to be processed next.

For this problem, we study approximation algorithms for different variants of non-identical machines. For unrelated machines we obtain a $\Theta(\log n + \log m)$-approximation, which we show to be optimal (up to constant factors) unless NP = RP. This is in stark contrast to the problem when there are no setup times, in which case a 2-approximation is known. We also identify two special cases that admit constant factor approximations. For uniformly related machines we provide simple constant factor approximations as well.[1] The results are part of the following publication:

> **2019** (with K. Jansen and M. Maack). "Scheduling on (Un-)Related Machines with Setup Times". In: *Proceedings of the 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, cf. [JMM19].

Minimizing Maximum Flow Time on a Machine with Setup Times. The goal of Chapters 6 and 7 is to study the basic model from Chapter 3 under a more general

[1] It is worth mentioning that in the publication [JMM19] on which this chapter is based, we even present a PTAS for uniformly related machines. However, this part of [JMM19] was mainly contributed by the two other authors and so is not included here. The remaining results (particularly, regarding uniform machines and the general case of unrelated machines) including the core ideas of the solutions and analyses are mainly my contribution to [JMM19], except for the two special cases in Section 5.3 to which the authors contributed equally.

objective function than makespan, applicable to jobs arriving over time. Precisely, we consider a setting in which a set of n jobs needs to be processed on a single machine. Each job has a release time at which it arrives (and before which it cannot be processed) and belongs to one of K classes. The machine needs to perform a setup whenever it switches from processing jobs of one class k' to a job of a different class $k \neq k'$. The goal is to minimize the maximum flow time, given by the maximum amount of time a job stays in the system. In Chapter 6, we provide and analyze an approximation algorithm for this problem achieving an approximation factor of 7. In Chapter 7, we consider the problem in an online setting where each job is only known to the scheduler as soon as it arrives and where the processing time of a job only becomes known upon its completion (non-clairvoyance). We are interested in the potential of simple "greedy-like" algorithms. We analyze a modification of the first-in-first-out (FIFO) strategy and show its competitiveness to be $\Theta(\sqrt{n})$, which is optimal for the considered class of algorithms. For $K = 2$ classes it achieves a constant competitiveness. Our main insight is obtained by an analysis of the smoothed competitiveness, one relatively new notion in "beyond worst-case analysis" [Rou19], which tries to overcome the drawbacks of worst-case performance measures. If processing times p_j are independently perturbed to $\hat{p}_j = (1 + X_j)p_j$, we obtain a competitiveness of $O(\sigma^{-2} \log^2 n)$ when X_j is drawn from a uniform or a (truncated) normal distribution with standard deviation σ. The result proves that bad instances are fragile and "practically" one might expect a much better performance than given by the $\Omega(\sqrt{n})$-bound. The two chapters are based on the following unpublished manuscript and the following publication, respectively:

> **2018**. "Approximating Maximum Flow Time on a Machine with Setup Times". Unpublished Manuscript, cf. [Mäc18].
>
> **2017** (with M. Malatyali, F. Meyer auf der Heide and S. Riechers). "Non-clairvoyant Scheduling to Minimize Max Flow Time on a Machine with Setup Times". In: *Revised Selected Papers of the 15th International Workshop on Approximation and Online Algorithms (WAOA)*, cf. [Mäc+17].

Cost-efficient Scheduling on Machines from the Cloud. Chapter 8 considers a different setting compared to the previous chapters, which are all based on the general model introduced in Chapter 3. In Chapter 8, we consider a scheduling problem where machines need to be rented from the cloud in order to process jobs. There are two types of machines available which can be rented for machine-type dependent prices and for arbitrary, not predefined durations. However, a machine-type dependent setup time is required before a machine is available for processing. Jobs arrive online over time, have deadlines and machine-type dependent processing times. The objective is to rent machines and to schedule jobs so as to meet all deadlines while minimizing the rental cost.

As we observe that in general no algorithm can be competitive for this problem, we perform a parameterized analysis, a tool in "beyond worst-case analysis" [Rou19],

that gives much more fine-grained insights in the sense of parameterized guarantees. To this end, we parameterize instances by their (minimum) slack, a parameter which models the "easiness" of an instance in a natural way: An instance is called to have a slack of β if, for all jobs, the difference between the job's release time and the latest point in time at which it needs to be started to meet its deadline is at least β. Denoting the largest setup time by s, no finite competitiveness is possible for the case $\beta < s$. However, for $\beta = (1 + \varepsilon)s$ with $\varepsilon \geq 0$, positive results are possible. Our main finding is an online algorithm for the case $1/s \leq \varepsilon \leq 1$. Its competitiveness only depends on ε and the cost ratio of the machine types and is proven to be optimal up to a factor of $O(1/\varepsilon^2)$. This chapter is based on the following publication:

> **2018** (with M. Malatyali, F. Meyer auf der Heide and S. Riechers). "Cost-Efficient Scheduling on Machines from the Cloud". In: *Journal of Combinatorial Optimization* vol. 36, no. 4. (A preliminary version has appeared in: Proceedings of the 10th International Conference on Combinatorial Optimization and Applications (COCOA) [Mäc+16]), cf. [Mäc+18].

1.2 Discussion of the Problems and Approaches

As mentioned in the introduction, this thesis aims at contributing to theoretical research, particularly worst-case analyses, in the area of scheduling with setup times. In the following two sections, we want to illustrate the necessity of such research and want to give evidence that scheduling with setup times needs investigations apart from research done on the respective non-setup counterparts. We will outline in Section 1.2.1 that all problems studied in this thesis clearly show a significant impact of setup times on the problems themselves. Therefore, solutions that are specifically tailored to these problems are required. In Section 1.2.2, we will give a first glimpse at the approaches and techniques used (also compared to those usually used for the non-setup counterparts).

1.2.1 The Impact of Setup Times

All problems considered in this thesis show one of two characteristics with respect to the impact of setup times on the difficulty of the problem: Either the problem is trivial to solve optimally when there are no setup times but non-trivial (even to approximate) in case there are setup times; or the addition of setups makes the approximation of the already NP-hard problem even more involved.

On the one hand, the problems in Chapters 6 to 8 are trivial to solve in case there are no setup times. For minimizing the maximum flow time on a single machine, it is folklore that a simple FIFO rule is optimal, which can be shown by a simple exchange argument. However, when we have to take care of setups, such an approach can become extremely bad as it cannot control or bound the additional number of setups that have to be performed compared to an optimal solution. (For example, think of

an instance where jobs arriving over time alternately belong to two different classes.) The problem of scheduling jobs on rented machines as considered in Chapter 8 is also trivial to solve in case there are no setup times. Here it is sufficient to assign each job to its own machine being of the type where it incurs less cost. This of course does not work when each machine incurs a setup time (for which we have to pay) at the beginning. It becomes important to reuse machines that have already been started for other jobs to reduce the cost due to setup overhead. Also, decisions on which machines to open should not be based on the cost of individual jobs but rather be based on the cost jointly incurred by larger sets of jobs.

On the other hand, the problem of minimizing the makespan, as considered in Chapters 4 and 5, is NP-hard when looking for optimal solutions and not trivial to approximate in case there are no setups. It gets even more involved when setups need to be performed. In case of identical machines, we not only have to make sure that the load is distributed among the machines as equally as possible but we also have to ensure that we do not perform too many setups. This cannot be taken into account by existing algorithms and thus renders them inappropriate. (For example, such an algorithm might assign each job of a given class to a different machine, which is, except in a few cases, not a good idea as way too many setups are performed.) In case of unrelated machines, we observe an even stronger separation between models with and without setup times in Chapter 5. It is proven that one can approximate the optimal makespan to within a factor of $\Theta(\log n + \log m)$ and that this is the best we can expect (under certain complexity theoretical assumptions). This is in stark contrast to the case without setup times, which can be approximated to within a factor at most 2 and which cannot be approximated to within a factor less than $3/2$ (unless P = NP).

1.2.2 A Glimpse at Used Approaches

The problems considered in Chapters 4 to 7 are closely related in the sense that all of them are based on the same job model introduced in Chapter 3: Jobs are classified into disjoint classes that determine when a machine needs to be reconfigured. Nonetheless, due to their different machine models and/or objective functions, solving them requires quite different approaches based on various techniques. Chapter 4 extends techniques developed for approximation schemes for scheduling problems without setup times. For a fixed number of machines moderate extensions of an existing approach lead to a fully polynomial-time approximation scheme, that is, a $(1+\varepsilon)$-approximation algorithm running in time polynomial in the input and $1/\varepsilon$. However, if m is part of the input, much more involved work needs to be done for our approach. Classical approximation schemes for scheduling without setup times are based on enumerating all possible solutions of a simplified instance. The simplification is usually achieved by rounding processing times to a small number of different ones and then treating all jobs with the same (rounded) processing times equally. That is, in a solution one does not have to distinguish different jobs with the same rounded processing times leading to a sufficiently small search

space for an exhaustive enumeration. In our case, however, we need to take into account class information during the enumeration of solutions. Particularly, we cannot treat jobs of the same (rounded) processing times equally, as they may belong to different classes. This observation needs to be addressed to be able to handle class information in the enumeration process efficiently. Our approach to this problem is to carefully define a special subset of all possible schedules, which we show to always contain a good schedule. At the same time, optimizing over this subset reduces the requirement on storing class information to a minimum so that an exhaustive enumeration becomes possible. This is achieved by the fact that for schedules belonging to the defined subset, class information of at most one class needs to be considered at any time. This allows us to come up with a $(3/2 + \varepsilon)$-approximation running in time polynomial in n, m and K.

In case of unrelated machines, a good approximation becomes much harder. While this is the case if we do not have setup times due to a lower bound of $3/2$, the situation gets even worse if there are setup times. In this case it is very hard to decide which machines to setup for which classes (even if for each job its processing time is either negligible or prohibitive large so that assigning jobs becomes trivial after the setups have been scheduled). We show in Chapter 5 that this leads to a relation to the classical SetCover problem and by that establish an inapproximability bound being logarithmic in the number of jobs (and which holds unless NP = RP). On the positive side, we will see that the natural linear programming formulation combined with an iterative randomized rounding achieves an optimal approximation factor. Also, some special cases turn out to admit constant factor approximations based on linear programming and the adaptation of known rounding techniques.

Intuitively, the challenge in the previously discussed problems stems from the assignment (distribution) of jobs to machines. On the other hand, given such an assignment, sequencing the jobs on a machine is trivial: For each class of which at least one job is processed on the machine, we perform exactly one setup and all jobs of such a class are processed consecutively. The challenging part completely shifts when considering jobs arriving over time combined with the maximum flow time objective. In this case, it is a priori not clear how many setups are required per class and machine. It should be clear that this number is usually larger than one as, for example, in case two jobs of the same class arrive at points far apart in time, they usually should not use the same setup in a good solution. Hence, it becomes much harder to find a good ordering of jobs on a machine and to control the number of setups one does compared to an optimal solution. To capture these aspects, we restrict ourselves to the single machine problem. Several insights about the structure of good solutions, allow us to find a constant factor approximation for this problem in Chapter 6 based on a hybrid algorithm using greedy steps as well as exhaustive enumeration parts. In Chapter 7, we then consider the problem in an online setting with the additional assumption that jobs' processing times are not known to the scheduler. While we study a rather simple algorithm for that, the main technical challenge here is the application of a smoothed competitive analysis, in which the processing times of adversarial instances are slightly perturbed by

random noise. The notion of smoothed competitiveness aims at mitigating the drawbacks of worst-case analysis and although it can provide interesting bounds covering results ranging from worst case to average case, it has so far only been applied in a very few papers. For our problem, the high level idea of such an analysis is the following. We first identify a way to witness a certain competitiveness based on the difference in the number of setups an optimal and the online solution perform for the jobs at the end of the online schedule. To have a large difference (which is intended from the adversary's perspective), the adversary has to create intervals in which a certain amount of workload is released. At this point, the random noise added to the adversarially chosen processing times becomes profitable. Intuitively, if the adversary chooses a large amount of workload in such an interval, perturbing may lead to a high flow time for the optimal solution, giving a small competitiveness. If the adversary chooses a smaller amount of workload instead, it will (with high probability) regularly happen that this small amount of workload is further decreased by the perturbations. In this case, the online algorithm can regularly catch up with the adversary and a certain threshold on the maximum flow time will never be violated (with high probability). The question how *regularly* this happens, depends on the length of the maximal time period during which the perturbation does not lead to such small workload. Because this length depends on the amount of perturbation, we finally obtain a bound parameterized in the variance of the distribution underlying the perturbation and thus, reflecting a range of a less to more powerful adversary.

Finally, Chapter 8 also considers, in some, though quite different way, a parameterized analysis. Since in general no competitive algorithm can exist for the problem at hand, we restrict the adversary in terms of how tight the deadlines of released jobs are allowed to be. We obtain competitive ratios that are parameterized by this looseness. Technically, our approach is based on an integer linear program that we use from time to time to compute solutions to subinstances consisting of those jobs released during the last computation. Thereby the formulation of the integer linear program as well as how its solutions are used and combined are based on a careful analysis of structural properties that can be presumed for good solutions.

CHAPTER 2

Preliminaries

Optimization problems and thus scheduling problems can be considered from different perspectives with respect to the knowledge an algorithm solving the problem has. In Section 2.1 we give a brief introduction to the common notions of online and (offline) approximation algorithms together with the standard measures to assess their quality. Then in Section 2.2 we briefly sketch two basic techniques that proved to be useful and are widely used in combinatorial optimization. We conclude this chapter by a collection of some useful tools from probability theory in Section 2.3.

2.1 Approximation and Online Algorithms

As already briefly mentioned in the introduction, there are mainly two concepts for scheduling algorithms: offline and online algorithms. An *offline algorithm* is one that is given the whole input instance in advance and based on this it has to compute a solution to the problem. That is, it computes a solution based on complete information. In our scheduling setting, this means that an offline scheduling algorithm knows all the jobs together with all the parameters characterizing them in advance and then has to come up with a good schedule. This perspective on a scheduling problem might be reasonable in some settings. For example, think of a manufacturer who wants to produce certain goods based on a given bulk of orders. Or the operator of a computing center who has to, given a batch of jobs, perform a set of computations. In other settings, the assumption of the availability of complete information is less reasonable. For example, a scheduling algorithm working in the operating system of a PC does not know in advance which jobs will pop up over time and will be needed to be scheduled. To capture such settings, an *online algorithm* is one that gets the input over time and that has to make decisions as time proceeds and before it knows the entire input. Therefore, it has

to make decisions based on incomplete information about the future, which may lead to suboptimal decisions. In the scheduling settings that we consider from an online perspective in this thesis, this means that jobs are only revealed to the online scheduling algorithm over time as soon as they arrive according to their release times. Note that besides this online-over-time model there are also other notions of online settings in scheduling, for example, when jobs arrive online over a list, that is, are revealed one by one. For further details, the interested reader is referred to [Sga96; PST04].

To assess the quality of offline and online algorithms, standard measures have evolved. We give a brief introduction to these notions in the following.

2.1.1 Approximation Algorithms

Considering the fact that an offline scheduling algorithm knows all information about an (input) instance in advance and before it has to make any decisions, it is certainly possible for an offline algorithm to compute optimal solutions. However, as most offline scheduling problems are NP-hard, they cannot be solved optimally in an *efficient* way unless P = NP. Since it is widely believed that P $\neq$ NP, there is only little hope that computing optimal schedules for such problems is tractable. Therefore, one is usually interested in *approximation algorithms*, which, for a given objective function f mapping each feasible solution to a real value describing its cost, produce solutions which are only by a certain factor away from being optimal. This idea is formalized in the following definition[1].

Definition 2.1 (Approximation Algorithm)**.** For a given objective function f, a (deterministic) polynomial time algorithm $\mathcal{A}$ is called an α-approximation algorithm if, on any instance $\mathcal{I}$, $\mathcal{A}$ computes a solution $\sigma(\mathcal{I})$ such that

$$f(\sigma(\mathcal{I})) \leq \alpha \cdot f(\textsc{Opt}(\mathcal{I})),$$

where $\textsc{Opt}(\mathcal{I})$ denotes an optimal solution to instance $\mathcal{I}$ under objective function f.

In case $\mathcal{A}$ is a randomized algorithm, we demand $\mathbb{E}[f(\sigma(\mathcal{I}))] \leq \alpha \cdot f(\textsc{Opt}(\mathcal{I}))$ where $\mathbb{E}$ denotes the expectation with respect to the random choices of the algorithm. In both cases, α is also often referred to as the *approximation factor* or *approximation ratio*. An algorithm $\mathcal{A}$ that computes, for any fixed $\varepsilon > 0$, a $(1+\varepsilon)$-approximation in time polynomial in the input, is called a *polynomial-time approximation scheme* (PTAS). If it also runs in time polynomial in $\frac{1}{\varepsilon}$, it is called a *fully polynomial-time approximation scheme* (FPTAS) and if its running time is of the form $g(\frac{1}{\varepsilon})$ times some polynomial in the input, where g is an arbitrary function, it is called *efficient polynomial-time approximation scheme* (EPTAS). An introduction to approximation algorithms and useful techniques for their design can be found in [WS11; Vaz01].

[1] Note that the definition assumes that we are concerned with minimization problems (which all of our problems actually are). However, one can define approximation algorithms for maximization problems analogously.

2.1.2 Competitive Algorithms

Online algorithms lack the knowledge about the future and therefore, they are usually not able to compute optimal solutions. To be able to assess the quality of an online algorithm, the notion of *competitiveness* became the standard way to measure the degree of suboptimality on a per instance basis. Similar to approximation algorithms, we have the following definition for minimization problems.

Definition 2.2 (Competitive Online Algorithm)**.** For a given objective function f, a (deterministic[2]) online algorithm $\mathcal{A}$ is called to be α-competitive if, on any instance $\mathcal{I}$, $\mathcal{A}$ computes a solution $\sigma(\mathcal{I})$ such that

$$f(\sigma(\mathcal{I})) \leq \alpha \cdot f(\textsc{Opt}(\mathcal{I})),$$

where $\textsc{Opt}(\mathcal{I})$ denotes an optimal (offline) solution to instance $\mathcal{I}$ under objective function f.

The value α of an α-competitive algorithm is often also referred to as its *competitive factor* or *competitive ratio*.

One often considers online algorithms together with competitive analysis as a game played between a malicious adversary and the online algorithm: The adversary dictates the instance with the goal of maximizing the ratio of the cost of the online algorithm and the cost of an optimal offline solution. Intuitively, the adversary tries to trick the online algorithm and urge it to decisions that will turn out to be bad (and particularly, different to those of an optimal offline solution) in the end.

Note that for online algorithms we do not require a polynomial runtime. Therefore, the competitiveness (only) focuses on the loss in the quality of solutions due to lack of knowledge of the future. This is in contrast to approximation algorithms, where the approximation factor captures the intractability of a problem due to restricted time that can be spent on finding a solution. An introduction to online algorithms and competitive analysis is given in [BE05] and a brief survey of some classical results can be found in [Alb03].

Finally, it is worth mentioning that although competitive analysis is the standard tool for analyzing online algorithms, it is also often criticized to be overly pessimistic. Single pathological instances can result in bounds that diverge from practical observations or high lower bounds can make the design of good competitive algorithms impossible. One approach to overcome this problem by weakening the all powerful adversary is the concept of smoothed competitiveness. In Chapter 7 we give a brief introduction to smoothed competitiveness and apply it in the context of an online scheduling problem with setup times. Another approach, which we use in Chapter 8, is to perform a parameterized analysis to overcome the problem of the existence of instances on which no online algorithm can perform well.

[2] Competitiveness can also be defined for randomized online algorithms. However, in this thesis we only consider deterministic online algorithms.

2.2 Basic Techniques

Next, we recall two basic techniques that are helpful and often applied when designing approximation algorithms.

2.2.1 Dual Approximation Framework

When designing approximation algorithms, it is often helpful to apply the concept of *dual approximations* as introduced by Hochbaum and Shmoys in [HS87]. Instead of coming up with an algorithm that directly optimizes the desired objective function value, one assumes that a bound T on the optimal objective function value is given. A dual α-approximation algorithm then computes a schedule with objective function value at most αT or correctly decides that no schedule with objective function value T exists at all. Employing this idea, a binary search for a good value T started on an interval $I \ni f(\textsc{Opt})$ that contains the optimal objective function value finally provides an approximation algorithm with approximation factor α (assuming $f(\textsc{Opt})$ to be integral). Note that, given such an interval I of length $|I|$ and a dual approximation algorithm with runtime RT, then leads to an approximation algorithm with a runtime of $O(\log(|I|) \cdot RT)$. Also observe that this approach allows us to consider a scheduling problem from a packing perspective: Instead of computing a schedule with a small objective function value, a dual approximation algorithm has to pack the jobs so that the guess T on the optimal value is not violated by the constructed solution. Unless stated otherwise, throughout this thesis we consistently use T to denote the guess on the optimal objective function value.

2.2.2 Linear Programming

Linear programming is a basic technique that has proven to be quite useful for the design and analysis of approximation algorithms. Many optimization problems, amongst others some of the scheduling problems considered in this thesis, can be exactly formulated as an *integer linear program* (ILP). Following the presentation in [WS11], a binary ILP formulates the problem at hand based on a set of (binary) decision variables constrained by linear inequalities (*constraints*) together with an *objective function* defined by a linear function on the decision variables. Such decision variables can, for example, be variables determining whether a job is processed on a certain machine and a constraint can, for example, ensure that not too many jobs are assigned to any machine. Formally, given n rational values $c_1, \ldots, c_n$, m rational values $b_1, \ldots, b_m$ and an $m \times n$ matrix $(a_{ij}) \in \mathbb{Q}^{m \times n}$, an ILP with binary variables can be represented as in Figure 2.1.

We call the value of the objective function achieved by the solution (that is, an assignment of the variables) that minimizes the objective function the *optimal value* of the program. Unfortunately, ILPs cannot be solved in polynomial time unless P = NP and therefore, formulating a problem as an ILP does not directly help us in finding a solution when we are restricted to a polynomial runtime. However, if

$$\begin{aligned} \text{minimize} \quad & \sum_{j=1}^{n} c_j x_j & & \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq b_i, & & i = 1, \ldots, m, \\ & x_j \in \{0, 1\} & & j = 1, \ldots, n. \end{aligned}$$

Figure 2.1: General form of a binary integer linear program.

we replace the constraints $x_j \in \{0, 1\}$ by $0 \leq x_j \leq 1$ for all $j = 1, \ldots, n$ (that is, we relax our constraints and do no require variables to be integers), we obtain a *linear relaxation* of the problem and the program becomes a *linear program* (LP). On the positive side, it is known that LPs can be solved in polynomial time using an interior points method [Kar84] or the ellipsoid method [GLS81]. On the other hand, a linear relaxation usually does not capture the original problem. For example, if a (binary) variable determines whether a job is processed on a certain machine, in a solution to the linear relaxation the variable can have a fractional value, not making any definite decision on the assignment of the job. Still, it turned out that LPs are very useful for the design of approximation algorithms as even a fractional solution can carry useful information about a problem's integral solutions. First of all, the optimal value of a linear relaxation is at most the optimal value of the ILP from which the relaxation is derived, thereby giving a lower bound for the ILP's value. Second, a fractional solution to the LP relaxation of the ILP formulation of a given problem can be useful for the construction of a (provably good) solution to the original problem. Popular examples for that is the concept of randomized rounding where fractional values are used as probabilities (e.g. [RT87; Sri99; MR95]) for rounding fractional values to integral ones, iterative rounding methods [LRS11], the approach of deterministic rounding of fractional values (e.g. [LST90; Cor+15]), and approaches where the values of an optimal fractional solution guide a simple list-scheduling algorithm constructing a feasible solution to the original problem (e.g. [Hal+97]). In this thesis, we will use ILPs in Chapter 8 and LPs together with iterative randomized as well as deterministic rounding in Chapter 5.

2.3 Some Probability Theory

In the following, we recall some definitions and results from probability theory. We assume the reader to be familiar with the basic notions such as random experiments, probability distributions, random variables and moments of random variables. We only briefly sketch some further results that we use in some of the chapters of this thesis. Unless stated otherwise, the presented content can be found in standard books such as [MU05].

We start with two bounds that are quite useful and convenient when designing randomized algorithms. They can, for example, be used to analyze an algorithm's failure probability or its expected behavior and are well-known and widely used.

Theorem 2.3 (Chernoff Bounds)**.** *Let $X_1, X_2, \ldots, X_n$ be independent random variables with $0 \leq X_i \leq 1$ for all $1 \leq i \leq n$. Let $X = \sum_{i=1}^{n} X_i$ and $\mathbb{E}[X] \leq \mu$. Then it holds for any $\delta \geq 1$*

$$\Pr[X \geq (1+\delta)\mu] \leq e^{-\delta\mu/3}.$$

Lemma 2.4 (Union Bound)**.** *Let $A_1, A_2, \ldots, A_n$ be a collection of events. Then it holds*

$$\Pr[A_1 \cup A_2 \cup \ldots \cup A_n] \leq \sum_{i=1}^{n} \Pr[A_i],$$

or slightly restated

$$\Pr[\exists i \text{ such that } A_i \text{ holds}] \leq \sum_{i=1}^{n} \Pr[A_i].$$

When computing the expectation of a random variable, it is sometimes more handy to distinguish several cases, that is, to study several conditional expectations. The following result formalizes this idea.

Lemma 2.5 (Law of Total Expectation)**.** *For two random variables X and Y it holds*

$$\mathbb{E}[X] = \sum_{y} \mathbb{E}[X \mid Y = y] \cdot \Pr[Y = y],$$

where the sum is over all y in the range of Y.

Also, it is more convenient (and often still sufficient) to upper bound the expectation based on tail estimates (for example obtained by applying Chernoff bounds) instead of carrying out an exact computation using the following result.

Lemma 2.6. *Let X be a random variable such that $\Pr[X \geq \alpha] \leq p$ and $X \leq B$. Then we have*

$$\mathbb{E}[X] \leq p \cdot B + \alpha.$$

In Chapter 7, we will analyze algorithms under the presence of random noise perturbing the adversarial instances. There, we will make use of the following two distributions, which are common in probability theory.

Definition 2.7 (Uniform Distribution)**.** The (continuous) uniform distribution $\mathcal{U}(a, b)$ over an interval $[a, b]$ is described by the probability density function

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{for } a \leq x \leq b, \\ 0 & \text{otherwise.} \end{cases}$$

Its expectation is $\frac{1}{2}(a+b)$ and its variance $\frac{1}{12}(b-a)^2$.

Definition 2.8 (Normal Distribution)**.** The normal distribution $\mathcal{N}(\mu, \sigma^2)$ with expectation μ and standard deviation σ is described by the probability density function

$$\phi_{\mu,\sigma}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \text{ for } -\infty < x < \infty.$$

The (cumulative) distribution function is denoted $\Phi_{\mu,\sigma}(x)$. For the standard normal distribution, that is, for $\mu = 0$ and $\sigma = 1$, we write $\Phi(x)$ instead of $\Phi_{0,1}(x)$ and $\phi(\cdot)$ instead of $\phi_{0,1}(\cdot)$ analogously.

While the normal distribution is a very common distribution for randomness, for our purposes it has the drawback that any value in $(-\infty, \infty)$ occurs with positive probability. Instead we would like to restrict the distribution to a certain range. One way to achieve this is to consider truncated normal distributions, which essentially are obtained by setting values outside the specified range to zero and then rescaling the values inside the range. The following definition and more about truncated normal distributions can be found in [Bur14].

Definition 2.9 (Truncated Normal Distribution)**.** Let $X \sim \mathcal{N}(\mu, \sigma^2)$ have a normal distribution. X conditioned on $a < X < b$ has a truncated normal distribution denoted $\mathcal{N}_{(a,b)}(\mu, \sigma^2)$ given by the probability density function

$$\begin{cases} \frac{\phi(\frac{x-\mu}{\sigma})}{\sigma(\Phi(\frac{b-\mu}{\sigma}) - \Phi(\frac{a-\mu}{\sigma}))} & \text{for } a \leq x \leq b, \\ 0 & \text{otherwise.} \end{cases}$$

The truncated normal distribution has the following moments

$$\mathbb{E}[X \mid a < X < b] = \mu + \sigma \frac{\phi(\alpha) - \phi(\beta)}{\Phi(\beta) - \Phi(\alpha)}$$

and

$$\mathbb{V}[X \mid a < X < b] = \sigma^2 \left[1 + \frac{\alpha\phi(\alpha) - \beta\phi(\beta)}{\Phi(\beta) - \Phi(\alpha)} - \left(\frac{\phi(\alpha) - \phi(\beta)}{\Phi(\beta) - \Phi(\alpha)}\right)^2\right]$$

where $\alpha := \frac{a-\mu}{\sigma}$ and $\beta := \frac{b-\mu}{\sigma}$.

When working with sums of independent random variables (for example, variables representing random noise as mentioned above) it is often hard to determine their exact distribution. One way to still get an estimate on their distribution is to rely on the central limit theorem. It is a well-known fact that, under certain circumstances, the cumulative distribution function of a sum of independent random variables can be approximated by the cumulative distribution function of the standard normal distribution. This approximation can even be quantified in terms of the error and the following bound can be found, for example, in [Fel66].

Theorem 2.10 (Berry-Esseen Inequality)**.** *Let $X_1, X_2, \ldots, X_n$ be independent random variables. Let $\mathbb{E}[X_i] = 0$ for all $1 \leq i \leq n$ and denote $\mathbb{E}[X_i^2] = \sigma_i^2 > 0$ and*

$\mathbb{E}[|X_i|^3] = \rho_i < \infty$. *Let* $S = \frac{X_1+X_2+\ldots+X_n}{\sqrt{\sigma_1^2+\sigma_2^2+\ldots+\sigma_n^2}}$, F *be the distribution function of* S *and* $\delta = \sup_x |F(x) - \Phi(x)|$. *By the central limit theorem we have*

$$\Pr[S \leq x] \leq \Phi(x) + \delta.$$

Also, δ *can be bounded as follows*

$$\delta \leq C_0 \left(\sum_{i=1}^{n} \sigma_i^2\right)^{-(3/2)} \cdot \sum_{i=1}^{n} \rho_i \leq C_0 \left(\sum_{i=1}^{n} \sigma_i^2\right)^{-(1/2)} \cdot \max_{1\leq i\leq n} \frac{\rho_i}{\sigma_i^2}$$

and by [She10], C_0 *can be upper bounded by* 0.56.

CHAPTER 3

Scheduling on Machines with Setup Times

Setup times are a fundamental aspect to consider when planning the execution of jobs or the production of goods on (parallel) machines. They frequently occur in natural settings where a machine, after doing some work of a certain type, has to be prepared in some way for upcoming jobs of a different type. A variety of applications and motivations for such models with an explicit modeling of setup times have been reported in the literature. Consider the following examples of which we think they suit our setting best:

- Setup times occur in production systems, for example, when machines need to be reconfigured or cleaned during the manufacturing of different customer orders [Pot91] or for preparations such as the calibration or change of tools [AGA99; All+08; All15]. In this context they are often referred to as changeover times.
- They occur in settings where we have multiple machines but only a single server/worker that has to be present at a machine to process a job. Moving the server from one machine to another incurs a setup time [Sah72]. A similar setting is one where we have multiple queues of requests that are served by a single server one after the other. The server requires a setup time when switching between the serving of different queues. For such a setting, though with objectives different to ours, a whole line of research known under the term *polling systems* with applications in, for example, computer communications, traffic and productions exists [LS90].
- Similarly, further examples are settings in which a server has to answer requests of different types, depending on the data to be loaded into memory and to be accessed by the server. Here setups occur due to the transfer of data to the

memory [DS00]. Similar settings occur when different jobs require different compilers to be present in memory [BD78], or when programs require the transfer of files they are dependent on [AS08].

- A further motivation might be reconfigurable computing where, for example, the use of field-programmable gate arrays (FPGAs) requires the consideration of non-negligible reconfiguration overhead times [KB15; Res+03] and the reuse of configurations among different jobs [Res+03]. [Pla+06] mentions the applicability of scheduling with setup times in the context of networked systems where resources are multiplexed among different services, for example, in case of a shared data center or a programmable network processor (also see [Kok05]).

- Finally, setup times can be used to model less obviously related settings, such as a scenario where an intersection of two streets is equipped with traffic lights and where setup times describe the time drivers need for start-up once they see green light [HR87].

Motivated by the above, we consider a scheduling model that incorporates setups and naturally generalizes the classical (parallel) machine scheduling problem: We are given a set of jobs as well as a single or parallel machines and the goal is to find a schedule that defines for each job when it is processed on which machine so as to optimize a given objective function. Our generalization assumes the set of jobs to be partitioned into classes. Whenever the machine switches from processing jobs of one class to jobs of a different class, a setup needs to take place for the reconfiguration of the machine. During a setup a machine cannot process any jobs leading to the necessity of taking into account setups in scheduling decisions.

In the course of Chapters 4 to 7, we study different variants of this general model, which we introduce more formally in Section 3.1 and for which we review relevant related work in Section 3.2. (The problem studied in Chapter 8 has a different motivation and focus and thus, the respective model and work related to it are discussed separately in the respective chapter.) Our models in Chapters 4 to 7 vary with respect to machine environments and objective functions. We start with the classical optimization goal of minimizing the makespan on identical machines in Chapter 4 and generalize this model to heterogeneous machines in Chapter 5. Afterwards we generalize the makespan objective to maximum flow time and study the single machine case in its offline as well as online setting in Chapter 6 and Chapter 7, respectively.

3.1 A General Model

We consider a model in which a set $\mathcal{J} = \{1, 2, \ldots, n\}$ of jobs has to be scheduled on a set $\mathcal{M}$ of $m \geq 1$ machines. Each job $j \in \mathcal{J}$ is characterized by a *processing time* (or *size*) p_{ij} for each machine $i \in \mathcal{M}$, which describes the time it takes to process

job j if assigned to machine i. Additionally, each job $j \in \mathcal{J}$ belongs to exactly one class $k_j \in \mathcal{K} = \{C_1, C_2, \ldots, C_K\}$ with $\mathcal{J} = C_1 \dot\cup \ldots \dot\cup C_K$ and $k_j = C_k$ if and only if $j \in C_k$. Before a job $j \in C_k$ can be processed on a machine, this machine has to be configured properly for processing jobs of class C_k and afterwards jobs of class C_k can be processed without additional setups until the machine is reconfigured for a class $C_{k'} \neq C_k$. That is, a setup needs to take place before the first job is processed on a machine and whenever the machine switches from processing a job j' to a job j with $k_j \neq k_{j'}$. Such a setup for class k_j on machine i takes time s_{ik_j} and while setting up a machine, it is blocked and cannot process any job. Given a schedule, we call all jobs sharing the same setup, that is, a maximal set of jobs of a common class scheduled contiguously and without any intermediate setup, a *batch*.

In this setting, a schedule has to assign each job $j \in \mathcal{J}$ to exactly one machine $i \in \mathcal{M}$ and specify a time interval of length p_{ij} during which the job is processed. Additionally, we require that a machine processes at most one job at a time and that the schedule fulfills the aforementioned requirements regarding setups. The goal is to find a feasible schedule that minimizes a certain objective function. In the following, we consider the objectives of minimizing the makespan (latest completion time of a job) on parallel machines and minimizing the maximum flow time (difference between the completion and release time of a job) on a single machine. We defer the more detailed specifications and formal definitions of the models for jobs, machines and the objective function to the respective chapters.

3.2 Related Work

In this section, we review relevant work related to the problems considered in Chapters 4 to 7. Most closely related to Chapter 4 and Chapter 5, we present results for optimizing the makespan in case there are no setups and take a look at existing results that explicitly model setup times. With respect to Chapter 6 and Chapter 7 we summarize existing work on scheduling with flow time (and related) objectives and with respect to analysis techniques we take a look at the notion of smoothed competitiveness.

3.2.1 Makespan Scheduling without Setups

Our model as described above can be seen as a generalization of classical parallel machine models without setup times (or, equivalently, where all setup times are 0), which have been studied quite a lot. We structure our review of literature on such models based on the kind of machines and depending on whether they are assumed to be identical or heterogeneous.

Identical Machines. The problem of minimizing the makespan on a set of identical parallel machines has been considered for quite a while and it has been extensively studied from the perspective of approximation algorithms. As early as in 1969, in

[Gra69] Graham proposes a scheduling algorithm nowadays known as list scheduling. It assigns one job after the other in the order of a given list and always assigns the current job to the machine that has the least load so far. Graham analyzes the list scheduling algorithm and shows an approximation factor of $2 - \frac{1}{m}$. He also proves that in case the list of jobs is ordered according to non-decreasing processing times (then we call the algorithm LPT abbreviating Longest Processing Time), the resulting schedule provides $(\frac{4}{3} - \frac{1}{3m})$-approximations. Later, the approximation ratios were significantly improved and PTASs with runtimes that are linear in the number n of jobs are known due to Alon et al. [Alo+98] and Hochbaum and Shmoys [HS87]. The approach in [Alo+98] even works for various objective functions such as minimizing (maximizing) the maximum (minimum) machine completion time or the sum of machine completion times. If the number m of machines is constant, even an FPTAS exists as proven by Horowitz and Sahni in [HS76].

Non-identical Machines. One widely studied model for non-identical machines is that of uniformly related ones. In this case each machine runs at a fixed speed, which is the same for all jobs and hence, for any two jobs j, j' and two machines i, i', $\frac{p_{ij}}{p_{i'j}} = \frac{p_{ij'}}{p_{i'j'}}$. Similar to LPT for identical machines, it is also defined for uniformly related machines: Here the current job is always assigned to the machine where it finishes first if appended to the current schedule. Kovács shows in [Kov10] that in this case LPT schedules provide $(1 + \frac{1}{\sqrt{3}})$-approximations. As for identical machines, it is also known for a long time due to the work of Hochbaum and Shmoys [HS88] that a PTAS can solve the problem of uniformly related machines arbitrarily close to optimal. More recently, in [Jan10] Jansen even shows that the running time can be further improved by coming up with an EPTAS.

The case of unrelated machines where no restrictions are imposed on the processing times significantly differs from the uniform case due to an inapproximability result of $\frac{3}{2}$ (unless P = NP) as proven by Lenstra, Shmoys and Tardos in [LST90]. On the positive side, the same work as well as [ST93] show that there are algorithms that provide 2-approximations based on rounding fractional solutions to a linear programming formulation of the problem. Shchepin and Vakhania improved this and show in [SV05] that the approximation factor can be improved to $(2 - \frac{1}{m})$. A purely combinatorial approach with approximation factor 2 is also known due to Gairing et al. [GMW07]. A special case of unrelated machines is the restricted assignment problem where each job j has a set M_j of eligible machines on which it can be processed and has the same size on all of them. While the aforementioned lower bound of $\frac{3}{2}$ from [LST90] applies to the restricted assignment problem as well, for special cases of the restricted assignment problem stronger results are known. For example, Ebenlendr et al. [EKS14] show that the same lower bound even holds for the more restrictive case where $|M_j| \leq 2$ for all j, and design a $\frac{7}{4}$-approximation algorithm for this case. For the general restricted assignment case, Svensson [Sve12] provides an algorithm for estimating the optimal makespan within a factor of $\frac{33}{17}$. Jansen and Rohwedder [JR17b] improve this to $\frac{11}{6}$ and also give a constructive

algorithm with quasipolynomial running time and approximation ratio $\frac{11}{6} + \varepsilon$ in [JR17a]. Another special case is that of unrelated machines of only a few (constant number of) types. In this case all machines of the same type are identical but different types can behave in an unrelated way. For that problem Jansen and Maack provide an EPTAS in [JM17].

3.2.2 Makespan Scheduling with Setups

Scheduling with an explicit modeling of setup times has a long history, particularly within the community of operations research. The vast majority of work there studies hardness results, exact algorithms and heuristics, which are evaluated through simulations, but without formal performance guarantees. The interested reader is referred to the exhaustive surveys on these topics by Allahverdi et al. [AGA99; All+08; All15]. In contrast, literature in the domain of approximation and online algorithms with proven bounds on the performance is much more scarce. In [MP93], Monma and Potts consider a model where jobs belong to different classes and a setup needs to take place whenever a machine switches from processing jobs of one class to a job of a different class. They consider m identical machines and (in contrast to this thesis) allow the preemption of jobs. They design two simple algorithms, one with an approximation factor of at most $\max\{\frac{3}{2} - \frac{1}{4m-4}, \frac{5}{3} - \frac{1}{m}\}$ if each class is small (i.e., setup time plus size of all jobs of a class are not larger than the optimal makespan), and a second one with an approximation factor of at most $2 - \frac{1}{\lfloor m/2 \rfloor + 1}$ for the general case. For the case of small classes, Chen presents an improved algorithm with approximation factor $\frac{3}{2}$ in [Che93]. Schuurman and Woeginger [SW99] consider the special case where each class only consists of a single job. They design a PTAS for the case of job-independent setup times and a $\frac{4}{3}$-approximation for the case of job-dependent setup times. Jansen et al. improve on this result by giving an EPTAS in [Jan+19]. The same work [Jan+19] also considers a similar model where jobs can not only be preempted but be split arbitrarily (thus, job parts can also be processed simultaneously on different machines). The authors present an EPTAS for the problem in case all machines are identical. For the case of unrelated machines, Correa et al. design a $(1 + \phi)$-approximation, where $\phi \approx 1.618$ is the golden ratio, as well as an inapproximability result of $\frac{e}{e-1}$ (unless P = NP) in [Cor+15].

3.2.3 Flow Time Scheduling

For the objective of minimizing the flow time there are some results for the classical model without setup times. In this case, it is known that FIFO is optimal for minimizing maximum flow time on a single machine. On m parallel machines FIFO achieves a competitiveness of $3 - \frac{2}{m}$ in the preemptive as well as the non-preemptive case as shown by Mastrolilli in [Mas04]. Further results include algorithms for unrelated machines with speed augmentation (which is required as otherwise there is a high lower bound on the achievable competitiveness) given by Anand et al. in

[Ana+17] and for related machines proposed by Bansal and Cloostermans in [BC16] and by Im et al. in [Im+17].

Concerning the model with setup times and a single machine, Divakaran and Saks design and analyze an (online) algorithm in [DS11] having a constant approximation factor for minimizing the maximum flow time. Also, they show that the offline problem is NP-hard in case the number K of classes is part of the input. In case K is a constant, it was known before that the problem can be solved optimally in polynomial time by a dynamic program proposed by Monma and Potts in [MP89].

There are also some results for objectives other than maximum flow time assuming job classes and setup times to be present. In [DS08] Divakaran and Saks give a 2-approximation for the weighted completion time objective and an algorithm achieving a maximum lateness that is at most the maximum lateness of an optimal solution for a machine running at half the speed. In [CVV16], Correa et al. study the objective of minimizing the weighted completion time in the setting where jobs can be split arbitrarily and each part requires a setup before being processed. They propose constant factor approximations for identical and unrelated machines. The total completion time objective is also considered in a setting where jobs consist of multiple operations belonging to different classes. A job is considered to be finished as soon as all its operations are completed and a setup time is incurred when switching between classes. For this problem the NP-hardness for a single machine is known due to Ng et al. [NCY02] as well as an optimal algorithm for a special case due to Gerodimos et al. [Ger+99]. This special case is rather artificial and assumes that the jobs can be renamed in a way so that if job j_i contains an operation of class k, also j_{i+1} contains an operation of class k which is at least as large as the one of j_i.

3.2.4 Smoothed Competitiveness

The notion of smoothed analysis has been introduced by Spielman and Teng in [ST04] to explain the discrepancy between the superior runtime performance of the simplex method in practice and its bad worst-case performance. The idea of this kind of analysis is to randomly perturb the instances dictated by the adversary and analyze the expected performance with respect to the random perturbations. A gentle introduction to this topic is given, for example, in [MR11] and a survey covering some of the numerous results obtained in this area is given in [ST09]. The concept of smoothed analysis has also been carried over to approximation algorithms (see, e.g., [ERV14; Bru+14]) and to competitive analysis [Bec+06]. However, the literature on smoothed approximation ratios and smoothed competitiveness is much more scarce. Particularly, smoothed competitiveness has, although considered as an interesting alternative to classical competitiveness (see, e.g., [HV12; Lóp16]), so far only been applied to two problems. In [Bec+06], Bechetti et al. study the Multilevel Feedback Algorithm for minimizing total flow time on parallel machines when preemption is allowed and non-clairvoyance is assumed. They consider a smoothing model in which integral processing times from the interval $[1, 2^Q]$ as

dictated by an adversary are perturbed by replacing the q least significant bits by a random number from $[1, 2^q]$. They prove a smoothed competitiveness of $O((2^q/\sigma)^3 + (2^q/\sigma)^2 2^{Q-q})$, where σ denotes the standard deviation of the underlying distribution. This, for example, becomes $O(2^{Q-q})$ for the uniform distribution. This result significantly improves upon the lower bounds of $\Omega(2^Q)$ and $\Omega(n^{\frac{1}{3}})$ known for the classical competitiveness of deterministic algorithms [MPT94]. In [SS05], Schäfer and Sivadasan apply smoothed competitive analysis to metrical task systems (a general framework for online problems covering, for example, the paging and the k-server problem). While any deterministic online algorithm is (on any graph with n nodes) $\Omega(n)$-competitive, the authors, amongst others, prove a sublinear smoothed competitiveness on graphs fulfilling certain structural properties. Finally, Scharbrodt et al. apply a notion similar to smoothed competitiveness in [SSS06]. They consider the problem of minimizing the total completion time on parallel machines and analyze the Shortest Expected Processing Time First (SEPT) strategy. While it is $\Omega(n)$-competitive, for processing times drawn from a gamma distribution they prove an expected competitiveness, defined as the expected ratio of SEPT's and Opt's total completion time, of $O(1)$.

CHAPTER 4

Scheduling on Identical Machines with Setup Times

Identical machines define the most basic and well-studied machine environment for parallel machines. In this setting, all machines are assumed to be completely alike so that the processing time of a job does not depend on the machine it is assigned to. In this chapter, we begin our study of scheduling problems with setup times as described in the general model in the previous chapter for such identical machines. Our objective is to assign jobs (and the respective setup operations) to machines so as to minimize the makespan of the resulting schedule, that is, the completion time of the latest job. Optimizing the makespan is a classical objective function that aims at finishing a given set of jobs as fast as possible by balancing the load roughly equally among all available machines. It is particularly interesting from the perspective of a computing center or the owner of a factory as it leads to high utilization and low cost.

Surprisingly, although a lot of research has been done on scheduling with setup times as well as on optimizing the makespan, there were no known results concerning the considered model by the time our results presented in this chapter were first published in [Mäc+15][1]. As discussed in Chapter 3, related problems have been considered from a different perspective aiming at heuristics and exact algorithms. In contrast, we are interested in approximation algorithms and we propose two simple ones in Section 4.2. The first one is a simple strategy yielding 2-approximate solutions for an arbitrary number m of machines while the second one is an FPTAS for the considered problem if m is constant. Section 4.3 presents the main contribution

[1] However, since this first publication, the problem attracted the attention of some researchers and in the course of that a PTAS [JL16] and even an EPTAS [Jan+19] as well as a fast $3/2$-approximation [DJ19] haven been developed. This interest has also led to some joint work, on which Chapter 5 is based.

of this chapter which is an algorithm whose approximation factor can be made arbitrarily close to $3/2$ with a runtime that is polynomial in the input quantities n, m and K. Finally, in Section 4.4 we very briefly study an online version where jobs arrive over time and shortly discuss how to turn, employing a known technique, our offline algorithm into an online strategy with a competitiveness arbitrarily close to 4.

4.1 Model & Notation

We consider the general model as described in Chapter 3 for the case that all m parallel machines are identical. That is, each job $j \in \mathcal{J}$ has a processing time $p_j \in \mathbb{N}_{\geq 0}$ that is the same for all machines so that $p_{ij} = p_j$ for all $i \in \mathcal{M}$. We also assume that the setup times are identical and hence, it holds $s_{ik} = s$ for some $s \in \mathbb{N}_{\geq 0}$ and for all $i \in \mathcal{M}$ and $k \in \mathcal{K}$.

The objective we consider in this chapter is the minimization of the makespan given by the time at which the last job finishes. In this case a schedule is implicitly given by a mapping $\sigma : \mathcal{J} \to \mathcal{M}$, where $\sigma(j)$ defines the machine on which job j is processed. Given σ, one can easily construct an actual schedule by sequencing all jobs of $\sigma^{-1}(i)$ on machine i in an arbitrary order as long as all jobs belonging to the same class are processed consecutively and preceded by a respective setup. Formally, the makespan is then given by the maximal load $\max_{i \in \mathcal{M}} L_i$ where $L_i = \sum_{j \in \sigma^{-1}(i)} p_j + |\{C_k : C_k \cap \sigma^{-1}(i) \neq \emptyset\}| \cdot s$. By abuse of notation we use Opt to denote an optimal schedule as well as its makespan.

For ease of presentation we provide some additional definitions. We refer to the overall processing time of all jobs of a class C_k as its *workload* and denote it by $w(C_k) \coloneqq \sum_{j \in C_k} p_j$ and we assume that for all $k \in \mathcal{K}$ it holds that $w(C_k) \leq \gamma \text{Opt}$ for some constant γ. By abuse of notation, by $w(C_k)$ we sometimes also represent (an arbitrary sequence of) those jobs belonging to class C_k. We say a job $j \in C_k$ forms an *individual class* if $C_k = \{j\}$. The processing time of the largest job in a given instance is denoted by $p_{max} \coloneqq \max_{1 \leq j \leq n}(p_j)$. We say a machine is an *exclusive machine* (of a class C_k) if it only processes jobs of a single class (class C_k). For better readability we use $M_1, \ldots, M_m$ to denote machines in this chapter.

4.2 Simple Approaches

In this section, we briefly discuss some simple results we have for our scheduling problem. We start with a simple and fast algorithm and then consider a computationally more expensive approach that provides an FPTAS for a constant number m of machines.

4.2.1 Fast 2-Approximation

Our main approximation algorithm presented in Section 4.3 follows the idea of a dual approximation algorithm as introduced in Section 2.2 and therefore, requires a good guess on the optimal makespan to initialize the interval on which we perform the binary search. We can find a suitable interval containing the optimal makespan by applying the following algorithm, which provides an interval of length Opt.

Algorithm 1 Description of the simple algorithm.

(1) Create a list of the form $w(C_1), s, w(C_2), s, \ldots, s, w(C_k)$.

(2) Let $B \coloneqq \max\left(s + p_{max}, \left\lceil \frac{Ks+\sum_{j=1}^{n} p_j}{m} \right\rceil\right)$ be a simple lower bound on Opt.

(3) Assign the jobs and setups to machines in the order of the list in the following way:

- If the load assigned to the current machine is smaller than B, the next job (or setup) of the sequence is assigned to the current machine.
- Otherwise, the algorithm proceeds with the next machine and assigns the job to it.

(4) Finally, make the resulting schedule feasible by adding missing setups.

Lemma 4.1. *Algorithm 1 runs in time $O(n \log n)$ and has an approximation factor of at most 2.*

Proof. Note that Step (4) can add at most one setup per machine. Thus, the makespan of the schedule obtained by Algorithm 1 is at most $B + s + p_{max} \leq 2B$ and we only need to show that it requires at most m machines.

For the analysis we take a different perspective on the process (however, it will be clear that the algorithm cannot use more machines than proven next). Note that the length of the sequence given by Step (1) is exactly $(K-1)s + \sum_{j=1}^{n} p_j < mB$. Thus, if we split it at points $\ell B, \ell \in \mathbb{N}$, into blocks of length B, we obtain at most m blocks. Observe that when Algorithm 1 proceeds with a new machine in the else-branch of Step (3) and if this new machine is the i-th one, it has already scheduled all jobs belonging to the first $(i-1)$ blocks. This proves that m machines are sufficient, which concludes the proof. □

The main shortcoming of Algorithm 1 is the naive packing of jobs, which can overfill a machine by almost p_{max} (or, in terms of the proof, the fact that jobs of size p_{max} might get split and then in a feasible schedule need to be processed during the interval $[B, B + p_{max})$. In our main algorithm in Section 4.3, we will address this shortcoming. To this end we will ensure that such situations of overfilling only

happen with jobs that are not too large so that a machine can be overfilled by at most $\approx {}^1/_2 p_{max}$.

4.2.2 Constant Number of Machines

As a second simple result we now show that the problem is rather easy to solve if the number m of machines is upper bounded by a constant. For this case we show how to obtain an FPTAS, i.e., an approximation algorithm that, given any $\varepsilon > 0$, computes a solution with approximation factor at most $1 + \varepsilon$ and runs in time polynomial in n, K and ${}^1/_\varepsilon$. First of all, note that it is simple to enumerate all possible schedules (up to permuting the batches per machine, which does not change the makespan) as follows: Sort the set of jobs according to classes. Let $\mathcal{S}_i, 0 \leq i \leq n$, be the set of all possible (partial) schedules for the first i jobs. Let $\mathcal{S}_0 = \emptyset$ and $j_1, \ldots, j_K$, be the indices i at which there is a change from a job of one class to one of another in the ordered sequence and $j_1 := 1$. To compute $\mathcal{S}_i$, if $i \neq j_1, \ldots j_K$, consider each schedule in $\mathcal{S}_{i-1}$. For each possible assignment of job i to a machine for which a setup took place for i's class k_i put the corresponding schedule into $\mathcal{S}_i$ (if the makespan is not larger than T, others can directly be discarded). If $i = j_\ell$ for some $1 \leq \ell \leq K$, first compute all $2^m - 1$ possible extensions of schedules in $\mathcal{S}_{i-1}$ by setups for i's class k_i and then proceed as in the case before. Obviously, choosing a schedule $S \in \mathcal{S}_n$ with minimum makespan yields an optimal solution.

To obtain an efficient algorithm from this straightforward enumeration of all possible schedules, we first define some dominance relation. This relation helps us to remove schedules during the enumeration process for which there are other schedules that will be at least as good for the overall instance.

Definition 4.2. After computing $\mathcal{S}_i$, a schedule $S \in \mathcal{S}_i$ is *dominated* by $S' \in \mathcal{S}_i$ if

- S and S' have the same load on the first $m - 1$ machines and the load of S' on the m-th machine is smaller and
- in case that $k_i = k_{i+1}$, in S and S' the same machines are set up for i's class k_i.

Note that by removing, for each combination of load values on the first $m - 1$ machines, all dominated and all but at most one non-dominated schedule directly after the computation of $\mathcal{S}_i$ and before the computation of $\mathcal{S}_{i+1}$, we may reduce the size of $\mathcal{S}_i$ without influencing the best obtainable makespan computed at the end in $\mathcal{S}_n$. However, we cannot ensure that the $\mathcal{S}_i$ have a small size. Thus, we consider the following rounding, which is applied before the enumeration: Round up s and the size p_j of each job j to the next integer multiple of ${}^{\varepsilon T}/_{(n+K)}$, where $\varepsilon > 0$ defines the desired precision of the FPTAS. As to any machine we assign at most n jobs and K setups, the rounding may introduce an additive error of at most $\varepsilon \cdot T \leq \varepsilon \cdot \textsc{Opt}$. Additionally, the rounding helps to make sure that each $\mathcal{S}_i$ is not too large after removing the respective (dominated) schedules. Due to our dominance definition, there are at most ${}^{(n+K)}/_\varepsilon$ different load values that may occur in schedules in S_i.

Hence, there are at most $2^m \cdot (^{(n+K)}/_{\varepsilon})^m$ many schedules in $\mathcal{S}_i$ that are not removed, thus proving the following theorem.

Theorem 4.3. *If the number m of machines is bounded by a constant, there is an FPTAS.*

4.3 A $(3/2+\varepsilon)$-Approximation Algorithm

In this section, we present the main algorithm of this chapter. The outline of our approach is as follows:
(1) We first identify a class of schedules that feature a certain structural property. We show that if we narrow our search for a solution to schedules belonging to this class, we still find a good schedule, i.e., one whose makespan is not too far away from an optimal one.
(2) We then show how to perform a rounding of the involved job sizes and further transformations and thereby significantly decrease the size of the search space.
(3) Finally, given such a (transformed) instance, it will be easy to optimize over the restricted class of schedules studied in (1) to obtain an approximate solution to any given instance.

4.3.1 Block-Schedules

We start by discussing the question how to narrow our study to a class of schedules that fulfill a certain property and still, be able to find a provably good approximate solution. Particularly, we focus on block-schedules, which are schedules satisfying a simple structural property, and which we define as follows.

Definition 4.4. Given an instance I, we call a schedule for I *block-schedule* if for all $1 \leq i \leq m$ the following holds: In the (partial) schedule for the machines $M_1, \ldots, M_i$, there is at most one class of which some but not all jobs are processed on $M_1, \ldots, M_i$.

Intuitively speaking, in a block-schedule all jobs of a class are executed in a block in the sense that they are assigned to consecutive machines and are not widely scattered.

In order to prove our main theorem about block-schedules, we first have to take care of jobs having a large processing time in terms of the optimal makespan. Let $L_i = \{j \in C_i : 1/2\textsc{Opt} - s < p_j < 1/2\textsc{Opt}\}$ be the set of *large* jobs of class C_i and $H_i = \{j \in C_i : p_j \geq 1/2\textsc{Opt}\}$ be the set of *huge* jobs of class C_i. Based on these definitions we show the following lemma.

Lemma 4.5. *With an additive loss of s in the makespan we may assume that*

1. *Each huge job forms an individual class,*
2. *There is a schedule with the property that all large jobs of class C_i are processed on exclusive machines, except (possibly) one large job $q_i \in L_i$, for each C_i, and*

3. *$q_i = argmin_{j \in L_i}\{p_j\}$ is the smallest large job in C_i and the machine it is processed on has a load of at most* OPT.

Proof. We prove the lemma by showing how to establish the three properties by transformations of the given instance I and an optimal schedule S for I with makespan OPT. To establish the first property, transform I into I' by putting each job $j \in H_i$ into a new individual class, for each class C_i. Because any machine processing such a huge job j cannot process any other huge or large job due to their definitions, the transformation increases the makespan of any machine by at most s.

Next, we focus on the second property. In S no machine can process two large jobs of different classes. Hence, we distinguish the following two cases: A machine processes one large job or a machine processes at least two large jobs. We start with the latter case and consider any machine that processes at least two large jobs of a class C_i. Because these two jobs already require at least $2\left\lceil (\text{OPT}+1)/2 - s \right\rceil + s \geq \text{OPT} - s + 1$ time units including the setup time, no job of another class can be processed and thus, this machine already is an exclusive machine. On the other hand, if a machine M_p only processes one large job $j \in C_i$, we can argue as follows. The machine M_p works on j for at least $\lceil (\text{OPT}+1)/2 \rceil$ time units (including the setup). Thus, the remaining jobs and setups processed by M_p can have a size of at most $\lfloor (\text{OPT}-1)/2 \rfloor$. If there is still another machine processing a single large job of C_i, we can exchange these jobs and setups with this large job and both involved machines have a makespan of at most $\text{OPT} + s$. Also, the machine from which the large job was removed does not contain any huge or large jobs anymore ensuring there is no machine where this process can happen twice. We can repeat this procedure until all (but possibly one) large jobs are paired so that the second property holds since no machine is considered twice.

Finally, to establish the third property, we can argue as follows: If the smallest large job q_i is the only large one on a machine in the schedule S, we can do the grouping just described without shifting q_i to another machine satisfying the desired bound on the makespan. If q_i is already processed on a machine together with another large job, we may pair the remaining jobs but (possibly) one (which is not processed together with another large job on a machine). In case there is such a remaining unpaired job, we finally exchange q_i with the unpaired job. The resulting schedule fulfills the desired properties. □

We now put the smallest large job q_i of each class C_i into a new individual class. Based on the previous result, there is still a schedule with makespan at most $\text{OPT} + s$ for the resulting instance.

In the next lemma, we directly deduce that there is a block-schedule with makespan at most $\text{OPT} + s$ if we allow some jobs to be split, i.e., some jobs are cut into two parts that are treated as individual jobs and processed on different machines. To this end, fix a schedule S for I fulfilling the properties of Lemma 4.5. By $\tilde{M}$ denote the exclusive machines according to schedule S and by $\tilde{C}_i$ the class C_i without those jobs processed on machines belonging to $\tilde{M}$.

Lemma 4.6. *Given the schedule S fulfilling the properties of Lemma 4.5, there is a schedule S' with makespan at most $\text{OPT}+s$ with the following properties:*

1. *A machine is exclusive in S' if and only if it belongs to $\tilde{M}$ and the partial schedule of these machines is unchanged.*
2. *When removing the machines belonging to $\tilde{M}$ and their jobs from S, we can schedule the remaining jobs on the remaining machines such that*
   a) *The block-property holds and*
   b) *only jobs with size at most $1/2\text{OPT}-s$ are split.*

Proof. Remove machines belonging to $\tilde{M}$ and the jobs scheduled on them from the schedule S obtaining $\tilde{S}$. We now show that there is a schedule S' with the desired properties. Similar to [SW99] consider a graph $G=(V,E)$ in which the nodes correspond to the machines in $\tilde{S}$ and there is an edge between two nodes if and only if in $\tilde{S}$ the respective machines process jobs of the same class. We argue for each connected component of G. Let m' be the number of nodes/machines in this component. Furthermore, let $C'=\{C'_1,\dots C'_l\}$ be the set of classes processed on these machines without those formed by single huge or large jobs and $H=\{h_1,\dots,h_r\}$ be the set of jobs processed on these machines that are either huge jobs or large jobs forming individual classes. Note that $r\leq m'$ since all jobs of H must be processed on different machines in $\tilde{S}$. By an averaging argument we know $\text{OPT}+s\geq \frac{1}{m'}\left(\sum_{i=1}^{l} w(\tilde{C}'_i)+\sum_{i=1}^{r} w(h_i)+(l+r+m'-1)s\right)$ and hence,

$$\sum_{i=1}^{l} w(\tilde{C}'_i)+(l-1)s\leq (m'-r)\text{OPT}+\sum_{i=1}^{r}(\text{OPT}-w(h_i)-s). \tag{4.1}$$

Consider the sequence $w(\tilde{C}'_1),s,w(\tilde{C}'_2),s,\dots,s,w(\tilde{C}'_l)$ of length $\sum_{i=1}^{l} w(\tilde{C}'_i)+(l-1)s$ and split it from the left to the right into blocks of length $\text{OPT}-w(h_1)-s,\dots,\text{OPT}-w(h_r)-s$, followed by blocks of length OPT. Note that each block has non-negative length. By Equation (4.1) we obtain at most m' blocks and by adding a setup to each block and the jobs h_i plus setup to the first r blocks, we can process each block on one machine.

Consequently, if we apply these arguments to each connected component and add the removed exclusive machines again, we have shown that there is a schedule S' with makespan at most $\text{OPT}+s$ satisfying the required properties of the lemma. □

Lemma 4.6 proves the existence of a schedule that almost fulfills the properties of block-schedules, whose existence is the major concern in this section. However, it remains to show how to handle jobs that are split as we do not allow splitting or preemption of jobs. Additionally, we need to describe how to place exclusive machines belonging to $\tilde{M}$, which are not taken care of by the previous lemma, into the obtained schedule in order to yield a block-schedule.

To simplify the description in the following, when we say we place an exclusive machine M_i before machine M_j, we think of a re-indexing of the machines such that the ordering of machines other than M_i and M_j stays untouched but now the new indices of M_i and M_j are consecutive. Also, a job j is *started* at the machine that processes (parts of) j and has the smallest index among all those processing j. A class C_i is processed at the end (beginning) of a machine if there is a job $j \in C_i$ that is processed as the last job (as the first job) on M_j.

Lemma 4.7. *A schedule fulfilling the properties of Lemma 4.6 can be transformed into a block-schedule with makespan at most* $3/2OPT$.

Proof. Consider an arbitrary class C_i. We distinguish three cases depending on where the jobs of C_i are placed in the schedule S' according to the proof of the previous lemma.

(1) There is a job in $\tilde{C}_i$ that is split among two machines M_j and M_{j+1}.

(2) There is no job in $\tilde{C}_i$ that is split.

(3) $\tilde{C}_i = \emptyset$.

In Case (1) there is a job in $\tilde{C}_i$ that is split, i.e., one part is processed until the completion time of M_j and one from time s on by M_{j+1}. Hence, we can simply place all machines of C_i between M_j and M_{j+1}. Since jobs that are split have size at most $1/2\textsc{Opt} - s$, we can process any split job completely on the machine on which it was started increasing its makespan to at most $3/2\textsc{Opt}$. We repeat this process as long as there are jobs left fulfilling Case (1). Note that for each class C_i, after having finished Case (1), there is no split job left.

In Case (2), we distinguish two cases. If the jobs in $\tilde{C}_i$ have an overall size of at most $1/2\textsc{Opt}$ (including setup), there either is no exclusive machine of C_i and hence no violation of the block-property, or we can process the jobs on an exclusive machine of C_i increasing its makespan to at most $3/2\textsc{Opt}$. If the jobs have an overall size of more than $1/2\textsc{Opt}$, we distinguish whether $\tilde{C}_i$ is processed at the end or beginning of a machine M_j or not. In the positive case, we can simply place any exclusive machines of C_i behind or before machine M_j. If $\tilde{C}_i$ is not processed at the end or beginning of a machine M_j, there must be a second class $\tilde{C}_{i'}$ that is processed at the beginning and a third class $\tilde{C}_{i''}$ that is processed at the end of machine M_j. Note that consequently the workload of $\tilde{C}_{i'}$ processed on M_j cannot be larger than $1/2\textsc{Opt} - s$. We can perform the following steps on the currently considered machine M_j:

1. Move all jobs from the class $C_{i'}$ that is processed at the beginning of M_j to machine M_{j-1} if $C_{i'}$ is also processed at the end of M_{j-1}. This only increases the makespan of M_{j-1} by at most $1/2\textsc{Opt} - s$.

2. Move all other jobs processed before some workload of C_i to one of their exclusive machines, if they exist.

3. Shift all the workload $w(\tilde{C}_i)$ to time 0 on machine M_j and shift other jobs to a later point in time.

4. Place all exclusive machines of C_i in front of M_j.

In Case (3), there are only exclusive machines. Such machines can simply be placed behind all other machines.

These steps establish the block-schedule property and no jobs are split anymore. Also note that each machine gets an additional workload of at most $1/2\text{OPT} - s$ without requiring additional setups. Thus, the required bound on the makespan holds, proving the lemma. □

Theorem 4.8. *Given an instance I with optimal makespan* OPT*, there is a transformation to I' and a block-schedule for I' with makespan at most $\text{OPT}_{BL} := \min\{\text{OPT} + p_{max} - 1, 3/2\text{OPT}\}$. It can be turned into a schedule for I with makespan not larger than OPT_{BL}.*

Proof. The bound $\text{OPT}_{BL} \leq 3/2\text{OPT}$ directly follows from Lemma 4.7 and the fact that there are only transformations performed on instance I by Lemma 4.5. The second bound (which gives a better result if $p_{max} \leq 1/2\text{OPT}$) follows by arguments quite similar to those used before: If $p_{max} \leq 1/2\text{OPT}$ holds, we skip the transformation of Lemma 4.5. Additionally, in the proof of Lemma 4.6 we do not remove exclusive machines (thus, considering all machines). Note that, since we skipped the transformation of Lemma 4.5, the set H is empty. Then, it is straightforward to calculate the second bound of $\text{OPT}_{BL} \leq \text{OPT} + p_{max} - 1$. □

4.3.2 Grouping & Rounding

In this section, we show how we can reduce the search space for block-schedules by rounding the involved processing times to integer multiples of some value depending on the desired precision $\varepsilon > 0$ of the approximation. We assume that the transformations described in previous sections have already been performed. In order to be able to ensure that the rounding of processing times cannot increase the makespan of the resulting schedule too much, we first need to get rid of classes and jobs that have a very small workload in terms of OPT_{BL} and ε. In the following, we use $\lambda > 0$ to represent the desired precision, i.e., λ essentially depends on the reciprocal of ε. We call every job j with $p_j \leq \text{OPT}_{BL}/\lambda$ a *tiny job* and every class C_i with $w(C_i) \leq \text{OPT}_{BL}/\lambda$ a *tiny class*.

Lemma 4.9. *Given a block-schedule for an instance I, with an additive loss of at most $4\text{OPT}_{BL}/\lambda$ in the makespan we may assume that tiny jobs only occur in tiny classes.*

Proof. We prove the lemma by applying the following transformations to each class C_i: In a first step, we greedily group tiny jobs of class C_i to new jobs with sizes in the interval $[\text{OPT}_{BL}/\lambda, 2\text{OPT}_{BL}/\lambda)$. In a second step, combine the (possibly) remaining

tiny grouped job $j \in C_i$ with a size less than OPT_{BL}/λ, with an arbitrary other job $j' \in C_i$. By this transformation we ensure that tiny jobs only occur in tiny classes and it remains to show the claimed bound on the makespan.

First of all, focus on the first step of the transformation and assume that we do not perform the second step. Let S be the given block-schedule for instance I. Lemma 2.3 in the work of Shachnai and Tamir [ST00] proves (speaking in our terms) that for the transformed instance there is a schedule S' with makespan of at most $\text{OPT}_{BL} + 2\text{OPT}_{BL}/\lambda$. The proof also implies that S' is still a block-schedule: For each machine M_j it holds that if M_j is configured for class C_i in the new schedule S', it has also been configured for C_i in the original block-schedule S. Thus, if S is a block schedule, so is S' since we do not have any additional setups in S'.

Now assume that also the second step of the transformation is carried out and consider the block-schedule S' we just proved to exist. Distinguish two cases, depending on where the tiny grouped job $j \in C_i$, which was paired in the second step, is processed in schedule S'. If j was paired with a job j' and both j and j' are assigned to the same machine in S', the schedule S' already is feasible for the transformed instance (possibly after shifting j and j' such that they are processed consecutively). If the paired jobs j and j' are processed on different machines in schedule S', there is a schedule whose makespan is by an additive of at most $2\text{OPT}_{BL}/\lambda$ larger than that of S'. To see this, note that in S' this case can happen at most twice per machine (for the classes processed at the beginning and end of the machine). Hence, we can place any paired jobs j and j' on the same machine yielding a schedule for the transformed instance with the claimed bound on the makespan. Finally, note that we can easily turn a schedule fulfilling the claimed bound on the makespan into a schedule for the original instance I satisfying the same bound on the makespan. □

Next, we take care of tiny classes that still might occur in a given instance. Again, without losing too much with respect to the optimal makespan we may assume a simplifying property as shown in the next lemma.

Lemma 4.10. *With an additive loss of at most $4\text{OPT}_{BL}/\lambda$ in the makespan we may assume the following properties:*

1. *Each tiny class consists of a single job.*
2. *In case that $\text{OPT}_{BL}/\lambda > s$, it has size $\text{OPT}_{BL}/\lambda - s$.*

Proof. At first note that with an additive loss of at most $2\text{OPT}_{BL}/\lambda$ in the makespan, we may assume that a tiny class is completely scheduled on one machine in a block-schedule. This is true because of reasons similarly used in the proof of the previous lemma: For each machine it holds that there are at most two different tiny classes of which some but not all jobs are processed on this machine. Hence, we may shift all jobs of such classes to one machine and thereby increase the makespan by at most $2\text{OPT}_{BL}/\lambda$.

Now distinguish two cases depending on whether $\text{OPT}_{BL}/\lambda > s$ or not. If this is the case, determine the length L of the sequence of all tiny classes (including setup times), round up L to an integer multiple of OPT_{BL}/λ, remove all tiny classes from the instance and instead, introduce $\lambda L/\text{OPT}_{BL}$ new classes each comprised of a single job with workload $\text{OPT}_{BL}/\lambda - s$. Observe that, given a block-schedule in which each tiny class is completely scheduled on one machine, we can simply replace tiny classes by these new classes, increasing the makespan by an additive of at most OPT_{BL}/λ. Also, this schedule implies a schedule for the instance in which tiny classes have not been grouped and its makespan is by an additive of at most OPT_{BL}/λ larger. This schedule is simply obtained by again replacing grouped tiny classes by its respective original classes.

In case that $\text{OPT}_{BL}/\lambda \leq s$, we simply group all jobs of a tiny class C_i to a new job j of the same size $p_j = w(C_i)$. Due to the fact that we might assume that a tiny class is completely scheduled on one machine, this proves the lemma. □

From now on, we assume that we have already conducted the grouping from the two previous lemmas and we describe how to round job sizes in order to reduce the search space for later optimization. The rounding approach is quite common for makespan scheduling.

Given an instance I, we compute its rounded version I' by rounding up the size of each job to the next integer multiple of $\text{OPT}_{BL}/\lambda^2$. We know that there is a block-schedule with makespan at most $\text{OPT}_{BL} + 8\text{OPT}_{BL}/\lambda$ and we also assume that the properties from Lemma 4.10 hold.

In case that $\text{OPT}_{BL}/\lambda > s$ each job has either a processing time of at least OPT_{BL}/λ or forms a tiny class with workload at least $\text{OPT}_{BL}/\lambda - s$. On the other hand, in case that $\text{OPT}_{BL}/\lambda \leq s$ and there are tiny classes consisting of a single job, to execute such a job, we need to perform a setup first which yields a processing time of at least OPT_{BL}/λ as well. Hence, we can have at most $\lambda + 8$ jobs on each machine in the considered block-schedule, leading to an additive rounding error of at most $(\lambda + 8) \cdot \text{OPT}_{BL}/\lambda^2$ in the makespan. Therefore, by choosing λ appropriately, there is a solution to the rounded instance that approximates OPT_{BL} up to any desired precision $\varepsilon > 0$.

4.3.3 Optimization over Block-Schedules

We are ready to show how to compute a block-schedule for the rounded instance I' with makespan at most $(1+\varepsilon)\text{OPT}_{BL}$ for any $\varepsilon > 0$. The obtained schedule directly implies a schedule for the original instance I with the same bound on the makespan.

Lemma 4.11. *If all job sizes are a multiple of $\text{OPT}_{BL}/\lambda^2$ and $\lambda > 0$ is a constant, there is only a constant number c_{cl} of different class-types.*

Proof. We can represent any class C_i by a tuple of length λ^2 describing how many jobs of each size $l \cdot \text{OPT}_{BL}/\lambda^2$, $1 \leq l \leq \lambda^2$, occur in class C_i. As each class has a size of at most $\gamma \cdot \text{OPT}$, each entry of the tuple is limited by $\gamma\lambda^2$ and there is at most a

constant number $c_{cl} \coloneqq (\gamma\lambda^2)^{\lambda^2}$ of different tuples describing the classes of I'. In the following we say that all classes represented by the same such tuple are of the same *class-type*, proving the lemma. □

We can represent the classes that have to be scheduled as a tuple of size c_{cl} where each entry contains the number of times classes of the respective class-type occur. Given a block-schedule S, we consider machine configurations that describe which classes are finished on the first i machines. We denote the sub-schedule induced by these first i machines by S_i.

Lemma 4.12. *If all job sizes are a multiple of* $\text{OPT}_{BL}/\lambda^2$ *and* $\lambda > 0$ *is a constant, the number of machine configurations representing* S_i *for some block-schedule* S *and some* $i > 0$ *is bounded by a value* c_{conf} *that is polynomial in* m*.*

Proof. First, note that in a block-schedule S, for every S_i, there is at most one class that is split due to the block-schedule property. Now, to uniquely define a candidate configuration, we need to store information about the classes that are finished, and in case a class has been split, the type of this class and which jobs of this class are finished. We reserve c_{cl} entries for the finished classes, where each entry corresponds to the number of classes of the certain type that has been fully finished. Each entry is at most $m \cdot (\lambda + 8)$ with similar arguments as in the proof of Lemma 4.11 and the reasoning concerning the maximum rounding error. For the class that has been split, we store the type of that class in an extra entry, which gives c_{cl} possible values. If there is no class that has been split, we leave this entry empty adding another possible value to the entry. Finally, we store the number of jobs from the split class that have been finished for each job size as λ^2 additional entries, where each entry does not exceed $c_{cl} \cdot \lambda$ similar to the structure in Lemma 4.11. Overall, we write a configuration as a tuple $(n_1, \ldots, n_{c_{cl}}, j, u_1, \ldots, u_{\lambda^2})$ and thus there are at most $c_{conf} \coloneqq (m(\lambda + 8))^{c_{cl}} \cdot (c_{cl} + 1) \cdot (c\lambda)^{\lambda^2}$ possible configurations. □

We now build a graph where we add a node for each machine configuration. We draw a directed edge from node u to v if and only if the machine configuration corresponding to v can be reached from the configuration u by using at most one additional machine with makespan not larger than $(1+\varepsilon)\text{OPT}_{BL}$. That is, assuming u is a possible sub-schedule induced by the first i machines, we verify whether v is a possible sub-schedule induced by the first $i + 1$ machines. We can do so as we assume that we have guessed OPT correctly and we can hence determine $(1 + \varepsilon)\,\text{OPT}_{BL}$ which is the amount of workload we will fit on one machine. In order to determine the edges of the graph that describes our search space, we prove the following lemma, where we denote $\mathbb{1}_B$ as the indicator variable which is 1 in case the boolean condition B is satisfied and 0 elsewise. Also, we define m_{pk} to be the number of jobs of type k in class-type p, where $k \in \{1, \ldots, \lambda^2\}$ and $p \in \{1, \ldots, c_{cl}\}$.

Lemma 4.13. *If each configuration* $(\vec{n}, j, \vec{u}) = (n_1, \ldots, n_{c_{cl}}, j, u_1, \ldots, u_{\lambda^2})$ *is represented by a node, there is a directed edge from node* $V = (\vec{n}, j, \vec{u})$ *to* $\tilde{V} = \left(\vec{\tilde{n}}, \tilde{j}, \vec{\tilde{u}}\right)$ *if*

and only if

$$\begin{aligned}
&\mathbb{1}_{j\neq\tilde{j}\vee u\neq\tilde{u}}s+\sum_{k=1}^{\lambda^2}\left((\tilde{u}_k-u_k)\cdot k\cdot\frac{\text{OPT}_{BL}}{\lambda^2}\right)\\
&+\sum_{p=1}^{c_{cl}}\left((\tilde{n}_p-n_p)\left(s+\sum_{k=1}^{\lambda^2}m_{pk}\cdot k\cdot\frac{\text{OPT}_{BL}}{\lambda^2}\right)\right)\\
&\leq(1+\varepsilon)\cdot\min\left\{\text{OPT}+p_{max}-1,\frac{3}{2}\text{OPT}\right\}. \qquad (4.2)
\end{aligned}$$

Proof. We prove the statement for the following cases:

1. $j\neq\tilde{j}$:

   First, note that the number of classes of type $p\in\{1,\ldots,c_{cl}\}$ that have been completed between node V and node $\tilde{V}$, i. e. on the additional machine, is expressed in the value $(\tilde{n}_p-n_p)$. Now, in order to finish all jobs from a class of type $p\in\{1,\ldots,c_{cl}\}$, we need to configure the machine for this class and afterward, the workload of all jobs contained in that class type needs to be finished. This leads to an overall processing time of $s+\sum_{k=1}^{\lambda^2}m_{pk}\cdot k\cdot\text{OPT}_{BL}/\lambda^2$ for all jobs of the specific class type. In case the class being finished is j, there is still the same setup time, but there is less workload to be completed. This can be described by subtracting the amount of work already finished in node V, which is $\sum_{k=1}^{\lambda^2}u_k\cdot k\cdot\text{OPT}_{BL}/\lambda^2$. Additionally, to reach the state represented by node $\tilde{V}$, class $\tilde{j}$ needs to be set up and the workload given by $\tilde{u}$ has to be completed yielding an additional processing time of $s+\sum_{k=1}^{\lambda^2}\tilde{u}_k\cdot k\cdot\text{OPT}_{BL}/\lambda^2$. Summing all these times up, we get exactly the value on the left-hand side of Equation (4.2).

2. $j=\tilde{j}\wedge\exists i,u_i>\tilde{u}_i$:

   In this case, we indeed have $j=\tilde{j}$, but as we have $u_i>\tilde{u}_i$ for some i, there are more jobs of type i finished in V than in $\tilde{V}$. Thus, the class that had been partly executed at the end of V needs to be completed and the proof of case 1 similarly applies.

3. $j=\tilde{j}\wedge\forall i,u_i\leq\tilde{u}_i\wedge u\neq\tilde{u}$:

   Here, the scheduler does not necessarily need to finish the class that had been partly executed at the end of V. However, the overall necessary workload is the same whether the work on the current class is only continued and not finished (cost s for setting up the machine for the class) and a new class is fully executed (cost s) or whether it is finished (cost s) and a new class of the same type is initialized (cost s) and not finished. Thus, the proof of case 1 still applies. Note that this also holds if the number of classes of type j that have been fully finished is the same in V and $\tilde{V}$.

4. $j = \tilde{j} \wedge u = \tilde{u}$:

   In this case, we save an overall workload of s in comparison to the other cases. This is due to the fact that we do not need to perform a setup for class j as we can restrict ourselves to executing entire classes.

Combining these cases completes the proof. □

It is time to show that a schedule using only m machines and finishing all jobs exists.

Lemma 4.14. *We can construct a graph G such that there is a path from the node representing no job at all (source) to the node representing the entire instance I' (target) that has a length of at most m.*

Proof. Using Theorem 4.8, there is a block-schedule with makespan at most OPT_{BL}. Due to Lemma 4.9 and Lemma 4.10 together with the additive rounding error and a suitable value for λ depending on ε, there exists a solution to the rounded instance I' with makespan at most

$$(1+\varepsilon)\text{OPT}_{BL} = (1+\varepsilon)\min\{\text{OPT} + p_{max} - 1, 3/2\text{OPT}\}.$$

By construction, the considered graph must contain a path describing this schedule, proving the lemma. Note that this naturally gives an approximation with factor at most $(1+\varepsilon)(\text{OPT} + p_{max} - 1)$ which is better in the case of $p_{max} \leq 1/2\,\text{OPT}$ and which gives a PTAS for unit processing times. □

Theorem 4.15. *By using breadth-first search on G, we can determine a schedule for the original instance I with makespan at most*

$$(1+\varepsilon)\min\left\{\frac{3}{2}\text{OPT}, \text{OPT} + p_{max} - 1\right\}.$$

It implies an algorithm with exactly this approximation guarantee and runtime polynomial in n, m and K.

Proof. Obviously, if we use breadth-first search on the graph, where the source vertex corresponds to the state where no job has been finished and the target vertex corresponds to the state where all jobs have been finished, this gives a path $p = (v_0, v_1, \ldots, v_l)$ of length at most m. By following this path and considering the difference between two consecutive nodes p_{i-1} and p_i, we can efficiently determine the jobs from instance I' to be scheduled on machine M_i. The resulting schedule can be efficiently transformed back into the final schedule for instance I as already discussed during the description of the transformation we apply to I. Also, since the number of nodes is essentially the number of configurations, which in turn is polynomial in m, the search can be carried out efficiently. □

4.4 An Online Variant

While in our original model discussed before we have assumed that all jobs are available at time 0, also online variants can be of interest. Consider a model in which a release time r_j is associated with each job j and a job is not known to the scheduler before r_j, i.e., jobs arrive in an online fashion. The objective remains the minimization of the makespan and we assess the quality of an online algorithm using standard competitive analyses as introduced in Chapter 2: An online algorithm is c-competitive if, for any instance, the makespan of the schedule computed by the online algorithm is by a factor of at most c larger than that of an optimal (offline) solution. It is known that in case there are no setup times, the online LPT rule achieves a competitiveness of $3/2$ [CV97] and that there is a lower bound of ≈ 1.3596 for any deterministic online algorithm [LZ16].

In our case, a very simple lower bound on the competitiveness can be obtained by exploiting the fact that any online algorithm cannot know the class of a job arriving later on in advance and hence, cannot perform a suitable setup operation beforehand. The following lemma shows that this fact results in a lower bound that can be arbitrarily close to 2.

Lemma 4.16. *No online algorithm can be c-competitive for $c \leq 2 - \varepsilon$ and any $\varepsilon > 0$.*

Proof. Consider an instance with (without loss of generality) $m = 2$ machines and the following adversary: At time 0 the adversary releases the first job of some class C_1 with processing time $p_1 = 1$. Then, at time s a second job with processing time $p_2 = 1$ is released belonging to a class for which the online algorithm has not performed a setup yet. Trivially, the optimal algorithm obtains a schedule with makespan $s + 1$ by performing at time 0 a setup for the first job on one machine and one for the second job on the second machine, and then processes the two jobs until time $s + 1$. Any online algorithm cannot do better than performing a setup for the second job at time s and then processing this job. This directly implies a makespan of at least $2s + 1$. Hence, the competitiveness is at least $\frac{2s+1}{s+1}$, which can be arbitrary close to 2 for large setup times s. □

In [SWW91], Shmoys et al. present a quite general technique to turn an offline algorithm for a scheduling problem without release times and an approximation factor of α into a 2α-competitive online algorithm for the respective problems with release times. Although this factor of 2 does not directly carry over to our scheduling problem since we also have to take into account setup operations, a slight modification yields the following result.

Theorem 4.17. *If each job is associated with a release time and jobs are revealed to the scheduler over time at these release times, our algorithm implies a polynomial time c-competitive online algorithm and c can be made arbitrarily close to* 4.

Proof. Although the proof is pretty much the same as that given in [SWW91], for the sake of completeness we state it again. Let 0 be the point in time where the first jobs arrive and call this set of jobs S_0. We apply our approximation algorithm and obtain a schedule for the jobs in S_0 and let F_0 be its makespan. Next we consider those jobs arriving between time 0 and F_0, call the set of them S_1 and compute a schedule for S_1 that begins at time F_0 and ends at time F_1. Generally, we call the set of jobs released during the interval $(F_{i-1}, F_i]$ the set S_{i+1} where F_i is the point in time where the schedule for S_i finishes. Then we schedule S_{i+1} using our approximation algorithm.

Let F_l be the makespan of the entire schedule. We can determine an upper bound on F_l as follows: First, observe that $F_{l-1} \leq F_{l-2} + (1+\varepsilon)(\textsc{Opt} + p_{max} + s)$ since the approximation quality of our algorithm makes sure that we need at most $(1+\varepsilon)(\textsc{Opt} + p_{max} + s)$ time to process the jobs in S_{l-1}. Note that we may need the additional setup time s because the optimal schedule might have already performed necessary setups earlier. Second, consider the instance I' obtained from I by releasing the jobs of S_l at time F_{l-2}. We observe that $F_l - F_{l-1} \leq (1+\varepsilon)(\textsc{Opt}+p_{max}+s) - F_{l-2}$ by the approximation quality of our algorithm and the fact that also the optimal solution cannot schedule jobs of S_l before F_{l-2}. Putting both inequalities together we obtain $F_l \leq 2(1+\varepsilon)(\textsc{Opt} + p_{max} + s) \leq 4(1+\varepsilon)\textsc{Opt}$, proving the theorem. □

CHAPTER 5

Scheduling on Heterogeneous Machines with Setup Times

Heterogeneous machines are a natural generalization of the machine environment of identical machines as considered in Chapter 4. The latter is a very important machine environment to consider as it studies parallel machines on the theoretically most fundamental level. However, in a real setting available machines can often not be considered identical. In contrast, a pool of machines might consist of machines of different generations or with different configurations and capabilities with respect to available peripheral devices, memory, speed or the like. Heterogeneity can even be an intentional design decision motivated by the observation that heterogeneous architectures feature the advantage of machines which are specialized for the processing of certain types of jobs (see, e.g.,[Gup+12]).

In this chapter, we consequently generalize the model from Chapter 4 to heterogeneous machines. We focus on the two classical models for such a machine environment: *unrelated* machines as first mentioned in [BJS74] and *uniformly related* machines as first considered in [HS76]. In the former case, we allow a job to have machine-dependent processing times, which can completely arbitrarily vary on different machines. We study this case in Section 5.2 and we start with a randomized rounding based algorithm to compute $\Theta(\log n + \log m)$-approximations in Section 5.2.1. We then prove that this bound is (asymptotically) tight (unless NP = RP) by providing a reduction from the SetCover problem in Section 5.2.2. We then turn to two special cases in Section 5.3, which both have clear motivations from manufacturing systems, and we show how a rounding technique from [Cor+15] can be employed to approximate these cases with a constant factor. In the latter case, that is, when we have uniformly related machines, each machine runs with a fixed speed, which, however, is the same for all jobs. In Section 5.4 we conclude with simple approximations for this case.

At this point it is also worth mentioning that the considered problem seems to be a quite important topic to study. It is a natural special case of the following problem, which is (with, on average, more than one newly published paper per year since 2010) quite present in the literature on heuristics and exact algorithms as surveyed in [All15], but seems to lack any theoretical investigations with provable performance guarantees: Jobs need to be processed on parallel unrelated machines and each job has a setup time that might depend on the machine as well as the preceding job. Note that in this chapter we require the setup times to have a certain regular structure in the sense that it is 0 for a job j if j is preceded by a job of the same class and otherwise it only depends on j's class and the machine.

5.1 Model & Notation

We consider the problem as described by the general model in Chapter 3 for the case of heterogeneous machines. In the most general model, the *unrelated* machines case, there are no restrictions on the processing times $p_{ij} \in \mathbb{N}_{\geq 0}$. In case of *uniformly related* machines, each machine $i \in \mathcal{M}$ has a fixed speed v_i and the processing time p_{ij} only depends on the job j and the speed of machine i. Hence, it is given by $p_{ij} = \frac{p_j}{v_i}$ for some $p_j \in \mathbb{N}_{\geq 0}$. Finally, we also consider the *restricted assignment* problem, where each job $j \in \mathcal{J}$ has a set $M_j \subseteq \mathcal{M}$ of eligible machines (on which it can be processed) and the processing time is the same on all of them, that is, $p_{ij} = p_j$ for all $i \in M_j$ and some $p_j \in \mathbb{N}_{\geq 0}$, and $p_{ij} = \infty$ otherwise.

For each of these variants we assume that the setup times behave similarly to the jobs. That is, in the unrelated case we have arbitrary setup times $s_{ik} \in \mathbb{N}_{\geq 0}$ depending on the machine i and the class k; in the uniform case we have $s_{ik} = \frac{s_k}{v_i}$; and in the restricted assignment case, we have $s_{ik} \in \{ s_k, \infty \}$. This modeling of the setup times is common in the literature [Bru07] and seems reasonable if we assume that the different behavior is due to qualitative differences between the machines, as suggested by the names of the problems.

Similar to Chapter 4, a schedule is implicitly defined by a mapping $\sigma : \mathcal{J} \to \mathcal{M}$ and the goal is to minimize (over all possible σ) the *makespan* $\max_{i \in \mathcal{M}} L_i$ given by the maximum *load* $L_i \coloneqq \sum_{j \in \sigma^{-1}(i)} p_{ij} + \sum_{k \in \{k_j : j \in \sigma^{-1}(i)\}} s_{ik}$ of the machines. Note that this definition reflects our view on the problem of scheduling with setup times as one can think of the load of a machine as the processing it has to do according to the jobs assigned to it plus the setups it has to perform for classes of which it does process jobs. This reflects that a machine i processes all jobs belonging to the same class in a batch and before switching from processing jobs of a class k' to jobs of class k it has to perform a setup taking s_{ik} time.

For simplicity of notation, for a fixed problem instance and an algorithm $\mathcal{A}$, we denote the schedule as well as the makespan of the schedule computed by $\mathcal{A}$ as $\mathcal{A}$. In this chapter, we always assume that the number m of machines is part of the input. Note that for constant m, the FPTAS from Chapter 4 is also applicable in the setting of (un-)related machines.

5.2 Unrelated Machines

In this section, we study the problem of scheduling unrelated parallel machines with setup times. Recall that for the classical model without setup times it is known [LST90] that it cannot be approximated to within a factor of less than $\frac{3}{2}$ (unless P = NP) and that 2-approximations are possible. This is in stark contrast to our setting where, as we will see, the existence of classes and setups makes the problem significantly harder so that not even any constant approximation factor is achievable. We approach the problem by formulating it as an integer linear program of which we round its optimal fractional solution by randomized rounding. We will see in Section 5.2.1 that this gives a tight approximation factor of $\Theta(\log n + \log m)$. In Section 5.2.2, we turn to inapproximability results and show that under certain complexity assumptions, this factor is essentially optimal.

Let $\ell \coloneqq \max_{j\in\mathcal{J}} \min_{i\in\mathcal{M}}(p_{ij} + s_{ik_j})$ and in the following assume that we have guessed the optimal makespan correctly as $T \in [\ell, n\cdot\ell]$ using the dual approximation framework as defined in Chapter 2. Consider the integer linear program `ILP-UM` as given in Figure 5.1, describing the problem at hand: For each job j, there is an assignment variable $x_{ij} \in \{0,1\}$ stating whether or not job j is assigned to machine i and note that if $p_{ij} > T$, we can require $x_{ij} = 0$ in Equation (5.5). Additionally, for each class k there is one variable $y_{ik} \in \{0,1\}$ indicating whether or not machine i has a setup for class k. Equation (5.1) ensures that the load, given by processed jobs and setups, on each machine does not violate the desired target makespan T. Due to Equation (5.2), each job is completely assigned to one machine, and by Equation (5.3) it is guaranteed that if a job j of class k_j is assigned to machine i, then a setup for class k_j is present on machine i.

5.2.1 Approximation Algorithm

Starting with an optimal solution (x^*, y^*) to the linear relaxation of `ILP-UM` where the variables x_{ij}, y_{ik} can attain any value from $[0,1]$, we can use the following

$$\sum_{j\in\mathcal{J}} x_{ij}p_{ij} + \sum_{k\in\mathcal{K}} y_{ik}s_{ik} \le T \qquad \forall i \in \mathcal{M} \tag{5.1}$$

$$\sum_{i\in\mathcal{M}} x_{ij} = 1 \qquad \forall j \in \mathcal{J} \tag{5.2}$$

$$y_{ik_j} \ge x_{ij} \qquad \forall i \in \mathcal{M}, j \in \mathcal{J} \tag{5.3}$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in \mathcal{M}, j \in \mathcal{J} \tag{5.4}$$

$$x_{ij} = 0 \qquad \text{if } p_{ij} > T \tag{5.5}$$

$$y_{ik} \in \{0,1\} \qquad \forall i \in \mathcal{M}, k \in \mathcal{K} \tag{5.6}$$

Figure 5.1: `ILP-UM` for scheduling unrelated machines.

approach as described in Algorithm 2 to compute an integral solution approximating an optimal schedule. It essentially rounds variables to integral ones with probabilities (w.p.) depending on the optimal fractional solution.

Algorithm 2 Description of the randomized rounding based algorithm.

(1) For each $i \in \mathcal{M}$ and $k \in \mathcal{K}$, set $y_{ik} = \begin{cases} 1, & \text{w.p.} \quad y^*_{ik} \\ 0, & \text{w.p. } 1 - y^*_{ik}. \end{cases}$
We perform a setup for class k on machine i if and only if $y_{ik} = 1$.

(2) For each $i \in \mathcal{M}$ and $k \in \mathcal{K}$ such that $y_{ik} = 1$, and each job $j \in \mathcal{J}$ with $k_j = k$, set $x_{ij} = \begin{cases} 1, & \text{w.p.} \quad x^*_{ij}/y^*_{ik} \\ 0, & \text{w.p. } 1 - (x^*_{ij}/y^*_{ik}). \end{cases}$
We assign job j to machine i if and only if $x_{ij} = 1$.

(3) Repeat Steps (1) and (2) $c \log n$ times.

(4) If there are unassigned jobs left, then schedule each job $j \in \mathcal{J}$ on machine $\operatorname{argmin}_{i \in \mathcal{M}}\{p_{ij} + s_{ik_j}\}$.

(5) If a job is assigned to multiple machines, remove it from all but one. If a setup of a class occurs multiple times on a machine, remove all but one.

The following analysis, though for a different problem, already appeared in a fairly similar way in [KLS10]. However, for the sake of completeness and due to small adaptations, we restate it in the following.

Lemma 5.1. *Step (4) is executed with probability at most* $\frac{1}{n^c}$.

Proof. Consider a fixed job $j \in \mathcal{J}$ and a fixed iteration h of the loop defined by Steps (1) to (3), $1 \leq h \leq c \log n$. Let $\bar{\mathcal{A}}^h_{ij}$ be the event that job j is not assigned to machine i after iteration h (and before iteration $h+1$). Let $\bar{\mathcal{A}}^h_j$ be the event that job j is not assigned to any machine after iteration h. We have

$$\Pr[\bar{\mathcal{A}}^h_{ij} | \bar{\mathcal{A}}^{h-1}_j] = (1 - y^*_{ik_j}) + y^*_{ik_j} \left(1 - \frac{x^*_{ij}}{y^*_{ik_j}}\right) = 1 - x^*_{ij}. \tag{5.7}$$

Taking into account all m machines, we then have

$$\Pr[\bar{\mathcal{A}}^h_j | \bar{\mathcal{A}}^{h-1}_j] \overset{(5.7)}{\leq} \prod_{i \in \mathcal{M}} (1 - x^*_{ij}) \overset{(5.2)}{\leq} \left(1 - \frac{1}{m}\right)^m \leq \frac{1}{e}. \tag{5.8}$$

Hence, for the probability that j is not assigned to any machine after h iterations

we have

$$\begin{aligned}\Pr[\bar{\mathcal{A}}_j^h] &= \Pr[\bar{\mathcal{A}}_j^h | \bar{\mathcal{A}}_j^{h-1}] \cdot \Pr[\bar{\mathcal{A}}_j^{h-1}] \\ &= \ldots = \Pr[\bar{\mathcal{A}}_j^h | \bar{\mathcal{A}}_j^{h-1}] \cdot \Pr[\bar{\mathcal{A}}_j^{h-1} | \bar{\mathcal{A}}_j^{h-2}] \cdot \ldots \cdot \Pr[\bar{\mathcal{A}}_j^1] \\ &\overset{(5.8)}{\leq} \left(\frac{1}{e}\right)^h,\end{aligned}$$

and hence for $h = c \log n$, we obtain the lemma. □

In the next lemma we show that the expected load assigned to a machine per iteration is bounded by $O(T)$. This together with the previous lemma, then proves the final result. Compared to [KLS10], there is a slight difference in our proof: If q_{ij} describes the probability that job j is assigned to machine i in an iteration of the randomized rounding algorithm, then in [KLS10] the authors can (and do) use the fact that $\sum q_{ij} p_{ij} \leq T$. This, however, is not true in our case due to different constraints in the underlying linear program.

Lemma 5.2. *Let L_i describe the load on machine i after the $c \log n$ iterations. Then,* $\Pr[L_i = O(T(\log n + \log m)) \ \forall i \in \mathcal{M}] = 1 - \frac{1}{n^c}$.

Proof. Let us first consider the load on the machines due to processed jobs. Let Z_{ij}^h be a random variable with

$$Z_{ij}^h = \begin{cases} p_{ij}/T, & \text{if } j \text{ is assigned to } i \text{ in iteration } h \\ 0, & \text{otherwise.} \end{cases}$$

Let $Z_i^{\mathcal{J}} = \sum_{h=1}^{c \log n} \sum_{j \in \mathcal{J}} Z_{ij}^h$. Then, we have

$$\mathbb{E}[Z_i^{\mathcal{J}}] = \frac{1}{T} \sum_{h=1}^{c \log n} \sum_{k \in \mathcal{K}} \sum_{j \in \mathcal{J}: k_j = k} \left(0 \cdot (1 - y_{ik}^*) + y_{ik}^* \left(\sum_{j: k_j = k} \frac{x_{ij}^*}{y_{ik}^*} p_{ij}\right)\right) \overset{(5.1)}{\leq} c \log n.$$

Using the essentially same reasoning to analyze the load on the machines due to setups and denoting $Z_i^{\mathcal{S}}$ the analog of $Z_i^{\mathcal{J}}$, we also have $\mathbb{E}[Z_i^{\mathcal{S}}] \leq c \log n$. Because all Z_i are sums of independent random variables with values in $[0, 1]$, we can now apply standard Chernoff-bounds as discussed in Chapter 2 and obtain for $\delta \coloneqq 3\left(\frac{\log(n+m)}{c \log n} + 1\right)$ that $\Pr[\exists i : L_i \geq (1+\delta) T c \log n] \leq \Pr[\exists i, x \in \{\mathcal{S}, \mathcal{J}\} : Z_i^x \geq (1+\delta) c \log n] \leq (m+n) \exp(-\frac{1}{3} \delta c \log n) \leq \frac{1}{n^c}$. □

Taking the last two lemmas together with the fact that the makespan is always upper bounded by $O(T \cdot n)$ if $T \geq \textsc{Opt}$, we obtain the following theorem.

Theorem 5.3. *With high probability and on expectation the randomized rounding approach provides a solution with makespan $O(T(\log n + \log m))$ if there is a schedule with makespan at most T.*

By choosing the parameter c sufficiently large when applying the algorithm within the dual approximation framework, we obtain an approximation factor of $O(\log n + \log m)$. This holds because we only fail to achieve this bound if in any iteration of the dual approximation framework with a makespan guess $T \geq \text{OPT}$, we do not find a schedule with makespan $O(T(\log n + \log m))$. The probability in any such iteration is at most $\frac{1}{n^c}$ and there are only $O(n)$ iterations. Thus, for sufficiently large c and by the fact that the makespan is always upper bounded by $O(\text{OPT} \cdot n)$, we obtain a schedule with (expected) makespan $O((\log n + \log m)\text{OPT})$. Also, it is not too hard to see that this bound is actually tight as one can prove an integrality gap of $\Omega(\log n + \log m)$ for the linear relaxation of `ILP-UM`. This can be shown by using a construction following the ideas for proving the integrality gap for SETCOVER (see, e.g., [Vaz01, p. 111-112]).

Lemma 5.4. *There exists an instance such that* $\text{OPT} \geq \rho C^*$*, where* C^* *is the smallest number such that the linear relaxation of* `ILP-UM` *is feasible and* $\rho = \Omega(\log n + \log m)$*. The statement even holds for the restricted assignment case.*

Proof. Let $N = 2^\kappa - 1$ for some natural number κ. We create N many jobs $\{1, \ldots, N\}$ and assume that we have $m = N$ machines. The processing time of job j on machine i is given by $p_{ij} = 0$ if $i \cdot j = 1$, where $i \cdot j$ denotes the inner product when viewing i and j as κ-dimensional vectors over $GF[2]$, and $p_{ij} = \infty$ otherwise. Let all such created jobs belong to class k_1 and let $s_{ik_1} = 1$ for all $i \in \mathcal{M}$. Now, let the instance I to be considered in the upcoming proof consist of m copies $k_1, k_2, \ldots, k_m$ of k_1.

An optimal (integral) solution needs to process all jobs on machines where they have processing time 0. Using the claim that it has to setup at least κ many machines per class, the optimal integral solution has a makespan of at least $\text{OPT} \geq \kappa$ by a simple averaging argument. The claim can be explained by basic observations from linear algebra: Assume $\kappa' < \kappa$ setups are sufficient for a class k. Let $i_1, \ldots, i_{\kappa'}$ be the indices of machines that have a setup for class k and interpret them as rows of a $\kappa' \times \kappa$ matrix A over $GF[2]$. The rank of the matrix is less than κ and hence, in the nullspace there is a vector v with $Av = 0$. Therefore, the job with index v cannot be processed on a machine where it has processing time 0 as no such machine is setup for class k.

On the other hand, we claim that the LP has a feasible solution with $C^* = 2$. Consider the following fractional assignment $x_{ij} = \frac{2}{m+1} \forall i \in \mathcal{M}, j \in \mathcal{J}$ such that $p_{ij} = 0$ and $x_{ij} = 0$ otherwise, and $y_{ik} = \frac{2}{m+1} \forall i \in \mathcal{M}, k \in \mathcal{K}$. First note that $p_{ij} = 0$ for $(m+1)/2$ machines i so that each job j is completely assigned. Also, the remaining constraints are satisfied. The makespan of the fractional solution is given by m setups that are fractionally assigned by $2/(m+1)$ to each machine and hence, it is less than 2.

Therefore, the integrality gap is $\Omega(\log N) = \Omega(\log m) = \Omega(\log n)$. □

Corollary 5.5. *There is a polynomial time randomized algorithm with approximation factor* $O(\log n + \log m)$*, which matches the integrality gap of the linear relaxation*

of ILP-UM.

5.2.2 Hardness of Approximation

We now show that the approximation factor of $\Theta(\log n + \log m)$ is (asymptotically) optimal unless all problems in NP have polynomial-time Monte Carlo algorithms. Recall that the complexity class RP (Randomized Polynomial-Time) is defined as the class of problems L for which there is a randomized algorithm running in polynomial time and with the following properties (see, e.g., [Gol08]):

- If the input $x \notin L$, the algorithm outputs "No" with probability 1.
- If the input $x \in L$, the algorithm outputs "Yes" with probability at least $\frac{1}{2}$.

Therefore, if such an algorithm outputs "Yes", it provides the correct answer; if it, however, outputs "No", it might err.

We show the following result on the hardness of approximating our problem on unrelated machines.

Theorem 5.6. *Scheduling with setup times on unrelated machines cannot be approximated within a factor of $o(\log n + \log m)$ in polynomial time unless NP = RP. This even holds for the restricted assignment case.*

To do so, we reduce from the following formulation of the well-known SetCover problem: In SetCoverGap we are given a universe $\mathcal{U}$ of $N := |\mathcal{U}|$ elements and a collection $\{S_1, S_2, \ldots, S_m\}$ of m subsets of $\mathcal{U}$. The goal is to decide whether there is a solution covering $\mathcal{U}$ that consists of t subsets or if (at least) αt subsets are needed. We call an instance with the former property a Yes-instance and with the latter a No-instance. A result from [AMS06] shows the following lemma.

Lemma 5.7 (Theorem 7 in [AMS06])**.** *There exists a t such that it is NP-hard to decide* SetCoverGap *for $\alpha = \Theta(\log N)$ and $\log m = O(\log N)$.*

The idea of our reduction is to exploit the apparent connection between SetCover and our unrelated machines variant: Each set is mapped to a machine and each element is mapped to a job. A machine can process a job if and only if the respective set contains the respective element. Additionally assuming that all jobs belong to the same class, by this we see that a Yes-instance requires much less setups than a No-instance. Unfortunately, this not yet leads to a respectively high and small makespan. However, by creating a larger number of classes and randomizing the mapping between sets and machines, we can achieve a (more or less) even distribution of setups that need to be done and hence, depending on the type of the SetCoverGap instance, a large or small makespan. We formalize this idea in the proof of Theorem 5.6.

Proof. Given an instance I for SetCoverGap, we construct an instance I' for our problem with the following properties:

1. The reduction can be done in polynomial time and I' consists of $n = \Theta(N^c)$ jobs, for some constant c.

2. If I is a No-instance, then I' has a makespan of at least $\Omega\left(\frac{K}{m} \cdot \alpha t\right)$.

3. If I is a Yes-instance, then I' has a makespan of at most $O\left(\frac{K}{m} \cdot t\right)$ with probability at least $\frac{1}{2}$.

Consequently, there is a gap of $\Omega(\alpha)$ and by Property 1. and Lemma 5.7, $\alpha = \Omega(\log n)$ and $\alpha = \Omega(\log m)$ and the existence of a polynomial-time algorithm with approximation factor $o(\log n + \log m)$ for our problem makes the problem SetCoverGap solvable by a Monte Carlo algorithm in polynomial time, yielding the theorem.

We now show how to construct I'. In instance I' there are m unrelated machines and $K = \frac{m}{t} \log m$ classes. All setup times are set to be 1, that is, $s_{ik} = 1$ for all $i \in \mathcal{M}, k \in \mathcal{K}$. The jobs $\{j_1^k, j_2^k, \ldots, j_N^k\}$ of class $k = 1, 2, \ldots, K$ are defined by the N elements in I in the following way: We choose a permutation $\pi_k : \mathcal{M} \to \mathcal{M}$ at random (and independent from the choices of $\pi_{k'}$ for $k' \neq k$). Then, for each element e in the SetCoverGap instance I, we create a job j_e^k in instance I' that has a size $p_{ij_e^k} = 0$ if $e \in S_{\pi_k(i)}$ and $p_{ij_e^k} = \infty$ otherwise.

Next, we take a look at the makespan of I' if I is a No-instance. In this case, at least αt sets are needed to cover all elements. However, this implies that for each class at least that many machines are needed to process all jobs (or otherwise the makespan is ∞). Therefore, by summing over all K classes, at least $K \cdot \alpha t$ setups need to be performed. By an averaging argument this leads to the existence of a machine with a load of at least $\frac{K}{m} \cdot \alpha t$.

We now turn our attention to the case where I is a Yes-instance and show that with probability at least $\frac{1}{2}$ there is a solution with makespan $O(\frac{K}{m} \cdot t)$. To this end, we setup a machine i for class k (and process all jobs j of class k on machine i that fulfill $p_{ij} = 0$) if $S_{\pi_k(i)}$ is part of the solution to I. Therefore, each class is setup on t of the m machines. For a fixed machine i and a fixed class k, the probability that i is setup for k is consequently $\frac{t}{m}$ since $\pi_k(i)$ is chosen uniformly at random. Also, the probability that i is setup for all classes of a fixed subset of r classes is $(\frac{t}{m})^r$ as the π_k are chosen independently. Therefore, the probability that a fixed machine i is setup for at least r classes is upper bounded by

$$\binom{K}{r}\left(\frac{t}{m}\right)^r \leq \left(\frac{Ket}{rm}\right)^r.$$

Hence, for the probability that there is some machine which is setup for at least $r := 2Ket/m + 2\log m = O\left(\frac{K}{m} \cdot t\right)$ classes is (for $m \geq 2$) upper bounded by

$$m \cdot \left(\frac{Ket}{rm}\right)^r \leq m \cdot \left(\frac{1}{2}\right)^{2\log m} \leq \frac{1}{m} \leq \frac{1}{2}.$$

Therefore, I' has a makespan of at most $O(\frac{K}{m} \cdot t)$ with probability at least $\frac{1}{2}$.

Also note that $\log(n) = \log(K \cdot N) \leq \log(m \log m \cdot N) = O(\log N)$, where the last equality holds due to the polynomial relation between m and N according to Lemma 5.7. This concludes the proof. □

5.3 Special Cases of Unrelated Machines

In this section, we identify and approximate two special cases of unrelated machines, for which constant factor approximations are possible. Both cases require classes to have certain structural properties that make the reduction and hence, the inapproximability from Section 5.2.2 invalid: Either we consider the restricted assignment case with the additional assumption that the set of eligible machines is the same for all jobs of each class, or we assume that, on each machine, all jobs of a given class have the same processing times. These two cases are considered in Section 5.3.1 and Section 5.3.2, respectively.

5.3.1 Restricted Assignment with Class-uniform Restrictions

Although even the restricted assignment variant of our scheduling problem cannot be approximated with a factor of $o(\log n)$ as shown in Theorem 5.6, we will now see that the following special case admits a much better approximation factor. Let the *restricted assignment problem with class-uniform restrictions* be defined as the restricted assignment problem with the additional constraint that for all $j, j' \in \mathcal{J}$ with $k_j = k_{j'}$ it holds $M_j = M_{j'}$. That is, all jobs of a class k have the same set of eligible machines and by abuse of notation call this set M_k. This case might have applications in, for example, a setting in which machines can be equipped with different tools; jobs of the same class require the same set of tools at the processing machine, the change of tools at a machine requires a setup and there are machine eligibility restrictions. An application with these characteristics (and, however, additionally sequence-dependent setups, release times and a different objective function) is described in [GM12] for the production of gears.

Note that we can add the following valid constraints given by Equations (5.9) to (5.11) to `ILP-UM`:

$$\sum_{j:k_j=k} x_{ij}p_{ij} + y_{ik}s_{ik} \leq y_{ik}T \qquad \forall i \in \mathcal{M}, k \in \mathcal{K} \tag{5.9}$$

$$x_{ij} = 0 \qquad \forall i \in \mathcal{M}, j \in \mathcal{J} : p_{ij} + s_{ik_j} > T \tag{5.10}$$

$$y_{ik} = 0 \qquad \forall i \in \mathcal{M}, k \in \mathcal{K} : s_{ik} > T \tag{5.11}$$

Equation (5.9) holds because a job of a class k can only by processed on a machine i if this machine is setup for class k. Additionally, Equations (5.10) and (5.11) avoid the assignment of jobs to machines where the setup or the job's processing time is too large. Let `ILP-RA` denote the program given by Equations (5.1) to (5.3) and (5.9) to (5.11). Unfortunately, we do not know how to round a solution to the

$$\sum_{k\in\mathcal{K}} \bar{x}_{ik}(\bar{p}_{ik} + \alpha_{ik}s_{ik}) \leq T \qquad \forall i \in \mathcal{M} \tag{5.12}$$

$$\sum_{i\in\mathcal{M}} \bar{x}_{ik} = 1 \qquad \forall k \in \mathcal{K} \tag{5.13}$$

$$\bar{x}_{ik} \geq 0 \qquad \forall i \in \mathcal{M}, k \in \mathcal{K} \tag{5.14}$$

$$\bar{x}_{ik} = 0 \qquad \forall i \in \mathcal{M}, k \in \mathcal{K} : s_{ik} > T \tag{5.15}$$

Figure 5.2: Description of `LP-RelaxedRA`.

linear relaxation of `ILP-RA` to a good approximation for our problem. However, instead we formulate a different, relaxed linear program `LP-RelaxedRA` as described in Figure 5.2, which we will utilize for our approximation algorithm. This linear program takes a different view in the sense that it does not operate on the level of jobs but instead it has a variable $\bar{x}_{ik}$ for each class-machine-pair determining the fraction of (the workload of) class k processed on machine i. Therefore, for $i \in M_k$ let $\bar{p}_{ik} \coloneqq \bar{p}_k$ where $\bar{p}_k \coloneqq \sum_{j:k_j=k} p_{ij}$ and for $i \notin M_k$ let $\bar{p}_{ik} = \infty$. Also, let $\alpha_{ik} \coloneqq \max\left\{1, \frac{\bar{p}_{ik}}{T-s_{ik}}\right\}$. If x is a feasible solution to `ILP-RA`, then $\bar{x}$ with $\bar{x}_{ik} \coloneqq \sum_{j:k_j=k} x_{ij}\frac{p_{ij}}{\bar{p}_{ik}}$ is a feasible solution to `LP-RelaxedRA` as the next lemma proves.

Lemma 5.8. *Let x be a feasible solution to `ILP-RA`. Then $\bar{x}$ is a feasible solution to `LP-RelaxedRA`.*

Proof. First, note that Equation (5.15) directly follows from Equations (5.10) and (5.11). Equation (5.13) is satisfied as we have

$$\begin{aligned}\sum_{i\in\mathcal{M}} \bar{x}_{ik} = \sum_{i\in\mathcal{M}} \sum_{j:k_j=k} x_{ij}\frac{p_{ij}}{\bar{p}_{ik}} &= \sum_{i\in M_k} \frac{1}{\bar{p}_{ik}} \sum_{j:k_j=k} x_{ij}p_{ij} \\ &= \frac{1}{\bar{p}_k} \sum_{j:k_j=k} p_j \sum_{i\in M_k} x_{ij} = 1,\end{aligned}$$

where the first equality follows by definition of $\bar{x}_{ik}$, the second because $\bar{x}_{ik} = 0$ if $i \notin M_k$ and the last one because $\sum_{i\in M_k} x_{ij} = 1$ by Equation (5.2).

To see why Equation (5.12) holds, first observe that for x we have

$$\sum_{j\in\mathcal{J}} x_{ij}p_{ij} + \sum_{k\in\mathcal{K}} \max\left\{\max_{j:k_j=k} x_{ij}, \frac{\sum_{j:k_j=k} x_{ij}p_{ij}}{T - s_{ik}}\right\} s_{ik} \leq T \;\forall i \in \mathcal{M} \tag{5.16}$$

due to Equations (5.1), (5.3), and (5.9). Then we have

$$\begin{aligned}\sum_{k\in\mathcal{K}} \bar{x}_{ik}(\bar{p}_{ik} + \alpha_{ik}s_{ik}) &= \sum_{k\in\mathcal{K}} \sum_{j:k_j=k} x_{ij}\frac{p_{ij}}{\bar{p}_{ik}} \sum_{j:k_j=k} p_{ij} + \sum_{k\in\mathcal{K}} \sum_{j:k_j=k} x_{ij}\frac{p_{ij}}{\bar{p}_{ik}}\alpha_{ik}s_{ik} \\ &= \sum_{j\in\mathcal{J}} x_{ij}p_{ij} + \sum_{k\in\mathcal{K}} \frac{\alpha_{ik}}{\bar{p}_{ik}} \sum_{j:k_j=k} x_{ij}p_{ij}s_{ik} \leq T,\end{aligned}$$

where the first and second equality follow from definition of $\bar{x}_{ik}$ and $\bar{p}_{ik}$ respectively, and the last inequality holds due to Equation (5.16). □

`LP-RelaxedRA` is identical to the LP given in [Cor+15]. There it is shown that an extreme solution to the LP can be rounded to a solution with makespan at most $2T$ that is feasible for the problem where jobs can be split arbitrarily but each part requires a (job-dependent) setup. Interestingly, even though in our model setups are associated with classes and even more crucial, we do not allow jobs to be split, the (essentially) same approach they use provides an approximation factor of 2 for our problem, too. The high-level idea how to obtain a 2-approximation based on an optimal (extreme) solution for `LP-RelaxedRA` is as follows: It is known that due to the structure of `LP-RelaxedRA`, the (bipartite) graph on the node set given by the classes and machines and edges between pairs of nodes whose variable is nonzero in the LP's solution is a pseudo-forest. We can exploit this fact to modify the solution such that it has a makespan of at most $2T$ and the following properties: Each machine processes at most one class partly (but not completely) and, for each class k, from the set of machines processing parts of k at most one machine has a load larger than T. This allows us to greedily assign the actual jobs according to the (modified) fractional solution to the machines and thereby increasing the load per machine (with load at most T) by at most one setup plus one job of the same class and hence, by at most T. The details are given next and for the sake of completeness, we restate the rounding procedure together with its properties from [Cor+15]: Given an extreme solution $\bar{x}^*$ to `LP-RelaxedRA`, all variables $\bar{x}_{ik}$ with $\bar{x}^*_{ik} \in \{0, 1\}$ will remain unchanged, are excluded from our further considerations and class k is processed on machine i if $\bar{x}^*_{ik} = 1$. Let $G = (V, E)$ be the bipartite graph on node set $V = \mathcal{K} \setminus \{k : \exists i \text{ with } \bar{x}^*_{ik} = 1\} \cup \mathcal{M}$ and edge set $E = \{\{i, k\} : 0 < \bar{x}^*_{ik} < 1\}$. G forms a graph in which each connected component is a pseudotree, which was already exploited in [LST90] for scheduling unrelated machines without setup times. For the sake of rounding, we now construct a subset $\tilde{E} \subseteq E$ of edges as follows (it is a slight simplification of the construction in [Cor+15], which, however, leads to the same result in Proposition 5.9): For each connected component, let j be an arbitrary class node if the component does not contain a cycle. Otherwise, remove an arbitrary cycle edge $\{i, j\}$ where, again, j is a class node. Now root the (resulting) tree in j and direct all edges away from the root. The set of edges leaving class nodes forms the set $\tilde{E}$.

The following observation is a direct consequence of the construction.

Proposition 5.9. *By the construction described above, we have the following two properties for a schedule induced by $\bar{x}$:*

1. *Each machine i processes at most one class k with $\{i,k\} \in \tilde{E}$ and $0 < x_{ik} < 1$, and*

2. *for each class k there is at most one machine i such that $\{i,k\} \notin \tilde{E}$ and $\bar{x}^*_{ik} > 0$.*

For each class k we choose an arbitrary machine i_k^+ such that $\{i_k^+, k\} \in \tilde{E}$. If there is a machine i_k^- such that $\bar{x}^*_{i_k^- k} > 0$ but $\{i_k^-, k\} \notin \tilde{E}$, we move all workload of k processed on i_k^- from i_k^- to i_k^+ and add a (full) setup for k to machine i_k^+. By this and Proposition 5.9 we then have the property that each machine processes at most one class fractionally. Let $M(k)$ be the set of machines that process (parts of) class k. Next, we prove the following lemma.

Lemma 5.10. *For any $k \in \mathcal{K}$, all machines in $M(k) \setminus \{i_k^+\}$ have a load of at most T. The load of i_k^+ is upper bounded by $2T$.*

Proof. Consider a class k. Note that $i_k^+ \neq i_{k'}^+$ for all $k' \neq k$ by Proposition 5.9. Hence, the first statement of the lemma holds. The second statement follows by an observation already made in [Cor+15]: By the constraints of `LP-RelaxedRA`, $\alpha_{i_k^- k} \bar{x}^*_{i_k^- k} \leq 1$ and by definition of $\alpha_{i_k^- k}$, we have $T \geq \bar{p}_{i_k^- k} / \alpha_{i_k^- k} + s_{i_k^- k}$. Taken together, $\bar{x}^*_{i_k^- k} \bar{p}_{i_k^- k} + s_{i_k^- k} \leq T$ and because we consider restricted assignment with class-uniform restrictions, we also have $\bar{x}^*_{i_k^- k} \bar{p}_{i_k^+ k} + s_{i_k^+ k} \leq T$, proving the lemma. □

Finally, we need to explain how to obtain the final feasible schedule with makespan at most $2T$. Obtaining a feasible schedule from the solution so far, requires adding a (full) setup for class k on all machines $i \in M(k) \setminus \{i_k^+\}$ as well as showing how to actually assign the jobs of k to the machines $i \in M(k)$. We say that a time slot of size x is reserved for class k on a machine i if $\bar{x}^*_{ik} \bar{p}_{ik} = x$. For any fixed class k, sort the machines in $M(k)$ so that machine i_k^+ comes last in this ordering. Starting with the first machine in the ordering, take the jobs of k and greedily fill them into the reserved time slots by assigning the current job to the current machine if the reserved time slot is not yet full. As soon as a machine is full, proceed with the next machine. It is not hard to see that by this procedure the load of each machine $i \in M(k) \setminus \{i_k^+\}$ is increased by an additive of at most $s_{ik} + \max_{j:k_j=k} p_{ij}$, which is upper bounded by T due to Equation (5.15) and the fact that also an optimal solution has to process a setup and the largest job of class k on some machine. The last machine i_k^+ keeps its load of at most $2T$. Therefore, we have proven the desired result.

Theorem 5.11. *The restricted assignment problem with class-uniform restrictions admits a 2-approximation.*

5.3.2 Unrelated Machines with Class-uniform Processing Times

A second special case that allows constant factor approximations is the one of unrelated machines in which all jobs of a given class have the same processing times on any machine. That is, for all $i \in \mathcal{M}$ and $j, j' \in \mathcal{J}$ it holds $k_j = k_{j'} \Rightarrow p_{ij} = p_{ij'}$.

This special case of unrelated machines might have applications prevalently in production systems. One example is mentioned in [Kim+02]: In the production of semiconductor wafers, machines for dicing operations are non-identical due to varying ages and manufacturers. The machines can produce different types of wafers and each machine has its own processing times depending on the characteristics of the machine and the wafer type while each item of a given type has identical processing times. Also, machines need to be adjusted whenever different types of wafers are diced leading to setup times, which, however, do not occur between items of the same type.

We solve this problem similarly to the restricted assignment problem with class-uniform restrictions in the previous section. To do so, we modify the approach as follows: First of all, we replace Equation (5.15) in `LP-RelaxedRA` by

$$\bar{x}_{ik} = 0 \;\; \forall i \in \mathcal{M}, k \in \mathcal{K} : s_{ik} + p_{ij} > T \text{ for some } j \text{ with } k_j = k. \tag{5.17}$$

Note that this is a valid constraint since all jobs of a class k have the same size on machine i and if a job together with its class' setup does not fit to a machine, no workload of class k will be assigned to i at all. Then we construct the set $\tilde{E}$ with the properties of Proposition 5.9 as before. Now, for each class $k \in \mathcal{K}$ let i_k^- be the machine such that $\bar{x}^*_{i_k^- k} > 0$ but $\{i_k^-, k\} \notin \tilde{E}$ (if it exists). Let $i_{k,\iota}^+$, $\iota = 1, \ldots, \iota_k$ be the machines such that $\bar{x}^*_{i_{k,\iota}^+ k} > 0$ and $\{i_{k,\iota}^+, k\} \in \tilde{E}$. In case $\bar{x}^*_{i_k^- k} > \frac{1}{2}$, process the entire class k on machine i_k^-. Otherwise, distribute the amount of k processed on i_k^- proportionally to the machines $i_{k,\iota}^+$ so that $\bar{x}^*_{i_k^- k}$ is set to 0 and $\bar{x}^*_{i_{k,\iota}^+ k}$ is at most doubled. After these steps, the load on each machine is at most $2T$. Finally, it remains to add at most one setup to each machine and, as before, to greedily fill the reserved slots by the actual jobs. This increases the load on each machine by an additive of at most T due to Equation (5.17) and hence, we have constructed a 3-approximation. Together with a straightforward adaptation of the reduction given in [Cor+15], we have the following result.

Theorem 5.12. *The unrelated machines case with class-uniform processing times admits a 3-approximation. It cannot be approximated to within a factor less than 2 unless $P = NP$.*

Proof. Consider the following variant of the SetCover problem called Max k-Cover: Given a universe of n elements and a collection $\{S_1, S_2, \ldots, S_m\}$ of m subsets of it. The goal is to choose k sets so as to maximize the number of covered elements. It is known that for the variant where all sets have cardinality $\frac{n}{k}$ it is NP-hard to decide whether all elements can be covered with k disjoint sets or if no k sets can

cover more than a $(1 - \frac{1}{e})$-fraction of all elements. Given an instance I for Max k-Cover, we construct an instance I' for our scheduling problem as follows: For each set S_i, we create a class C_i consisting of $\frac{n}{k}$ jobs. We have $m - k$ universal machines on which all classes have a setup time of 1 and all jobs have a size of 0. Additionally, we create one machine M_i for each element e_i in instance I: If $e_i \in S_\kappa$, class κ has a setup time 0 on machine M_i and all its jobs have processing time 1. Otherwise, the setup time and all processing times of class κ are ∞.

Now assume that I is a Yes-instance. Consider the classes corresponding to the k (disjoint) sets covering the universe. We can schedule all their jobs on the respective element-machines such that each machine gets exactly one job with processing time 1 and respective setup of size 0. The remaining $m - k$ classes can be processed on the $m - k$ universal machines, assigning a complete class to each of them. Since the setup time is 1 and the jobs' processing times are 0, we have a schedule with makespan 1.

Next, let I be a No-instance. Note that each universal machine can have a setup of at most one class or otherwise the makespan is directly at least 2. Then, at most $m - k$ classes can be processed on universal machines. The remaining k classes need to be assigned to element-machines. However, for any k classes, their accumulated processing time of n needs to be processed on at most $(1 - \frac{1}{e})n$ element machines. Averaging leads to the fact that there is a machine with a load of at least $\lceil e/(e-1) \rceil = 2$. □

5.4 Uniformly Related Machines

We conclude this chapter by briefly considering the case of uniformly related machines. As already mentioned in the introduction, the paper this chapter is based on provides a PTAS for this problem. Here, we focus on approaches that are undeniably inferior with respect to the approximation factor but that might be more practical as they are much simpler or faster.

We start with a simple and very general way that shows how to use existing approaches for scheduling uniformly related machines without setup times to solve our problem with setup times. For a given instance I, let $J_s^k = \{j \in \mathcal{J} : k_j = k, p_j < s_k\}$ be the set of jobs of class k being smaller than the setup time of k. Consider the modified instance I' in which all jobs of $\mathcal{J} \setminus (\bigcup_{i=1}^{K} J_s^k)$ are given as in I and all jobs of J_s^k are replaced by $\lceil \sum_{j \in J_s^k} p_j / s_k \rceil$ many (placeholder) jobs of class k, each with a size of s_k. Then apply a standard algorithm for scheduling uniformly related machines, ignoring any classes and not scheduling any setups to obtain an (infeasible) schedule for I'; and finally obtain a feasible schedule for the original instance I by adding all required setups to the computed schedule and replacing the placeholder by the actual jobs. This can be done by replacing a placeholder job of class k by choosing any unscheduled jobs from J_s^k until their summed up size first exceeds the size of the placeholder job or there are no unscheduled jobs left. We have the following lemma.

Lemma 5.13. *Let $\mathcal{A}$ be an algorithm with approximation factor α for scheduling uniformly related machines without setup times. Applying $\mathcal{A}$ in the framework described above, yields a 3α-approximation for scheduling uniformly related machines with setup times.*

Proof. Consider an optimal schedule σ for I and let S_i be the set of classes for which there is a setup on machine i. Then, there is also a schedule for instance I' with load at most $\textsc{Opt} + \sum_{k \in S_i} s_{ik}$ on machine i by scheduling $\lceil \sum_{j \in \sigma^{-1}(i) \cap J_s^k} p_{ij}/s_{ik} \rceil$ many placeholders of class k on machine i. Also, when not scheduling any setups, this load is decreased to at most $\textsc{Opt}$. Hence, there is a schedule for I' with makespan at most $\textsc{Opt}$ when no setups are scheduled. Therefore, using $\mathcal{A}$ and ignoring classes and not scheduling setups, we find a schedule with makespan at most $\alpha\textsc{Opt}$. It remains to insert setups and to replace the placeholder by the actual jobs to obtain a feasible solution for I. Let C_i be the set of classes of which jobs are scheduled on machine i in the schedule obtained by $\mathcal{A}$. Replacing the placeholder by actual jobs can increase the makespan by at most $\sum_{k \in C_i} s_{ik}$ and adding the required setups can increase it by the same amount. Recall that the schedule computed by $\mathcal{A}$ has a makespan of at most $\alpha\textsc{Opt}$ and there is at least one job of size at least s_{ik} on machine i for each $k \in C_i$. Therefore, $\sum_{k \in C_i} s_{ik} \leq \alpha\textsc{Opt}$ and the lemma follows. □

Note that the best bound Lemma 5.13 implies is a $(3+\varepsilon)$-approximation, which we obtain by using a PTAS such as [Alo+98]. When aiming at simple and very fast solutions, one could use the LPT-rule as algorithm $\mathcal{A}$ in Lemma 5.13 leading to an approximation factor of $3(1+\frac{1}{\sqrt{3}}) \approx 4.74$ [Kov10]. As LPT just sorts all jobs of I' by non-increasing size and adds one after the other to the machine where it finishes first, this approach has a very efficient runtime of $O(n \log n)$.

Next, we show how to come up with a very efficient algorithm, described in Algorithm 3, that improves on the best bound that can be achieved by using Lemma 5.13 in case we have class-independent setup times, that is, $s_k = s$ for all $k \in \mathcal{K}$.

Theorem 5.14. *Algorithm 3 is a 3-approximation algorithm.*

Proof. Let $\{M_1, M_2, \ldots, M_\ell\}$ be those bins containing jobs of J_s. Let $j_1, j_2, \ldots, j_q$ be the jobs packed into M_i (in this order). We first analyze the load of M_i, $1 \leq i < \ell$. We can upper bound the load of M_i by $L_i \leq \sum_{j=1}^{q-1}(\frac{p_{j_j}}{v_i} + \frac{s}{v_i}) + \frac{p_{j_q}}{v_i} + \frac{s}{v_i}$. By definition of J_s and the fact that we overpack each bin by at most one job, we then also have $\sum_{j=1}^{q-1}(\frac{p_{j_j}}{v_i} + \frac{s}{v_i}) \leq 2T$. Finally, $\frac{p_{j_q}}{v_i} + \frac{s}{v_i} \leq T$ since $\textsc{Opt}$ cannot pack more jobs into the bins $M_1, \ldots, M_{\ell-1}$ and because we have packed jobs in non-increasing order of their size. Thus, for $1 \leq i < \ell$, we have $L_i \leq 3T$.

Next, consider the load of M_ℓ. In case no jobs of $\mathcal{J} \setminus J_\ell$ are packed into M_ℓ, $L_\ell \leq 3T$. Otherwise, let j be the last job packed into M_ℓ. Then, the load is upper bounded by $L_\ell \leq 2T + \frac{s}{v_\ell}$. We defer the bounding of $\frac{s}{v_\ell}$ by T to the end and first proceed with the bins M_i for $i > \ell$.

Note that the amount of jobs and setups our algorithm packs in Step (3) cannot be larger than what $\textsc{Opt}$ has to pack into bins M_i with $i > \ell$. Hence, if m' is the

Algorithm 3 Description of NextFit algorithm for uniformly related machines with class-independent setup times.

(1) Let $J_\ell = \{j \in \mathcal{J} : p_j \geq s\}$ and consider M_i as a bin of size $T \cdot v_i$.

(2) Pack all jobs of J_ℓ by filling bin by bin in order of non-increasing size.
- Sort jobs in J_ℓ in non-increasing order of their sizes p_j.
- Add jobs in sorted order to bins in a next-fit manner by adding current job to current bin as long as it is not full. Otherwise proceed with the next bin.

(3) Pack all remaining jobs and one setup per involved class.
- Sort jobs in $\mathcal{J} \setminus J_\ell$ according to classes and add a setup between consecutive jobs of different classes.
- Add items (jobs, setups) to bins in a next-fit manner (starting with last bin of the previous step).

(4) Add missing setups where necessary.

last bin used by our algorithm, then $\frac{s}{v_{m'}} \leq T$. Therefore, for $\ell < i < m$ we have $L_i \leq T + \frac{s}{v_i} + p_{j_q}$, as we overpack by at most one job and have to add at most one additional setup in Step (4). Thus, we have $L_i \leq T + \frac{s}{v_i} + p_{j_q} \leq T + 2\frac{s}{v_i} \leq 3T$. Also, $L_\ell \leq 2T + \frac{s}{v_\ell} \leq 3T$, which concludes our proof. □

Finally, we want to briefly discuss how to use the approach from the previous section to achieve a rather simple 4-approximation algorithm, improving on the bound given by LPT and Lemma 5.13 and reducing the technical complexity of using a PTAS in Lemma 5.13. We sketch the following result.

Theorem 5.15. *Using* `LP-RelaxedRA` *as described in Figure 5.2 together with Proposition 5.9, we can obtain a* 4*-approximation algorithm.*

Proof (Sketch). Let $J_s \coloneqq \{j \in \mathcal{J} : p_j < s_{k_j}\}$ and $J_\ell \coloneqq \mathcal{J} \setminus J_A$. For each job $j \in J_\ell$ create a new class k_j with setup time $s_{k_j} = p_j$ and one job j_k with processing time 0. Call the set of newly created classes $\mathcal{K}'$. We solve this modified instance by using `LP-RelaxedRA` from Section 5.3 and then construct the set $\tilde{E}$ as before with the properties of Proposition 5.9. Now, for each class $k \in \mathcal{K} \setminus \mathcal{K}'$ let i_k^- be the machine such that $\bar{x}^*_{i_k^- k} > 0$ but $\{i_k^-, k\} \notin \tilde{E}$ (if it exists). Let $i_{k,\iota}^+$, $\iota = 1, \ldots, \iota_k$ be the machines such that $\bar{x}^*_{i_{k,\iota}^+ k} > 0$ and $\{i_{k,\iota}^+, k\} \in \tilde{E}$. In case $\bar{x}^*_{i_k^- k} > \frac{1}{2}$, process the entire class k on machine i_k^-. Otherwise, distribute the amount of k processed on i_k^- proportionally to the machines $i_{k,\iota}^+$ so that $\bar{x}^*_{i_k^- k}$ is set to 0 and $\bar{x}^*_{i_{k,\iota}^+ k}$ is at most doubled. Also, replace each setup for a class $k_j \in \mathcal{K}'$ that is completely assigned

to a machine, i.e., for which $x_{ik_j} = 1$ for $k_j \in \mathcal{K}'$ and some $i \in \mathcal{M}$, by the actual job j and its setup (and recall that $s_{k_j} \geq p_j$). Then, it is easy to verify that each machine has a load of at most $2T$. To finally obtain a feasible schedule, it remains to add at most one setup to each machine and, as before in Section 5.3, to greedily fill the reserved slots by the actual jobs. This increases the load on each machine by at most one setup and at most one job and hence, by an additive of at most $2T$. (Note that both are upper bounded by T due to the respective constraints of `LP-RelaxedRA` and in case of a job j from J_ℓ due to the preliminary step of setting the setup time of the newly created class to the size of j.) This concludes the proof for a 4-approximation. □

CHAPTER 6

Offline Scheduling for Maximum Flow Time on a Machine with Setup Times

Makespan is a popular objective function and quite reasonable from the perspective of the entity processing the jobs, for example, a computing center, particularly if all jobs arrive in a bundle. However, if jobs only become ready for processing over time or if one wants to better measure the experience of users issuing the jobs, the question of how a good schedule looks like requires other objective functions. From this perspective it is rather desired that each individual job is finished as early as possible after it is available for processing. Therefore, a natural objective in such a scenario is the minimization of the maximum flow time as introduced in [BCM98]. It is defined as the maximum time a job spends in the system. That is, the time between the arrival of a job and its completion. Note that maximum flow time is a natural generalization of the makespan objective since if all jobs are released at the same time, the maximum flow time boils down to the makespan objective. It describes the quality of service as, for example, perceived by users and aims at schedules being responsive to *each* job. This is in contrast to the total flow time objective, which leads to a good average performance but also to potentially unfair solutions as individual jobs may suffer starvation.

In this chapter, we study the general problem from Chapter 3 for the maximum flow time objective on a single machine. We consider it as an offline problem and since the problem is known to be NP-hard [DS11], we are interested in approximation algorithms. It is known [MP89] that the problem can be solved optimally in time $O(K^2 \cdot n^{2K})$ and the online algorithm in [DS11] can be applied offline and achieves an approximation factor of 11 in time $O(n^2)$. We improve on this factor by coming up with a new approximation algorithm in Section 6.3, after defining the model and necessary notation in Section 6.1 and deriving some simple insights about optimal solutions in Section 6.2.

6.1 Model & Notation

We consider the problem described by the general model in Chapter 3 for the case where the set $\mathcal{J}$ of n jobs needs to be processed by a *single* machine. In this case, each job $j \in \mathcal{J}$ has one processing time $p_j \in \mathbb{N}_{\geq 0}$ and a release time $r_j \in \mathbb{N}_{\geq 0}$ before which it cannot be processed. Similarly, with each class $k \in \mathcal{K}$ one setup time $s_k \in \mathbb{N}_{\geq 0}$ is associated and, as usual, whenever the machine switches from processing a job of class k' to a job of a class $k \neq k'$, a setup of length s_k needs to take place first. The goal is to compute a non-preemptive schedule, in which each job runs to completion without interruption once it is started, that minimizes the *maximum flow time* $F \coloneqq \max_{j \in \mathcal{J}} F_j$ where F_j is the amount of time job j spends in the system. That is, a job j arriving at r_j, started in a schedule at t_j and completing its processing at $c_j \coloneqq t_j + p_j$ has a flow time $F_j \coloneqq c_j - r_j$. Note that different from the makespan objective, a schedule in this setting will usually have several setups for a class and may have idle time (for example when the next job to be processed not yet arrived). A schedule is implicitly given by a starting time t_j for each $j \in \mathcal{J}$. As the machine can process at most one job at a time, we require that $[t_j, t_j + p_j) \cap [t_{j'}, t_{j'} + p_{j'}) = \emptyset$ for all $j, j' \in \mathcal{J}$ and $j \neq j'$. Also, due to the requirement on performing setups when switching between different classes, we demand that $t_j \geq t_{j'} + p_{j'} + s_{k_j}$ if $k_j \neq k_{j'}$, $t_j \geq t_{j'}$ and no job j'' exists with $t_{j''} \in (t_{j'} + p_{j'}, t_j]$. In the following, we use $F^*(\mathcal{I})$ to denote the maximum flow time of an optimal solution $\textsc{Opt}(\mathcal{I})$ and usually omit $\mathcal{I}$ if it is clear from the context.[1]

6.2 Basic Properties and Observations

For our approach, we partition the time into intervals of length F^* and based thereon the set of jobs into sets $J_i \coloneqq \{j \in \mathcal{J} : (i-1)F^* \leq r_j < iF^*\}$ of jobs released in the i-th interval, for all $i \in \mathbb{N}$. Recall that a batch is a maximal set of jobs of a common class scheduled contiguously and sharing the same setup. We distinguish different kinds of batches depending on the sets J_i of which jobs are processed in a batch. We call a batch an *i-single batch* if it only processes jobs from J_i. We call a batch an *i-pair batch* if it only processes jobs from J_i and J_{i+1}. We call a batch an *i-triple batch* if it only processes jobs from J_i, J_{i+2} and possibly J_{i+1}. We call a batch an *(i, i')-critical batch* if it processes jobs from J_i and from $J_{i'}$ for $i' \geq i + 3$ and no jobs from any $J_{i''}$ for $i'' > i'$. We use the shorter notation *i-critical batch* if we do not need to refer to the value of i'. The term critical is due to the fact that we will build non-critical batches greedily while for critical batches we will need an exhaustive enumeration. We also use the notation *i-* batch* where * is a wildcard for single, pair, triple and critical. By definition we have the following observation.

Observation 6.1. A job $j \in J_i$ can be processed in an i-single batch, $(i-1)$-pair or i-pair batch, $(i-2)$-triple, $(i-1)$-triple or i-triple batch, or an (i', i'')-critical

[1] As in [DS11], we assume that the optimal flow time is at least $F^* \geq \max_{k \in \mathcal{K}} s_k$. This is not stated explicitly but assumed implicitly in [DS11] (cf. proof of Property 6.)

batch if $i' \leq i \leq i''$. These are the only possibilities for $j \in J_i$.

We start with two propositions about triple and critical batches in optimal solutions.

Proposition 6.2. *In an optimal schedule the sum of the number of i-triple and i-critical batches is at most 1 for each i.*

Proof. Recall that an i-triple batch as well as an i-critical batch contains jobs from J_i and $J_{i'}$ for $i' \geq i+2$. Therefore, in an optimal schedule such a batch needs to be started before $(i+1)F^*$ since otherwise the job from J_i would have a flow time larger than F^*. Also, it cannot be finished before $(i+1)F^*$ since it contains a job from $J_{i'}$. Hence, the respective batch needs to be processed at time $(i+1)F^*$ and thus, there can be at most one such batch. □

The next observation is based on the fact that jobs which do not belong to a (critical) batch but which are released during its processing can only be processed before or after it. Intuitively, as critical batches are very long, this implies that no such jobs are released at all except for the border cases where the respective jobs can be processed before and after the critical batch, respectively.

Proposition 6.3. *Consider an optimal schedule with an (i,i')-critical batch. All jobs of J_{i+1} ($J_{i'-1}$) that are not processed in this critical batch are processed before (after) the critical batch. There are no jobs in $J_{i+2} \cup \ldots \cup J_{i'-2}$ that are not processed in this critical batch if $i' \geq i+4$.*

Proof. Note that the i-critical batch needs to be started before $(i+1)F^*$ and cannot be finished before $(i+2)F^*$. Hence, all jobs of J_{i+1} not being part of the critical batch are processed before it as they cannot be processed after it. Similarly, all jobs of $J_{i'-1}$ not being part of the critical batch need to be processed after it since $i'-1 \geq i+2$ and hence they are released after the critical batch is already started. Finally, suppose to the contrary that for some i^* with $i+2 \leq i^* \leq i'-2$ there is a job in J_{i^*} not being part of the critical batch. This job cannot be started before the critical batch so it needs to be started afterwards. However, then it would finish later than $(i'-1)F^* \geq (i^*+1)F^*$, which contradicts that the maximum flow time is F^*. □

6.3 Approximation Algorithm

In this section, we design and analyze our approximation algorithm. To this end, in Section 6.3.1, we develop and formally define a certain class of schedules that are only by a constant factor worse than optimal schedules and have certain nice structural properties. These properties help us to come up with an approximation algorithm in Section 6.3.2. In the following we assume that F^* is known to the algorithm, which is justified by the fact that it can easily be applied within the binary search of the dual approximation framework as introduced in Chapter 2 on the interval $[\max_j p_j, \max_j r_j + \sum_{k=1}^{K} s_k + \sum_{j=1}^{n} p_j] \ni F^*$.

6.3.1 EarlyBatch-Schedules

In the following, we use the term *relaxed schedule* to denote a schedule that is feasible except for the fact that jobs might be started before their release times. Note that a relaxed schedule with maximum flow time F in which no job is started more than Δ before its release time, can easily be transformed into a feasible schedule with maximum flow time at most $F + \Delta$ by delaying the starting times of all jobs by Δ. Next, we are going to define the structure of certain good and nicely structured schedules. Because this structure goes hand in hand with the algorithm that we will later develop in Section 6.3.2 for the actual computation of such schedules, we first describe our strategy in a nutshell: We proceed in iterations such that in iteration i, we build i-single and i-pair batches and either i-triple or one i-critical batch. The $(i+1)$-* batches of the next iteration are then simply appended to the schedule so far. The pair and triple batches are built greedily in the sense that if $J_i \cap C_k \neq \emptyset$, those jobs $J_{i+1} \cap C_k$ and $J_{i+2} \cap C_k$ are processed in the same batch as J_i. This aims at minimizing the number of setups we need to perform while, on the other hand, it might require jobs to be started too early (compared to their release times). In contrast, critical batches are not built greedily but instead we will guess the right ones (or rather their classes) as we will see later on. We formalize these ideas concerning the structure of schedules we will consider as follows.

Definition 6.4 (EarlyBatch-Schedule)**.** A relaxed schedule is called an *EarlyBatch-*Schedule if all jobs of $J_i \cap C_k$ are processed in the same batch, for all i and k, and the following holds inductively for any $i = 1, 2, \ldots$:
If there is no i-critical batch, then

1. For any k, if $J_i \cap C_k \neq \emptyset$ and $J_{i+2} \cap C_k \neq \emptyset$ and the jobs of J_i are not already scheduled, $(J_i \cup J_{i+1} \cup J_{i+2}) \cap C_k$ form an i-triple batch.

2. For any k, if $J_i \cap C_k \neq \emptyset$ and $J_{i+1} \cap C_k \neq \emptyset$ and the jobs of J_i are not already scheduled or part of an i-triple batch, $(J_i \cup J_{i+1}) \cap C_k$ form an i-pair batch.

3. All remaining jobs of J_i are processed in i-single batches.

4. All i-* batches are processed before any $(i+1)$-* batch. i-* batches are processed in an arbitrary fixed order starting with the i-* batch with largest setup time.

If there is an i-critical batch (of a class k), then

1. For any $k' \neq k$, if $J_i \cap C_{k'} \neq \emptyset$ and $J_{i+1} \cap C_{k'} \neq \emptyset$ and the jobs of J_i are not already scheduled, $(J_i \cup J_{i+1}) \cap C_{k'}$ form an i-pair batch.

2. All jobs of J_{i+1} not already processed, are processed in $(i+1)$-single batches.

3. The i-critical batch contains all jobs of class k of $J_i \cup \ldots \cup J_{i'}$, where $i' > i+2$ is the first index such that $J_{i'-1}$ contains jobs of a class $k' \neq k$.

4. All remaining jobs of J_i are processed in i-single batches.

5. The batches are processed in an arbitrary order as long as the i-critical batch is processed last (also after the $(i+1)$-single batches) and of the remaining i-* batches the one with largest setup time is processed first. All $(i'-1)$-* batches are processed after the i-critical batch.

Note that once the classes of all critical batches are fixed, the respective EarlyBatch-Schedule is uniquely determined and can be computed by greedily building single, pair and triple batches according to Definition 6.4. This even holds if one only knows the i'-critical batches for $i' \leq i$ when building the i-* batches.

Lemma 6.5. *There is an EarlyBatch-Schedule with the following properties for each i: All jobs in i-single, i-pair and i-triple batches are finished by $(i+3)F^*$ and are not started before $(i-1)F^*$. All jobs of i-critical batches belonging to $J_{i'}$ are started not before $(i'-1)F^*$ and are finished not after $(i'+1)F^*$. Hence, there is an EarlyBatch-Schedule with flow time at most $7F^*$.*

Proof. We show how we can iteratively construct a schedule with the desired properties. Starting with an optimal solution, we do so by an induction on the modification steps using the following induction hypothesis: Before iteration $i+1$, the schedule S_i constructed so far can be decomposed into S_i^1, S_i^2 and S_i^3 such that each is a feasible relaxed schedule for its respective set of jobs (and consequently, no batch can belong to more than one subschedule) and its concatenation S_i^1, S_i^2 and S_i^3 (in this ordering) equals the original schedule S_i. Furthermore,

1. S_i^1 fulfills the properties of an EarlyBatch-Schedule and the desired bounds as given in the lemma for all $i' \leq i$.

2. In S_i^2 all jobs of $J_{i'}$ are not started earlier than $(i'-1)F^*$ and not finished after $(i+3)F^*$.

3. In S_i^3 all jobs of $J_{i'}$ are not started before $(i'-1)F^*$ and not finished after $(i'+1)F^*$.

Before the first iteration, we choose $S_0^1 = S_0^2 = \emptyset$ and S_0^3 as an optimal solution so that the desired properties hold. Without loss of generality, we assume that the optimal solution fulfills the property that all jobs of a class are processed in non-decreasing order with respect to their release times. (Note that in the following we will skip some iterations as due to Definition 6.4 there are essentially no jobs to consider in iterations $i+1, \ldots, i'-2$ if there is an (i,i')-critical batch.)

Now we suppose the induction hypothesis holds up to $i-1$ and we show it for i. Our starting point is the decomposition of the schedule into S_{i-1}^1, S_{i-1}^2 and S_{i-1}^3 from the induction hypothesis. The idea now is to find an interval $[t_1, t_2]$ in which we reorder the jobs so that afterward we can easily define S_i^1, S_i^2 and S_i^3 using the induction hypothesis together with the newly ordered jobs. For an illustration refer to Figure 6.1. Let S be S_{i-1}^2 if it is not empty and otherwise

S equals S^3_{i-1}. Let t'_1 be the starting time of the first job j_{first} in S. Let t_1 be the starting time of the setup directly before j_{first}. Note that if $j_{\text{first}} \in J_{i'}$ for $i' \geq i+1$ then $t_1 \geq t'_1 - s_{k_{j_{\text{first}}}} \geq iF^* - F^* \geq (i-1)F^*$ since $s_{k_{j_{\text{first}}}} \leq F^*$ and by induction hypothesis. Otherwise, $t_1 \geq t'_1 - s_{\max} \geq (i-1)F^* - s_{\max}$, where $s_{\max} = \max_{k:J_i \cap C_k \cap S \neq \emptyset} s_k$, since $t'_1 \geq (i-1)F^*$ as S only contains jobs from $J_{i''}$ for $i'' \geq i$. Let t_2 be the finishing time of the last job j_{last} that is supposed to belong to S^1_i (according to Definition 6.4), which we are going to construct. We make a case distinction depending on whether we do or do not have an i-critical batch. (Note that this can depend on the optimal solution we started with as well as on previous iterations.)

(*Case: No i-critical batch.*) First, we consider the case that there is no i-critical batch. Note that $t_2 \leq (i+3)F^*$ by induction hypothesis and because $j_{\text{last}} \in J_i \cup J_{i+1} \cup J_{i+2}$. Within interval $[t_1, t_2]$ we reorder the jobs so that the desired properties hold. To this end, we might need to move jobs to other batches (particularly, to establish the desired properties for S^1_i) and need to reorder batches. However, note that we do not need to do more setups than before so that we can establish the desired properties and still all jobs are started and finished within the interval $[t_1, t_2]$. S^1_{i-1} extended by the new i-* batches forms S^1_i and the desired properties hold. S^3_i is defined as follows. Let t' denote the starting time of the first job of S^3_{i-1}. If $\max\{t_2, t'\} = t_2$, S^3_i is given as the subschedule of S^3_{i-1} starting at t_2 extended by the setup directly preceding the first job. Otherwise, S^3_i equals S^3_{i-1}. The jobs in between S^1_i and S^3_i form S^2_i. We can ensure that no job in S^2_i from a set $J_{i'}$ is started before $(i'-1)F^*$ as this holds before and we do not need to start any such job earlier. The upper bound on the finishing times directly follows from the upper bound on t_2 or by induction hypothesis (with respect to S^2_{i-1} if $t' > t_2$), which concludes this case.

(*Case: i-critical batch.*) Next, consider the case that there is an i-critical batch. We build the desired i-single, i-pair and $(i+1)$-single batches and can guarantee the desired bounds on starting and finishing times for these jobs as in the previous case. Note that we do not do additional setups compared to $S^2_{i-1} \cup S^3_{i-1}$ due to Proposition 6.3. Next, if J_{i^*} for $i^* \geq (i+1)F^*$ is the first set containing jobs that are not in the i-critical batch, note that the critical batch cannot contain jobs of J_{i^*+2} due to Proposition 6.3. Note that the i-critical batch belongs to S^3_{i-1} so that its jobs satisfy the bounds on starting and finishing times as desired in the lemma. However, it might be necessary to extend it (by jobs from the respective class and belonging to sets up to J_{i^*+1} to satisfy the definition of EarlyBatch schedules). Note that $t_2 \leq (i^*+2)F^*$ and hence, we get the upper bound on the finishing times as desired. Extending the i-critical batch by the respective jobs and noting that we can skip the next iterations until i^*, we obtain $S^1_{i^*-1}$. $S^3_{i^*-1}$ is the subschedule of S^3_{i-1} starting at t_2 extended by the directly preceding setup. In between $S^1_{i^*-1}$ and $S^3_{i^*-1}$ we have $S^2_{i^*-1}$. Note that it can only contain jobs from $J_{i'}$ for $i' \geq i^*$ by definition of i^*. They are finished by $t_2 \leq (i^*+2)F^*$ and hence, the desired bounds hold, which concludes the proof. □

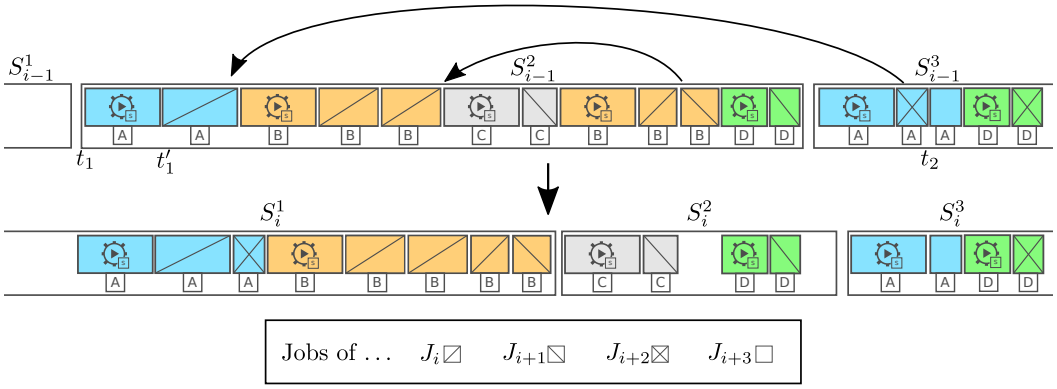

Figure 6.1: Illustration of the induction step from $i-1$ to i. Rectangles represent jobs, gears represent setups and colors (characters $\{A, B, C, D\}$, respectively) represent classes of jobs and setups.

6.3.2 Constructing EarlyBatch-Schedules

Given Definition 6.4 and Lemma 6.5, it would be easy to compute a schedule with flow time at most $7F^*$ if we knew the i-critical batches for all i. Unfortunately, this is not the case though and consequently, our strategy will be to guess each i-critical batch. Let S_{i-1} denote a schedule fulfilling the properties of Definition 6.4 for all $i' \leq i-1$ and suppose we know the class of the i-critical batch. Then we can easily compute subschedule S_i by building the i-single batches, i-pair batches and (in case there is no i-critical batch) the i-triple batches greedily according to the structure of EarlyBatch-Schedules of Definition 6.4. Now we can enumerate all possible solutions satisfying the aforementioned properties: Given set $\mathcal{S}_{i-1}$ of all possible schedules S_{i-1}, we build the set $\mathcal{S}_i$ of all possible schedules S_i by extending each schedule in $\mathcal{S}_{i-1}$ by the respective batches for each possible choice for the class of the i-critical batch. Note that there are $K+1$ choices, namely K classes and the possibility to not have an i-critical batch. Unfortunately, this does not yet give a polynomial time algorithm.

However, we observe that in ech $\mathcal{S}_i$ there might be schedules that we can safely remove before computing $\mathcal{S}_{i+1}$ and thereby we will see that we can make the algorithm run in polynomial time. Of course, whenever we construct an infeasible schedule, i.e., a schedule that does not fulfill the properties of Lemma 6.5, we will remove it. Additionally, we consider the following concept of domination.

Definition 6.6. Let $S, \bar{S}$ in $\mathcal{S}_i$ be two schedules having i-critical batches of the same class. If S is shorter than $\bar{S}$, we say S *dominates* $\bar{S}$.

Due to our definition of EarlyBatch-schedules (particularly the fact that jobs are "pulled" to early batches and in case of an i-critical batch the fact that $(i+1)$-batches are pulled in front of the critical batch), we can show that a dominating and its respective dominated schedule include the exact same set of jobs. Based on that, the next lemma follows.

Lemma 6.7. *If S_i dominates $\bar{S}_i$ and $\bar{S}_i$ can be extended to a schedule with the properties of Lemma 6.5, then S_i can be as well.*

Proof. We only need to show that S_i and $\bar{S}_i$ contain the exact same set of jobs. This directly implies the lemma. Note that all jobs from J_{i^*} for $i^* \leq i$ are contained in both schedules which directly follows from the construction. All jobs from J_{i+1} are part of both schedules as we built the respective $(i+1)$-single batches because we do have an i-critical batch according to Definition 6.6. Furthermore, no job of J_{i+2} that is not in the i-critical batch is part of the schedules as no i-triple batches are built and such jobs cannot belong to any other already built batches. If the last job of the i-critical batch belongs to $J_{i'}$, both schedules do not contain any jobs from J_{i^*} not being in the i-critical batch for $i^* \geq i' - 1$. Furthermore, according to Definition 6.4, there are no jobs in $J_{i+2} \cup \ldots \cup J_{i'-2}$ that are not part of the i-critical batch. □

Due to the previous lemma, we only need to keep one not dominated schedule for each possible choice for the class of the i-critical batch in $\mathcal{S}_i$. Schedules that do not have an i-critical batch are all kept (except those which we can directly exclude because they do not satisfy Lemma 6.5). Therefore, we have $|\mathcal{S}_i| \leq k + |\mathcal{S}_{i'}|$, where $i' < i$ is the largest index such that $J_{i'} \neq \emptyset$. Note that we can easily determine the at most n values for which $J_i \neq \emptyset$ in advance so that we can skip the respective iterations when calculating the sets $\mathcal{S}_i$. Hence, $|\mathcal{S}_i| \leq nk$, for all relevant values of i, and we therefore designed an algorithm with the properties summarized in the final theorem.

Theorem 6.8. *There is an approximation algorithm with approximation factor 7 and a running time polynomial in n and K.*

CHAPTER 7

Online Scheduling for Maximum Flow Time on a Machine with Setup Times

Maximum flow time, as already studied in Chapter 6, is a natural objective function for considering the resulting scheduling problem in an online setting. Recall that in this case, the jobs arrive over time at their respective release times and they are not known to a scheduler in advance. Additionally, in the strong *non-clairvoyant* variant of the online notion as introduced in [MPT94], the processing time of a job only becomes known upon completion instead of being available to a scheduler upon the arrival of a job. Such an assumption is reasonable, for example, in settings in which user interaction is present or processing times may depend on further inputs not known upon the arrival of a job.

In this chapter, we study the potential of conceptually simple ("greedy-like"), non-clairvoyant online algorithms in terms of their (smoothed) competitiveness. The formal definitions of the notions "greedy-like" and smoothed competitiveness are given in Section 7.1 together with a description of the formal model. In Section 7.2 we analyze the competitiveness of "greedy-like" algorithms and show matching upper and lower bounds of $\Theta(\sqrt{n})$, where the bound is achieved by a simple modification of the *First In First Out* (FIFO) strategy. Interestingly, the competitiveness linearly improves with an increasing optimal flow time (assuming n, the setup time and the largest processing time are fixed) and thus, is much better in case the optimal flow time is large. Also, in case of only two classes the competitiveness improves to a constant. Our main result is an analysis of the smoothed competitiveness of this algorithm in Section 7.4, which is shown to be $O(\sigma^{-2}\log^2 n)$ where σ denotes the standard deviation of the underlying smoothing distribution. It shows worst-case instances to be fragile against random noise and that, except on some pathological instances, the algorithm achieves a much better performance than suggested by the worst-case bound on the competitiveness. The bound even holds with high

probability when noise is present and thus, one can expect that a bad performance is rather unlikely to occur in practice.

7.1 Model, Notation & Notions

We consider the general problem from Chapter 3 instantiated as in Chapter 6. That is, n jobs, partitioned into K classes, are to be scheduled on a single machine. Each job j has a processing time $p_j \in \mathbb{R}_{\geq 1}$, a release time $r_j \in \mathbb{R}_{\geq 0}$ and a parameter k_j defining the class it belongs to.[1] The machine can process at most one job at a time. Whenever it switches from processing jobs of one class to a different one and before the first job is processed, a setup taking a constant time $s \in \mathbb{R}_{\geq 0}$ needs to take place during which the machine cannot be used for processing. The goal is to compute a non-preemptive schedule that minimizes the maximum flow time F, where (recalling from Chapter 6) $F \coloneqq \max_{1 \leq j \leq n} F_j$ and $F_j \coloneqq c_j - r_j$.

Recall that a *batch* is a sequence of jobs, all of a common class k, that are processed in a contiguous interval without any intermediate setup. For a batch B, we use $\tau(B)$ to denote the common class of B's jobs and $w(B) \coloneqq \sum_{j \in B} p_j$ to denote its workload. We refer to setup times and idle times as *overhead* and overhead is associated to a job j if it directly precedes j in the schedule. For an interval $I = [a, b]$ we also use $l(I) \coloneqq a$ and $r(I) \coloneqq b$ and $w(I) \coloneqq \sum_{j : r_j \in I} p_j$ to denote the workload released in interval I.

Non-Clairvoyant Greedy-like Online Algorithms We consider our problem in an online setting where jobs arrive over time at their release times and are not known to the scheduler in advance. Upon arrival the scheduler gets to know a job together with its class but does not learn about its processing time, which is only known upon its completion (*non-clairvoyance*) [MPT94]. We are interested in the potential of conceptually simple algorithms. Although such algorithms might not lead to the best achievable performance, they often have strong points such as being very fast, working quite well on realistic data or being easy to implement and modify as discussed in [BPS17]. A prime example of conceptually simple scheduling algorithms are *priority algorithms* that, intuitively speaking, assign each job a single parameter (called its priority) and then base each scheduling decision solely on the jobs' priorities as described in [Gup+12]. EarliestDeadlineFirst, ShortestProcessingTimeFirst, HighestDensityFirst or FIFO are just a few of numerous well-known and used such algorithms. Priority algorithms have been formally defined to describe the concept of *greedy-like* (or myopic) algorithms for classical combinatorial offline problems by Borodin et al. in [BNR03]. We adopt this concept and for our online problem

[1] Note that allowing processing times to be real-valued is more general than the assumption of having integral processing times in Chapter 6. Here we use this model so that the small random perturbations that we will perform do not lead to instances not adhering to our model. The results of Chapter 6 also carry over to real-valued processing times except that we lose an ε in the approximation factor due to imprecision when guessing the optimal flow time within the dual approximation framework.

we define greedy-like algorithms to work as follows: When a job completes (and when the first job arrives), the algorithm determines a total ordering of *all possible* jobs without looking at the actual instance. It then chooses (among already arrived yet unscheduled jobs) the next job to be scheduled by looking at the instance and selecting the job coming first according to this ordering.

Quality Measure To analyze the quality of online algorithms, we facilitate competitive analysis as described in Chapter 2. Recall that it compares solutions of the online algorithm to solutions of an optimal offline algorithm which knows the complete instance in advance. Precisely, an algorithm ALG is called *c-competitive* if, on any instance $\mathcal{I}$, $F(\mathcal{I}) \leq c \cdot F^*(\mathcal{I})$, where $F(\mathcal{I})$ and $F^*(\mathcal{I})$ denote the flow time of ALG and an optimal (clairvoyant) offline solution on instance $\mathcal{I}$, respectively.

Although competitive analysis is the standard measure for analyzing online algorithms, it is often criticized to be overly pessimistic. That is, a single or a few pathological and very rarely occurring instances can significantly degrade the quality with respect to this measure. To overcome this, various alternative measures have been proposed in the past (see, e.g., [HV12; Lóp16; KP00]). One approach introduced in [Bec+06] is *smoothed competitiveness*. Here the idea is to slightly perturb instances dictated by an adversary by some random noise and then analyze the expected competitiveness, where expectation is taken with respect to the random perturbation. Formally, if input instance $\mathcal{I}$ is smoothed according to some smoothing (probability) distribution f and if we use $N(\mathcal{I})$ to denote the instances that can be obtained by adding noise according to f to instance $\mathcal{I}$, the smoothed competitiveness c_{smooth} is defined as $c_{\text{smooth}} := \sup_{\mathcal{I}} \mathbb{E}_{\hat{\mathcal{I}} \xleftarrow{f} N(\mathcal{I})} \left[\frac{F(\hat{\mathcal{I}})}{F^*(\hat{\mathcal{I}})} \right]$. Note that the smoothed competitiveness considers the expectation of the ratio and therefore still compares offline and online solutions on a per-instance basis. Compared to the option of considering the ratio of expectations, this has, for example, the advantage that by using Markov's inequality one can derive bounds on the distribution of the competitive ratio. (Actually, we even get a stronger result and show that our bound on the ratio not only holds on expectation but even with high probability.) For a further discussion on the differences between these definitions also see [SSS06].

We will smooth instances by randomly perturbing processing times. We assume the adversary to be *oblivious* with respect to perturbations. That is, the adversary constructs the instance based on the knowledge of the algorithm and f (so that $\mathcal{I}$ is defined at the beginning and is not a random variable).

7.2 A Non-Clairvoyant Online Algorithm

In this section, we present a simple greedy-like algorithm and analyze its competitiveness. When designing an algorithm for the considered problem, two aspects should be taken into account: First, it should try to keep the number of setups small by processing jobs of the same class together in one batch. However, second, this batching should not be too strict as otherwise other jobs might be delayed too

much. Therefore, the idea of the algorithm Balance, as presented in Algorithm 4, is to find a tradeoff between preferring jobs with early release times and jobs that are of the class the machine is currently configured for. This is achieved by the following idea: Whenever the machine is about to idle at some time t, Balance checks whether there is a job j available that is of the same class k_j as the machine is currently configured for, denoted by $active(t)$. If this is the case and if there is no job j' with a "much smaller" release time than j, job j is assigned to the machine. The decision whether a release time is "much smaller" is taken based on a parameter λ, called *balance parameter*. This balance parameter is grown over time based on the maximum flow time encountered so far and, at any time, is of the form α^q, for some $q \in \mathbb{N}$ which is increased over time and some constant α determined later.[2] Note that Balance is a greedy-like algorithm by using the adjusted release times for determining the ordering of jobs.

Algorithm 4 Description of the algorithm Balance.

(1) Let $\lambda = \alpha$. ▷ for some constant α

(2) If the machine idles at time t,
process available job with smallest **adjusted release time** $\bar{r}_j(t)$

$$\bar{r}_j(t) \coloneqq \begin{cases} r_j & \text{if } k_j = \text{active}(t) \\ r_j + \lambda & \text{else} \end{cases}$$

after doing a setup if necessary.
To break a tie, prefer job j with $k_j = \text{active}(t)$.

(3) As soon as a job j completes with $F_j \geq \alpha\lambda$, set $\lambda \coloneqq \alpha\lambda$.

7.2.1 Basic Properties of Balance

The following two properties follow from the definition of Balance and relate the release times and flow times of consecutive jobs, respectively. For a job j, let $\lambda(j)$ denote the value of λ when j was scheduled.

Proposition 7.1. *Consider two jobs j_1 and j_2. If $k_{j_1} = k_{j_2}$, both jobs are processed according to FIFO. Otherwise, if j_2 is processed after j_1 in a schedule of* Balance, $r_{j_2} \geq r_{j_1} - \lambda(j_1)$.

Proof. The first statement directly follows from the definition of the algorithm. Consider the statement for two jobs j_1 and j_2 with $k_{j_1} \neq k_{j_2}$. Let t be the point in

[2] A variant of this algorithm with a fixed λ and hence without Step (3) has been previously mentioned in [DS11] as an algorithm with $\Omega(n)$-competitiveness for the clairvoyant variant of our problem.

time at which j_1 is assigned to the machine. If $\text{active}(t) \neq k_{j_1}$ and $\text{active}(t) \neq k_{j_2}$, it follows $r_{j_2} \geq r_{j_1}$. If $\text{active}(t) = k_{j_1}$, then because j_1 is preferred over j_2, we have $r_{j_1} = \bar{r}_{j_1}(t) \leq \bar{r}_{j_2}(t) = r_{j_2} + \lambda(j_1)$. Finally, if $\text{active}(t) = k_{j_2}$ we know by the fact that j_1 is preferred that $r_{j_1} + \lambda(j_1) = \bar{r}_{j_1}(t) < \bar{r}_{j_2}(t) = r_{j_2}$, which proves the proposition. □

Proposition 7.2. *Consider two jobs j_1 and j_2. If j_1 is processed before j_2 and no job is processed in between, then $F_{j_2} \leq F_{j_1} + p_{j_2} + s + \lambda(j_1)$.*

Proof. First note that Balance does not idle deliberately. Hence, if there is idle time between the processing of job j_1 and j_2, then $t_{j_2} \leq r_{j_2} + s$ holds. Thus, we have $F_{j_2} \leq s + p_{j_2}$ proving the claim.

If there is no idle time, by definition j_1 is finished by time $r_{j_1} + F_{j_1}$. Since j_2 is processed directly afterward, it is finished not later than $r_{j_1} + F_{j_1} + s + p_{j_2}$. By Proposition 7.1 this is upper bounded by $r_{j_2} + \lambda(j_1) + F_{j_1} + s + p_{j_2}$, which proves the desired bound. □

7.3 Competitive Analysis

To analyze the competitiveness of Balance, we carefully define specific subschedules of a given schedule S of Balance, which we will use throughout our analysis of the (smoothed) competitiveness. Given $\alpha^q \geq F^*$, $q \in \mathbb{N}$, let S_{α^q} be the subschedule of S that starts with the first job j with $\lambda(j) = \alpha^q$ and ends with the last job j' with $\lambda(j') = \alpha^q$. That is, S_{α^q} is the maximal subschedule of jobs scheduled with α^q as balance parameter. For a fixed $\delta < \alpha$, let $S^\delta_{\alpha^q}$ be the suffix of S_{α^q} such that the first job in $S^\delta_{\alpha^q}$ is the last one in S_{α^q} with the following properties: (1) It has a flow time of at most $(\alpha - \delta)\alpha^q$, and (2) it starts a batch. (We will prove in Lemma 7.3 that $S^\delta_{\alpha^q}$ always exists.) Without loss of generality, let $j_1, \ldots, j_m$ be the jobs in $S^\delta_{\alpha^q}$ such that they are sorted by their starting times, $t_1 < t_2 < \ldots < t_m$. Let $B_1, \ldots, B_\ell$ be the batches in $S^\delta_{\alpha^q}$. The main idea of Lemma 7.3 is to show that, in case a flow time of $F > \alpha^{q+1}$ is ever reached, the interval $[r_{j_1}, r_{j_m}]$ is in a sense dense: Workload plus setup times in $S^\delta_{\alpha^q}$ is at least by $\delta\alpha^q$ larger than the length of this interval. Intuitively, this holds as otherwise the difference in the flow times F_{j_1} and F_{j_m} could not be as high as $\delta\alpha^q$, which, however, needs to hold by the definition of $S^\delta_{\alpha^q}$. Additionally, the flow time of all jobs is shown to be lower bounded by $3\alpha^q$. Roughly speaking, this holds due to the following observation: If a fixed job has a flow time below $3\alpha^q$, then the job starting the next batch can, on the one hand, not have a much smaller release time (by definition of the algorithm). On the other hand, it will therefore not be started much later, leading to the fact that the flow time cannot be too much larger than $3\alpha^q$ (and in particular, is below $(\alpha - \delta)\alpha^q$ for sufficiently small δ, which contradicts the definition of $S^\delta_{\alpha^q}$).

Lemma 7.3. *Let $\alpha^q \geq F^*$ and $\delta \leq \alpha - 10$. Then $S^\delta_{\alpha^q}$ always exists and all jobs in $S^\delta_{\alpha^q}$ have a flow time of at least $3\alpha^q$. Also, if $F > \alpha^{q+1}$, it holds $\sum_{i=1}^{\ell} w(B_i) + r_{j_1} - r_{j_m} \geq \delta\alpha^q - (\ell - 1)s$.*

Proof. We first prove that a job with the two properties starting off $S^{\delta}_{\alpha^q}$ exists. Let $\tilde{j}_1, \ldots$ be the jobs in S_{α^q}. Consider the last job $\tilde{j}_0$ processed directly before $\tilde{j}_1$. By applying Proposition 7.2 twice, we have $F_{\tilde{j}_1} \leq F_{\tilde{j}_0} + p_{\tilde{j}_1} + s + \alpha^{q-1} \leq \alpha^q + p_{\tilde{j}_0} + s + \alpha^{q-1} + p_{\tilde{j}_1} + s + \alpha^{q-1} < 6\alpha^q$. Among jobs in S_{α^q} that have a different class than $\tilde{j}_1$, consider the job $\tilde{j}_i$ with the lowest starting time. We show that it is a candidate for starting $S^{\delta}_{\alpha^q}$, implying that $S^{\delta}_{\alpha^q}$ exists. Property (2) directly follows by construction. For the flow time of $\tilde{j}_i$, we know that only jobs of the same class as $\tilde{j}_1$ are scheduled between $\tilde{j}_1$ and $\tilde{j}_i$. This implies that jobs $\tilde{j}_2, \ldots, \tilde{j}_{i-1}$ are released in the interval $[r_{\tilde{j}_1}, r_{\tilde{j}_i} + \alpha^q]$. The interval can contain a workload of at most $r_{\tilde{j}_i} + \alpha^q - r_{\tilde{j}_1} + F^*$ (see also Proposition 7.5), hence the flow time of job $\tilde{j}_i$ is at most $F_{j_1} = F_{\tilde{j}_i} \leq (r_{\tilde{j}_1} + F_{\tilde{j}_1} + (r_{\tilde{j}_i} + \alpha^q - r_{\tilde{j}_1} + F^*) + s + p_{j_i}) - r_{\tilde{j}_i} \leq 9\alpha^q + p_{\tilde{j}_i} = 9\alpha^q + p_{j_1} \leq (\alpha - \delta)\alpha^q$. Property (1) and the existence of $S^{\delta}_{\alpha^q}$ follow. Since $t_{j_1} = c_{j_1} - p_{j_1}$ and $F_{j_1} = c_{j_1} - r_{j_1}$, we also have $t_{j_1} \leq r_{j_1} + 9\alpha^q \leq r_{j_1} + (\alpha - \delta)\alpha^q$ (*).

We now show that during $S^{\delta}_{\alpha^q}$, the machine does not idle and each job in $S^{\delta}_{\alpha^q}$ has a flow time of at least $3\alpha^q$. Assume this is not the case. Denote by t the last time in $S^{\delta}_{\alpha^q}$ where either an idle period ends or a job with a flow time of less than $3\alpha^q$ completes. We denote the jobs scheduled after t by $\hat{j}_1, \ldots$ and the first job of the first batch started at or after t by $\hat{j}_i$. Similar to above, all jobs $\hat{j}_1, \ldots, \hat{j}_i$ are released in the interval $[t - 3\alpha^q, r_{\hat{j}_i} + \alpha^q]$. The overall workload of these jobs is at most $r_{\hat{j}_i} + 4\alpha^q - t + F^* \leq r_{\hat{j}_i} + 5\alpha^q - t$. Job $\hat{j}_i$ is thus finished by $t + (r_{\hat{j}_i} + 5\alpha^q - t) + s \leq r_{\hat{j}_i} + 6\alpha^q$. This is a contradiction to $F_{\hat{j}_i} > (\alpha - \delta)\alpha^q$, which is supposed to hold by construction of $S^{\delta}_{\alpha^q}$.

Finally, since there are no idle times and by (*), for the last job j_m of $S^{\delta}_{\alpha^q}$ we have $F_{j_m} \leq r_{j_1} + (\alpha - \delta)\alpha^q + \sum_{i=1}^{\ell} w(B_i) + (\ell - 1)s - r_{j_m}$. By the assumption that $F > \alpha^{q+1}$ and the definition of j_m to be the first job with flow time at least α^{q+1}, we obtain the desired result. □

We will also make use of Corollary 7.4, which follows from the proof of Lemma 7.3.

Corollary 7.4. *The statement of Lemma 7.3 also holds if $S^{\delta}_{\alpha^q}$ is replaced by $S^{\delta}_{\alpha^q}(j)$ for any job $j \in S^{\delta}_{\alpha^q}$ with $F_j \leq (\alpha - \delta)\alpha^q$, where $S^{\delta}_{\alpha^q}(j)$ is the suffix of $S^{\delta}_{\alpha^q}$ starting with job j.*

Next we give simple lower bounds for the optimal flow time F^*. Besides the direct lower bound $F^* \geq \max\{s, p_{max}\}$, where $p_{max} \coloneqq \max_{1 \leq j \leq n} p_j$, we can also prove the bound given in Proposition 7.5. For a given interval I, let $\text{overhead}_{\text{OPT}}(I)$ be the overhead in OPT between the jobs j_1 and j_2 released in I and being processed first and last in OPT, respectively. Precisely, $j_1 \coloneqq \text{argmin}_{j:r_j \in I} t_j$ and $j_2 \coloneqq \text{argmax}_{j:r_j \in I} t_j$ for t_j being the starting time of job j in OPT.

Proposition 7.5. *As lower bounds for F^* we have $F^* \geq \max\{s, p_{max}\}$ as well as $F^* \geq \max_I\{w(I) + \text{overhead}_{\text{OPT}}(I) - |I|\}$.*

Proof. We have $F^* \geq c_{j_2} - r_{j_2}$. On the other hand, $c_{j_2} \geq r_{j_1} + \text{overhead}_{\text{OPT}}(I) + w(I)$. Thus, $F^* \geq \text{overhead}_{\text{OPT}}(I) + w(I) + l(I) - r(I) = w(I) + \text{overhead}_{\text{OPT}}(I) - |I|$. □

Combining Lemma 7.3 and Proposition 7.5, we easily obtain that the competitiveness can essentially be bounded by the difference in the number of setups Opt and Balance perform on those jobs which are part of $S^{\delta}_{\alpha^q}$. Let $I(S^{\delta}_{\alpha^q})$ be the (smallest) interval in which all jobs belonging to $S^{\delta}_{\alpha^q}$ are released, $I(S^{\delta}_{\alpha^q}) \coloneqq [\min_j\{r_j : j \in S^{\delta}_{\alpha^q}\}, \max_j\{r_j : j \in S^{\delta}_{\alpha^q}\}]$. We have the following bound.

Lemma 7.6. *Let $\alpha^{q+1} \leq F < \alpha^{q+2}$ and $3 \leq \delta \leq \alpha - 10$ and $\alpha^q \geq F^*$. It holds $F \leq \alpha^2(\delta-2)^{-1}(F^* + \mathit{overhead}_{\text{Balance}}(S^{\delta}_{\alpha^q}) - \mathit{overhead}_{\text{Opt}}(I(S^{\delta}_{\alpha^q})))$.*

Proof. Suppose to the contrary that $\alpha^q > (\delta-2)^{-1}(F^* + \text{overhead}_{\text{Balance}}(S^{\delta}_{\alpha^q}) - \text{overhead}_{\text{Opt}}(I(S^{\delta}_{\alpha^q})))$. By Proposition 7.1 we have $I(S^{\delta}_{\alpha^q}) \subseteq [r_{j_1} - \alpha^q, r_{j_m} + \alpha^q]$ and using Proposition 7.5 we obtain a contradiction as

$$\begin{aligned} F^* &\geq w(I(S^{\delta}_{\alpha^q})) + \text{overhead}_{\text{Opt}}(I(S^{\delta}_{\alpha^q})) + r_{j_1} - r_{j_m} - 2\alpha^q \\ &\geq \sum_{i=1}^{\ell} w(B_i) + \text{overhead}_{\text{Opt}}(I(S^{\delta}_{\alpha^q})) + r_{j_1} - r_{j_m} - 2\alpha^q \\ &\overset{(\textit{Lemma 7.3})}{\geq} \delta\alpha^q - (\ell-1)s + \text{overhead}_{\text{Opt}}(I(S^{\delta}_{\alpha^q})) - 2\alpha^q > F^*, \end{aligned}$$

where the last inequality follows from our assumption. □

Throughout the rest of the chapter, we assume that $\delta = 3$ and $\alpha = 13$ fulfilling the properties of Lemmas 7.3 and 7.6. Our goal now is to bound the competitiveness by upper bounding the difference of the overhead of Opt and Balance in $S^{\delta}_{\alpha^q}$ for some $\alpha^q = \Omega(\sqrt{n \cdot s \cdot p_{max}})$. In Lemma 7.8 we will see that to obtain a difference of $i \cdot s$, a workload of $\Omega(i \cdot \alpha^q)$ is required. Using this, we can then upper bound the competitiveness based on the overall workload of $O(n \cdot p_{max})$ available in a given instance in Theorem 7.9. Before we can prove Lemma 7.8 we need the following insight. Given $S^{\delta}_{\alpha^q}$ for some $q \in \mathbb{N}_0$ such that $\alpha^q \geq F^*$. Let $j_{k,i}$ be the first job of the i-th batch of some fixed class k in $S^{\delta}_{\alpha^q}$. We show that the release times of jobs $j_{k,i}$ and $j_{k,i+1}$ differ by at least α^q. Intuitively, this holds due to the definition of the balance parameter and the fact that in $S^{\delta}_{\alpha^q}$ all jobs starting a batch have a flow time of at least $3\alpha^q$. If $j_{k,i}$ and $j_{k,i+1}$ had release times differing by less than α^q, this would then lead to these jobs being processed in the same batch.

Lemma 7.7. *Given $S^{\delta}_{\alpha^q}$, it holds $r_{j_{k,i}} > r_{j_{k,i-1}} + \alpha^q$, for all $i \geq 2$ and all k.*

Proof. Consider a fixed job $j_{k,i}$ and suppose to the contrary that $r_{j_{k,i}} \leq r_{j_{k,i-1}} + \alpha^q$ holds. As each job in $S^{\delta}_{\alpha^q}$ that is the first of a batch (except the very first such job) has a flow time of at least $3\alpha^q$, job $j_{k,i-1}$ is not started before $r_{j_{k,i}}$ (otherwise it would be finished not later than $r_{j_{k,i-1}} + \alpha^q + p_{j_{k,i}} \leq r_{j_{k,i-1}} + \alpha^q + F^* \leq r_{j_{k,i-1}} + 2\alpha^q$ with flow time smaller $3\alpha^q$). Also, because $j_{k,i-1}$ is the first job of a batch, all jobs j processed later fulfill $r_j \geq r_{j_{k,i-1}}$. But then at the time t at which the $(i-1)$-th batch is finished, $\bar{r}_{j_{k,i}}(t) \leq r_{j_{k,i-1}} + \alpha^q \leq r_j + \alpha^q = \bar{r}_j(t)$ and hence, $j_{k,i}$ would be preferred over all such jobs j and thus would belong to the same batch as $j_{k,i-1}$. This contradicts the definition of $j_{k,i}$, proving the lemma. □

In the next lemma we essentially show that jobs together forming a single batch in Opt can be processed in at most two batches in the schedule of Balance and that a situation where this actually happens requires certain conditions to be fulfilled.

Lemma 7.8. *Let B be a batch in* Opt. *If all jobs from B are part of $S^{\delta}_{\alpha^q}$, an overhead of at most $2s$ is associated to them in the schedule of* Balance.

Also, if the overhead associated to B in Opt *is smaller than $2s$ and in the schedule of* Balance *it is $2s$, it needs to hold*

1. $w(B) \geq \alpha^q - F^* - s =: \bar{w}$ *and*
2. *there is an interval of length α^q such that jobs of size at least $\bar{w}$ that belong to B are released in it.*

Proof. Assume to the contrary that Balance processes the jobs of B in three batches with $j_1, j_2, j_3 \in B$ being the jobs starting the first, the second and the third batch, respectively. Then there need to be two jobs i_1 and i_2 that are processed between the first and second, and second and third such batch, respectively. Since j_2 is preferred over i_2 and by Lemma 7.7, we have $r_{i_2} \geq r_{j_2} \geq r_{j_1} + \alpha^q$. Also, since i_2 is preferred over j_3 and by Lemma 7.3, we have $r_{i_2} + \alpha^q \leq r_{j_3}$. Hence, Opt can neither process i_2 before nor after B (since then either j_1 or i_2 would have a flow time larger than F^*), which is a contradiction to the fact that B is a batch in Opt.

If Balance processes the jobs of B in two batches, let $j_1, j_2 \in B$ be the jobs starting the first batch and the second batch, respectively. We start with the case that $w(B) < \bar{w}$ and show a contradiction. We know that $r_{j_2} > r_{j_1} + \alpha^q$. Consider an optimal schedule. As j_1 cannot be started after $r_{j_1} + F^*$ and because Opt processes j_1 and j_2 in the same batch B, the processing of B needs to cover the interval $[r_{j_1} + F^*, r_{j_2}] \supseteq [r_{j_1} + F^*, r_{j_1} + \alpha^q]$. As $w(B) < \alpha^q - F^* - s$ this implies an additional overhead of at least s associated to B, contradicting our assumption.

Therefore, assume that $w(B) \geq \bar{w}$ but there is no interval of length α^q with jobs of B of size at least $\bar{w}$. We know that j_1 needs to be started not later than $r_{j_1} + F^*$. Also, the workload of jobs of B released until $r_{j_1} + \alpha^q$ is below $\bar{w}$. Hence, there needs to be a job in B released not before $r_{j_1} + \alpha^q$. This implies that the processing of B needs to cover the entire interval $[r_{j_1} + F^*, r_{j_1} + \alpha^q]$. However, this implies an additional overhead associated to B of at least s, contradicting our assumption. □

We are now ready to bound the competitiveness of Balance. Intuitively, it is based on the fact that, as soon as Balance's flow time is sufficiently large, the balance parameter is large as well. Then, Balance will not perform too many setups compared to the optimal solution as it will batch jobs that are released far apart from each other and which the optimal solution cannot batch. On the other hand, if Balance actually does more setups than an optimal solution, this requires a certain number of jobs according to the previous lemma. Therefore, with Lemma 7.6 we can bound the competitiveness based on the number of jobs available.

Theorem 7.9. Balance *is $O(\sqrt{n})$-competitive. Additionally, it holds $F = O(F^* + \sqrt{np_{max}s})$.*

Proof. If $F \leq \sqrt{n \cdot s \cdot p_{max}}$ holds, we are done as $F^* \geq \sqrt{s \cdot p_{max}}$ by Proposition 7.5.

Hence, consider the case where $F > \sqrt{n \cdot s \cdot p_{max}}$ and assume $\alpha^{q+1} \leq F < \alpha^{q+2}$. Also we can assume $F^* \leq \frac{F}{\alpha^3} < \alpha^{q-1}$ as otherwise we have a constant competitiveness. Consider $S^{\delta}_{\alpha^q}$. We call a batch B of Opt *short* if $w(B) < \bar{w}$ and *long* otherwise. According to Lemma 7.8, we know that the overhead associated to jobs belonging to short batches is not larger in a schedule of Balance than in Opt. On the other hand, overhead associated to jobs belonging to long batches can be at most by s larger in Balance than in Opt. However, as a long batch requires a workload of $\bar{w} = \alpha^q - F^* - s \geq \alpha^q - 2F^* \geq \alpha^q - 2\alpha^{q-1} \geq \frac{\sqrt{n \cdot s \cdot p_{max}}}{2\alpha^2}$, there can be at most $O(\sqrt{n} \cdot \sqrt{\frac{p_{max}}{s}})$ many long batches as n jobs can have a workload of at most $n \cdot p_{max}$. Hence, by Lemma 7.6 we obtain the desired result. □

Note that by the previous theorem, the competitiveness is bounded by a function that is linearly decreasing in F^* and increasing in $\sqrt{n \cdot s \cdot p_{max}}$.

For the case $K = 2$ we can even strengthen the statement of Lemma 7.7. Given $S^{\delta}_{\alpha^q}$, let job j_i be the first job of the i-th batch in $S^{\delta}_{\alpha^q}$ and note that $k_{j_i} = k_{j_{i+2}}$ as the batches form an alternating sequence of the two classes. We have the following lemma.

Lemma 7.10. *Given $S^{\delta}_{\alpha^q}$, if $K = 2$ then it holds $r_{j_i} > r_{j_{i-1}} + \alpha^q$, for all $i \geq 3$.*

Proof. Consider a fixed job j_i with $i \geq 3$ and suppose to the contrary that $r_{j_i} \leq r_{j_{i-1}} + \alpha^q$ holds. By definition of $S^{\delta}_{\alpha^q}$, job j_{i-1} is not started before r_{j_i} and all jobs processed later have a release time not smaller than $r_{j_{i-1}}$. Hence, by the definition of Balance, j_i would belong to the same batch as j_{i-2}, which is a contradiction. □

Based on this fact, we can show that Opt can essentially not process any jobs that belong to different batches in $S^{\delta}_{\alpha^q}$ in one batch. Hence, Opt performs roughly the same number of setups as Balance does and we have the following theorem by Lemma 7.6.

Theorem 7.11. *If $K = 2$, then* Balance *is $O(1)$-competitive.*

Proof. Assume $F > 2F^*$ as otherwise we are done. Consider $S^{\delta}_{\alpha^q}$ such that $\alpha^q < F \leq \alpha^{q+1}$. By Lemma 7.10, we know that $r_{j_{i+1}} > r_{j_i} + 2F^*$ for all $i \in [2, \ell]$. Now suppose to the contrary that the optimal solution processes two jobs j_i and j_{i+2} in the same batch. As j_i cannot be completed later than $r_{j_i} + F^*$ and job j_{i+2} is not released before $r_{j_{i+2}}$, this batch needs to cover the interval $[r_{j_i} + F^*, r_{j_{i+2}}]$. However, job j_{i+1} needs to be started during the interval $[r_{j_{i+1}}, r_{j_{i+1}} + F^*] \subseteq [r_{j_i} + F^*, r_{j_{i+2}}]$, which is a contradiction.

Hence, the optimal solution cannot process any two jobs j_i and j_{i+2}, for all $i \geq 2$, in the same batch. By Lemma 7.6 we obtain a competitiveness of $O(1)$. □

To conclude this section, we show that the bound of $O(\sqrt{n})$ from Theorem 7.9 for the competitiveness of Balance is tight and that a lower bound of $\Omega(\sqrt{n})$ holds for any greedy-like algorithm as defined in Section 7.1. This also implies that the $\Omega(\sqrt{n})$

bound holds for BALANCE independent of how λ is chosen or increased (and even if done at random). The construction in the proof of Theorem 7.12 is a generalization of a worst-case instance given in [DS11].

Theorem 7.12. *Any greedy-like algorithm is $\Omega(\sqrt{n})$-competitive.*

Proof. The adversary will be defined such that the optimum flow time is $O(1)$ while a fixed greedy-like algorithm A has a flow time of $\Omega(\sqrt{n})$. We define the adversary by specifying the instance in phases. Let the setup time be $s = 1$.

- During the i-th phase, $\sqrt{n}$ unit size jobs of two classes $k_{i_1} \neq k_{i_2}$ that did not occur in any previous phase are released in $\sqrt{n}$ consecutive (discrete) time steps.

- The first job of phase i is released at time $(i-1)(\sqrt{n}+2)$. (Hence, $\sqrt{n}$ jobs are released in $\sqrt{n}$ time steps, then two time steps no job is released. Afterward this pattern is repeated.)

- The first job released in phase i is of class k_{i_1} and the second one of class k_{i_2}. If A prefers the job of class k_{i_1}, let all remaining jobs of phase i be of class k_{i_1}. If A prefers the job of class k_{i_2}, let all remaining jobs of phase i be of class k_{i_2}.

We analyze the flow time of OPT and algorithm A. For each phase, the optimal solution can first process the job belonging to the class of which only one job is released and afterward all remaining jobs released during the phase. Hence, OPT can always start the setup for phase i before or at time $(i-1)(\sqrt{n}+3)$ and has processed all jobs of phase $i-1$ at that point in time, because it executes $\sqrt{n}$ unit size jobs and needs two setups per phase. This gives a maximum flow time of at most 5 (tight if A prefers the job of class k_{i_1}; then OPT prioritizes the job of class k_{i_2} and the first job of class k_{i_1} remains in the system for 5 time step due to 3 time steps until the job of class k_{i_2} is finished and two additional time steps for the execution of the job itself).

For A we first make two observations. (1) We can assume that for some phase i, A neither processes the job released first nor the job released second after the job released last as otherwise $A = \Omega(\sqrt{n})$ holds. (2) We can assume that A processes all jobs of phase i before any job of phase $i+1$ because no two jobs of different phases can be processed in the same batch and hence, processing a job of phase i later than a job of phase $i+1$ cannot be advantageous. By these two observations, the algorithm A has to do three setups for each phase by the construction of the instance. Thus, it finishes the last job of phase i not before $i(\sqrt{n}+3)$. As the adversary can construct $\sqrt{n}$ phases, the last job is finished at $\sqrt{n}(\sqrt{n}+3)$ and it is released not later than $(\sqrt{n}-1)(\sqrt{n}+2)+\sqrt{n} = \sqrt{n}(\sqrt{n}+2)-2$. Hence, the flow time of A is at least $\sqrt{n}(\sqrt{n}+3) - (\sqrt{n}(\sqrt{n}+2)-2) = \sqrt{n}+2 = \Omega(\sqrt{n})$. □

7.4 Smoothed Competitive Analysis

In this section, we analyze the smoothed competitiveness of Balance. We consider the following *multiplicative smoothing model* from [Bec+06]. Let p_j be the processing time of a job j as specified by the adversary in instance $\mathcal{I}$. Then the perturbed instance $\hat{\mathcal{I}}$ is defined as $\mathcal{I}$ but with processing times $\hat{p}_j$ defined by $\hat{p}_j = (1 + X_j)p_j$ where X_j is chosen at random according to the smoothing distribution. For $0 < \varepsilon < 1$ being a fixed parameter describing the strength of perturbation, we consider two smoothing distributions. In case of a *uniform smoothing distribution*, X_j is chosen uniformly at random from the interval $[-\varepsilon, \varepsilon]$. More formally, $X_j \sim \mathcal{U}(-\varepsilon, \varepsilon)$ where $\mathcal{U}(a, b)$ denotes the continuous uniform distribution as formally defined in Chapter 2. Hence, for $\hat{p}_j$ we have $\hat{p}_j \in [(1 - \varepsilon)p_j, (1 + \varepsilon)p_j]$. In case of a *normal smoothing distribution*, X_j is chosen from a normal distribution with expectation 0, standard deviation $\sigma = \frac{\varepsilon}{\sqrt{2.64}}$ and truncated at -1 and 1. More formally, $X_j \sim \mathcal{N}_{(-1,1)}(0, \sigma^2)$ where $\mathcal{N}_{(a,b)}(\mu, \sigma^2)$ denotes the truncated normal distribution as defined in Chapter 2.

Our goal is to prove a smoothed competitiveness of $O(\varepsilon^{-2} s \log^2 n)$. We analyze the competitiveness by conditioning it on the flow time of Opt and its relation to the flow time of Balance. Let $\mathcal{E}^q_{\text{Opt}}$ be the event that $F^* \in [\alpha^q, \alpha^{q+1})$ and $\mathcal{E}^q_{\text{Balance}}$ be the event that $F > c_1 \alpha^{q+1} \varepsilon^{-2} s \log^2 n$ (for a constant value of c_1 determined by the analysis). Also, denote by $\bar{\mathcal{E}}^q_x$ the respective complementary events. Then for a fixed instance $\mathcal{I}$ we obtain

$$\begin{aligned}
\mathbb{E}_{\hat{\mathcal{I}} \leftarrow N(\mathcal{I})}\left[\frac{F(\hat{\mathcal{I}})}{F^*(\hat{\mathcal{I}})}\right] &= \sum_{q \in \mathbb{N}} \mathbb{E}\left[\frac{F(\hat{\mathcal{I}})}{F^*(\hat{\mathcal{I}})} \mid \mathcal{E}^q_{\text{Opt}} \wedge \bar{\mathcal{E}}^q_{\text{Balance}}\right] \cdot \Pr[\mathcal{E}^q_{\text{Opt}} \wedge \bar{\mathcal{E}}^q_{\text{Balance}}] \\
&\quad + \sum_{q=\lfloor \log_\alpha s \rfloor}^{\lceil \log_\alpha n \rceil} \mathbb{E}\left[\frac{F(\hat{\mathcal{I}})}{F^*(\hat{\mathcal{I}})} \mid \mathcal{E}^q_{\text{Opt}} \wedge \mathcal{E}^q_{\text{Balance}}\right] \cdot \Pr[\mathcal{E}^q_{\text{Opt}} \wedge \mathcal{E}^q_{\text{Balance}}] \\
&\quad + \sum_{q > \lceil \log_\alpha n \rceil} \mathbb{E}\left[\frac{F(\hat{\mathcal{I}})}{F^*(\hat{\mathcal{I}})} \mid \mathcal{E}^q_{\text{Opt}} \wedge \mathcal{E}^q_{\text{Balance}}\right] \cdot \Pr[\mathcal{E}^q_{\text{Opt}} \wedge \mathcal{E}^q_{\text{Balance}}] \ .
\end{aligned}$$

Note that the first sum is by definition directly bounded by $O(\varepsilon^{-2} s \log^2 n)$ and the third one by $O(\sqrt{s})$ according to Theorem 7.9. Thus, we only have to analyze the second sum. We show that we can complement the upper bound on the ratio, which can be as high as $\Theta(\sqrt{n})$ by Theorems 7.9 and 7.12, by $\Pr[\mathcal{E}^q_{\text{Opt}} \wedge \mathcal{E}^q_{\text{Balance}}] \leq \frac{1}{n}$. From now on we consider an arbitrary but fixed $q \geq \lfloor \log_\alpha s \rfloor$, and in the following we analyze $\Pr[\mathcal{E}^q_{\text{Opt}} \wedge \mathcal{E}^q_{\text{Balance}}]$. Let $\Gamma \coloneqq \alpha^{i-1}$ such that i is the largest integer with $\alpha^i \leq c_1 \varepsilon^{-2} \alpha^{q+1} s \log^2 n$. Thus we have $\Gamma \geq c_1 \alpha^{q-1} \varepsilon^{-2} s \log^2 n$.

On a high level, the idea of our proof is as follows: We first define a careful partitioning of the time horizon into consecutive intervals (Section 7.4.1). Depending on the amount of workload released in each such interval and an estimation of the amount of setups required for the respective jobs (Section 7.4.2), we then classify each of them to either be dense or sparse (Section 7.4.3). We distinguish two cases depending on the number of dense intervals in $\mathcal{I}$. If this number is sufficiently large,

F^* is, with high probability (w.h.p.), not much smaller than F (Lemma 7.16). This holds as w.h.p. the perturbation increases the workload in a dense interval so that even these jobs cannot be scheduled with a low flow time by Opt. In case the number of dense intervals is small, the analysis is more involved. Intuitively, we can show that w.h.p. there is only a logarithmic number of intervals between any two consecutive sparse intervals in which the perturbation decreases the workload to a quite small amount. Between such sparse intervals the flow time cannot increase too much (even in the worst case) and during a sparse interval Balance can catch up with the optimum: If taking a look at the flow time of the job completing at time t and continuing this consideration over time, we obtain a sawtooth pattern always staying below a not too large bound for the flow time of Balance (Lemma 7.17).

7.4.1 Partitioning of Instance $\mathcal{I}$

We define a partitioning of the instance $\mathcal{I}$, on which our analysis of the smoothed competitiveness will be based on. We partition the time interval $[r_{min}, r_{max}]$, where r_{min} and r_{max} are the smallest and largest release time, as follows: Let a *candidate interval* C be an interval such that $|C| = \Gamma$ and such that for some k it holds $\sum_{j:r_j \in C, k_j = k} p_j \geq \frac{\Gamma}{4}$. Intuitively, a candidate interval C is an interval on which, in $\hat{\mathcal{I}}$, Balance possibly has to perform more setups than Opt does (which, if all jobs released in the interval belong to S^δ_Γ and under the assumption that $\mathcal{E}^q_{\text{Opt}} \wedge \mathcal{E}^q_{\text{Balance}}$ holds, according to Lemma 7.8 requires a workload of at least $\frac{\Gamma}{2}$ in $\hat{\mathcal{I}}$ and hence, at least $\frac{\Gamma}{4}$ in $\mathcal{I}$). Let C_1 be the first candidate interval C. For $i > 1$ let C_i be the first candidate interval C that does not overlap with C_{i-1}.

Now we consider groups of $\mu := \left\lceil \frac{\varepsilon^2 \Gamma}{c_2 s^2 \log^2 n} \right\rceil$ many consecutive candidate intervals C_i, for some constant c_2 determined by the further analysis. Precisely, these groups are defined as $I_1 = [r_{min}, r(C_\mu)]$, $I_2 = (r(C_\mu), r(C_{2\mu})]$ and so on. In the rest of this chapter we consistently use I_i to denote these intervals. Let $\bigcup_i I_i = [r_{min}, r_{max}]$ by (possibly) extending the last I_i so that its right endpoint is r_{max}. Although it worsens constants involved in the competitiveness, we use $\mu \leq \frac{2\varepsilon^2 \Gamma}{c_2 s^2 \log^2 n}$ for $c_1 \geq \alpha c_2$ for the sake of simplicity.

7.4.2 Estimation of Setups in I_i

Before we can now classify intervals I_i to be dense or sparse, we need an estimate $N_s(I)$ on the number of setups Opt and Balance perform on jobs released in a given interval I. We want $N_s(I)$ to be a value uniquely determined by the instance $\mathcal{I}$ and hence, in particular not to be a random variable. This is essential for our analysis and avoids the computation of any conditional probabilities and the analysis of events not being stochastically independent (for example, the amount of work to be done in a certain interval is influenced by dependent events concerning the workload in the perturbed instance and the required number of setups). For the definition of $N_s(I)$ consider the construction by SetupEstimate(I) in Algorithm 5. For a fixed interval I, it essentially mimics Balance in S^δ_Γ in the sense that Lemma 7.13 holds

Algorithm 5 Description of SetupEstimate(I).

Construct a sequence $(j_1, j_2, \ldots, j_m)$ of all jobs released in I as follows:

(1) For $i = 1, 2, \ldots, m$ set j_i to be job $j \notin (j_1, \ldots j_{i-1})$ with smallest $\bar{r}_j$, where

$$\bar{r}_j := \begin{cases} r_j & \text{if } k_j = k_{j_{i-1}} \\ r_j + \Gamma & \text{else.} \end{cases}$$

To break a tie, prefer job j with $k_j = k_{j_{i-1}}$.

(2) Let $N_s(I)$ be the number of values i such that $k_{j_i} \neq k_{j_{i-1}}$.

completely analogous to Lemma 7.8. Also, note that the construction is indeed invariant to job sizes and hence to perturbations. It should not be understood as an actual algorithm for computing a schedule, however, for ease of presentation we refer to the sequence constructed as if it was a schedule. Particularly, we say that it processes two jobs j_i and $j_{i'}$ with $k_{j_i} = k_{j_{i'}}$ in different batches if there is an i'' such that $i < i'' < i'$ with $k_{j_{i''}} \neq k_{j_i}$.

For two jobs j_1 and j_2 of a common class k which start two batches in SetupEstimate(I), $r_{j_2} \geq r_{j_1} + \Gamma$ holds. Hence, by the exact same line of arguments as in the proof of Lemma 7.8 we have the following lemma.

Lemma 7.13. *Assume $\mathcal{E}^q_{\text{Opt}} \wedge \mathcal{E}^q_{\text{Balance}}$ holds. Let B be a batch in* Opt. *Let I be such that $r_j \in I$ for all $j \in B$. An overhead of at most $2s$ is associated to B in* SetupEstimate*(I).*

Also, if the overhead associated to B in Opt *is smaller than $2s$ and in the schedule of* SetupEstimate*(I) it is $2s$, $w(B) \geq \Gamma - F^* - s =: \bar{w}$ needs to hold and there needs to be an interval of length Γ such that jobs of size at least $\bar{w}$ that belong to B are released in this interval.*

In the next two lemmas we show that $N_s(I_i)$ is indeed a good estimation of the number of setups Opt and Balance have to perform, respectively. Lemma 7.14 essentially follows by Lemma 7.13 together with the definition of I_i to consist of μ many candidate intervals. To prove Lemma 7.15 we exploit the fact that all jobs in S^δ_Γ have a flow time of at least 3Γ by Lemma 7.3 so that Balance and SetupEstimate essentially behave in the same way.

Lemma 7.14. *Assume $\mathcal{E}^q_{\text{Opt}} \wedge \mathcal{E}^q_{\text{Balance}}$ holds and consider I_i for a fixed i. For the overhead of* Opt *it holds $\text{overhead}_{\text{Opt}}(I_i) \geq (N_s(I_i) - 6\mu)s$.*

Proof. Recall that by Lemma 7.13 Opt may have less overhead if it processes some jobs in one batch that are processed in two batches by SetupEstimate(I). However, a necessary condition for this is a workload of jobs of one class with size at least $\frac{\Gamma}{4}$ (in the unperturbed instance $\mathcal{I}$) and released in an interval of length Γ. Let $\tilde{C}_1, \ldots, \tilde{C}_\mu$ be the candidates in I_i. Associate all jobs released in $[l(\tilde{C}_1), l(\tilde{C}_1) + 2\Gamma]$

to candidate $\tilde{C}_1$ and inductively associate all jobs released in $[l(\tilde{C}_j), l(\tilde{C}_j) + 2\Gamma]$ not associated to a candidate $\tilde{C}_{j'}$ for $j' < j$ to $\tilde{C}_j$. Note that by this construction, a workload of at most 3Γ (in the perturbed instance $\hat{\mathcal{I}}$) can be associated to each candidate $\tilde{C}_j$ as otherwise $F^* > \Gamma$ contradicting $\mathcal{E}^q_{\mathrm{OPT}}$. Hence, taking the workload associated to $\tilde{C}_1, \ldots, \tilde{C}_\mu$, the necessary conditions of Lemma 7.13 can be fulfilled at most 6μ times and they cannot be fulfilled for any workload not associated to a candidate interval $\tilde{C}_j$. Hence, we have $\mathrm{overhead}_{\mathrm{OPT}}(I_i) \geq (N_s(I_i) - 6\mu)s$, proving the lemma. □

Lemma 7.15. *Consider an interval I_i and suppose that all jobs from I_i belong to S^δ_Γ. Then it holds $\mathrm{overhead}_{\mathrm{BALANCE}}(I_i) \leq N_s(I_i)s$, where $\mathrm{overhead}_{\mathrm{BALANCE}}(I_i)$ denotes the overhead of* BALANCE *associated to jobs j with $r_j \in I_i$.*

Proof. Consider the subschedule S' of BALANCE starting with the first job from I_i to which overhead is associated and ending with the last one from I_i to which overhead is associated. Let Z and Z' be the sequences of jobs as induced by SETUPESTIMATE(I_i) and S', respectively. We remove all jobs not released during I_i from Z' and all jobs which are not part of S' from Z. Compare both resulting sequences Z and Z' and note that they consist of the exact same sets of jobs. If both are identical, the lemma holds because no overhead contributing to $\mathrm{overhead}_{\mathrm{BALANCE}}(I_i)$ can be associated to a job removed from Z'.

Hence, consider the case that Z and Z' differ and let j and j' be the jobs in Z and Z', respectively, at which both sequences differ the first time. Then, in Z job j is preferred over job j' and in Z' job j' is preferred over job j. This can only be the case when j' is scheduled by BALANCE at a time t such that $r_j > t$. Because in Z job j is preferred over j', it needs to hold $r_j \leq r_{j'} + \Gamma$. But at the time t at which j' is scheduled it needs to hold $t \geq r_{j'} + 3\Gamma - s - p_{j'}$ as otherwise its flow time is smaller than 3Γ which contradicts the assumption that it belongs to S^δ_Γ by Lemma 7.3. Hence, we obtain a contradiction as we have $r_j > t$ and $r_j \leq t$ and thus, Z and Z' are identical. □

7.4.3 Good and Bad Events

We are now ready to define good and bad events, which are outcomes of the perturbation of the job sizes that help the algorithm to achieve a small and help the adversary to achieve a high competitiveness, respectively. Let $w_{\mathcal{I}}(I_i) \coloneqq \sum_{j:r_j \in I_i} p_j$ and $w_{\hat{\mathcal{I}}}(I_i) \coloneqq \sum_{j:r_j \in I_i} \hat{p}_j$ denote the workload released in the interval I_i in instance $\mathcal{I}$ and $\hat{\mathcal{I}}$, respectively. We distinguish two kinds of intervals I_i and associate a good and a bad event to each of them. We call an interval I_i to be *dense* if $w_{\mathcal{I}}(I_i) + N_s(I_i)s \geq |I_i|$ and associate an event $\mathcal{D}^{\mathrm{good}}_i$ or $\mathcal{D}^{\mathrm{bad}}_i$ to I_i depending on whether $w_{\hat{\mathcal{I}}}(I_i) \geq w_{\mathcal{I}}(I_i) + \frac{\varepsilon^2\Gamma}{18\sqrt{c_2 s \log n}}$ holds or not. Symmetrically, we call an interval I_i to be *sparse* if $w_{\mathcal{I}}(I_i) + N_s(I_i)s < |I_i|$ and associate an event $\mathcal{S}^{\mathrm{good}}_i$ or $\mathcal{S}^{\mathrm{bad}}_i$ to I_i depending on whether $w_{\hat{\mathcal{I}}}(I_i) \leq w_{\mathcal{I}}(I_i) - \frac{\varepsilon^2\Gamma}{18\sqrt{c_2 s \log n}}$ holds or not.

We next show two lemmas which upper bound $\Pr[\mathcal{E}^q_{\text{OPT}} \wedge \mathcal{E}^q_{\text{BALANCE}}]$ by the probability of occurrences of good events. As we will see in Theorem 7.19 this is sufficient as we can prove the respective good events to happen with sufficiently large probability.

Lemma 7.16. $\Pr[\mathcal{E}^q_{\text{OPT}} \wedge \mathcal{E}^q_{\text{BALANCE}}] \leq \Pr[\textit{no event } \mathcal{D}_i^{\textit{good}} \textit{ happens}]$.

Proof. We show that if an event $\mathcal{D}_i^{\text{good}}$ happens, then $\mathcal{E}^q_{\text{OPT}}$ does not hold. Consider a dense interval I_i and assume an event $\mathcal{D}_i^{\text{good}}$ occurs. Then we have by definition of dense intervals and the definition of event $\mathcal{D}_i^{\text{good}}$ that $w_{\mathcal{I}}(I_i) + N_s(I_i)s \geq |I_i|$ and $w_{\hat{\mathcal{I}}}(I_i) \geq w_{\mathcal{I}}(I_i) + \frac{\varepsilon^2\Gamma}{18\sqrt{c_2}s\log n}$. Taken together, $w_{\hat{\mathcal{I}}}(I_i) + N_s(I_i)s \geq |I_i| + \frac{\varepsilon^2\Gamma}{18\sqrt{c_2}s\log n}$. On the other hand, together with Lemma 7.14 we then have $w_{\hat{\mathcal{I}}}(I_i) + \text{overhead}_{\text{OPT}}(I_i) \geq |I_i| + \frac{\varepsilon^2\Gamma}{18\sqrt{c_2}s\log n} - 6s \cdot \mu$. By Proposition 7.5 we have

$$\begin{aligned} F^* &\geq w_{\hat{\mathcal{I}}}(I_i) + \text{overhead}_{\text{OPT}}(I_i) - |I_i| \geq \frac{\varepsilon^2\Gamma}{18\sqrt{c_2}s\log n} - 6s\left(\frac{2\varepsilon^2\Gamma}{c_2 s^2 \log^2 n}\right) \\ &\geq \frac{\varepsilon^2\Gamma}{18\sqrt{c_2}s\log n}\left(1 - \frac{12\cdot 18}{\sqrt{c_2}\log n}\right) \geq \frac{c_1\alpha^{q-1}\log n}{18\sqrt{c_2}}\left(1 - \frac{12\cdot 18}{\sqrt{c_2}\log n}\right) > \alpha^{q+1} \end{aligned}$$

for sufficiently large $c_1 > c_2$ and n. Then $\mathcal{E}^q_{\text{OPT}}$ does not hold. □

In Theorem 7.19 we will see that the number N_D of dense intervals in $\mathcal{I}$ can be bounded by $N_D < 7\log n$ as otherwise the probability for event $\mathcal{E}^q_{\text{OPT}}$ to hold is only $\frac{1}{n}$.

Thus, we consider the case $N_D < 7\log n$. Consider the sequence of events associated to sparse intervals. A *run* of events $\mathcal{S}_i^{\text{bad}}$ is a maximal subsequence such that no event $\mathcal{S}_i^{\text{good}}$ happens within this subsequence.

Lemma 7.17. *If* $N_D < 7\log n$, $\Pr[\mathcal{E}^q_{\text{OPT}} \wedge \mathcal{E}^q_{\text{BALANCE}}] \leq \Pr[\exists \textit{ run of } \mathcal{S}_i^{\textit{bad}} \textit{ of length } \geq 14\log n]$.

Proof. We assume that all runs of events $\mathcal{S}_i^{\text{bad}}$ are shorter than $14\log n$ and $\mathcal{E}^q_{\text{OPT}} \wedge \mathcal{E}^q_{\text{BALANCE}}$ holds and show a contradiction. From $\mathcal{E}^q_{\text{BALANCE}}$ we can deduce by Lemma 7.3 that S^δ_Γ exists. Since we will use the following reasoning iteratively, let $S = S^\delta_\Gamma$. Using the terminology from Lemma 7.3, let $j_1, j_2, \ldots, j_m$ be the jobs in S and, as before, ℓ be the number of batches in S. By Lemma 7.3 it needs to hold $\sum_{i=1}^{\ell} w_{\hat{\mathcal{I}}}(B_i) + r_{j_1} - r_{j_m} \geq 3\Gamma - (\ell-1)s$. Let $I_{\iota+1}$ be the first interval I_i such that $l(I_{\iota+1}) \geq r_{j_1}$. Let κ be chosen such that κ is the smallest integer where in $I_{\iota+\kappa}$ an event $\mathcal{S}_{\iota+\kappa}^{\text{good}}$ occurs if κ exists and otherwise set κ such that $I_{\iota+\kappa}$ ends with r_{max}. Note that it holds $\kappa < 21\log n$ because of the assumption $N_D < 7\log n$ and the length of the longest run. Let $I_\iota = [\min_{1\leq i\leq m} r_{j_i}, l(I_{\iota+1}))$. We claim that all jobs belonging to $\bigcup_{i=0}^{\kappa} I_{\iota+i}$ need to have a flow time below $\alpha\Gamma$. Assume this is not the

case. We have a contradiction as

$$\begin{aligned}
F^* &\geq w_{\hat{\mathcal{I}}}(I(S)) + \text{overhead}_{\text{OPT}}(I(S)) - |I(S)| \\
&\geq \sum_{i=1}^{\ell} w_{\hat{\mathcal{I}}}(B_i) + \text{overhead}_{\text{OPT}}(I(S)) + r_{j_1} - r_{j_m} - 2\Gamma \\
&\geq \sum_{i=1}^{\ell} w_{\hat{\mathcal{I}}}(B_i) + r_{j_1} - r_{j_m} - 2\Gamma + \text{overhead}_{\text{BALANCE}}(I(S)) - \frac{21 \log n \cdot 12 s\varepsilon^2\Gamma}{c_2 s^2 \log^2 n} \\
&\geq \Gamma - \frac{252 \log n s\varepsilon^2\Gamma}{c_2 s^2 \log^2 n} \geq \varepsilon^2\Gamma\left(\frac{1}{\varepsilon^2} - \frac{252}{c_2 s \log n}\right) \geq \frac{1}{2}\varepsilon^2 c_1 \varepsilon^{-2}\alpha^{q-1} s \log^2 n > \alpha^{q+1}
\end{aligned}$$

where we used Proposition 7.5 in the first inequality, the fact that $|I(S)| \leq (r_{j_m} - r_{j_1}) + 2\Gamma$ in the second, Lemmas 7.14 and 7.15 in the third, Lemma 7.3 in the fourth and in the remaining inequalities suitable values for $c_1 > c_2$ and the fact that $\Gamma \geq c_1\varepsilon^{-2}\alpha^{q-1} s \log^2 n$. Observe that in case $r(I_{\iota+\kappa}) = r_{max}$, we are done as $\mathcal{E}^q_{\text{BALANCE}}$ cannot hold.

Otherwise, consider the situation directly before the first job $\tilde{j}$ with $r_{\tilde{j}} > r(I_{\iota+\kappa})$ is started. Denote the subschedule of S up to (not including) job $\tilde{j}$ by $\tilde{S}$. Let $\text{overhead}_{\text{BALANCE}}(I)$ be the overhead in S associated to jobs released in the interval I. Let $\text{overhead}_{\text{BALANCE}}(I, \tilde{S})$ and $\text{overhead}_{\text{BALANCE}}(I, \neg\tilde{S})$ be the overhead of jobs released in interval I and which are part and not part of $\tilde{S}$, respectively. Let $w_{\hat{\mathcal{I}}}(I, \tilde{S})$ and $w_{\hat{\mathcal{I}}}(I, \neg\tilde{S})$ be the workload of jobs released in interval I and which are part and not part of $\tilde{S}$, respectively. For brevity let $L = w_{\hat{\mathcal{I}}}([0, r_{j_1}), \tilde{S}) - w_{\hat{\mathcal{I}}}([r_{j_1}, r_{\tilde{j}}), \neg\tilde{S}) + \text{overhead}_{\text{BALANCE}}([0, r_{j_1}), \tilde{S}) - \text{overhead}_{\text{BALANCE}}([r_{j_1}, r_{\tilde{j}}), \neg\tilde{S})$. We can then bound the workload and setups in $\tilde{S}$ by

$$\begin{aligned}
&w_{\hat{\mathcal{I}}}(\tilde{S}) + \text{overhead}_{\text{BALANCE}}(\tilde{S}) \\
\leq& w_{\hat{\mathcal{I}}}([r_{j_1}, l(I_{\iota+\kappa}))) + w_{\hat{\mathcal{I}}}(I_{\iota+\kappa}) + w_{\hat{\mathcal{I}}}([0, r_{j_1}), \tilde{S}) - w_{\hat{\mathcal{I}}}([r_{j_1}, r_{\tilde{j}}), \neg\tilde{S}) \\
&+ \text{overhead}_{\text{BALANCE}}([r_{j_1}, l(I_{\iota+\kappa}))) + \text{overhead}_{\text{BALANCE}}(I_{\iota+\kappa}) \\
&+ \text{overhead}_{\text{BALANCE}}([0, r_{j_1}), \tilde{S}) - \text{overhead}_{\text{BALANCE}}([r_{j_1}, r_{\tilde{j}}), \neg\tilde{S}) \\
\leq& l(I_{\iota+\kappa}) - r_{j_1} + F^* + 21 \log n 12 s \cdot \frac{\varepsilon^2\Gamma}{c_2 s^2 \log^2 n} + |I_{\iota+\kappa}| - \frac{\varepsilon^2\Gamma}{18\sqrt{c_2} s \log n} + L \\
=& r(I_{\iota+\kappa}) - r_{j_1} + F^* + \frac{\varepsilon^2\Gamma}{s \log n 18\sqrt{c_2}}\left(\frac{252 \cdot 18}{\sqrt{c_2}} - 1\right) + L \\
\leq& r(I_{\iota+\kappa}) - r_{j_1} + F^* + \frac{c_1\alpha^{q-1}\log n}{18\sqrt{c_2}}\left(\frac{252 \cdot 18}{\sqrt{c_2}} - 1\right) + L < r(I_{\iota+\kappa}) - r_{j_1} - 2F^* + L,
\end{aligned}$$

where we used Proposition 7.5 together with Lemma 7.14 and the fact that to $I_{\iota+\kappa}$ an event $\mathcal{S}^{\text{good}}_{\iota+\kappa}$ is associated in the second inequality, the lower bound on Γ in the third inequality and suitable values for c_1 and c_2 in the last inequality. Then, job $\tilde{j}$ is started before $r_{j_1} + F_{j_1} + r(I_{\iota+\kappa}) - r_{j_1} - 2F^* + L + s$ and finished by $r(I_{\iota+\kappa}) + F_{j_1} + L$

with flow time $F_{\tilde{j}} \leq F_{j_1} + L$. For $S = S^{\delta}_{\Gamma}$ we have $L \leq -w_{\hat{\mathcal{I}}}([r_{j_1}, r_{\tilde{j}}), \neg\tilde{S}) - \text{overhead}_{\text{BALANCE}}([r_{j_1}, r_{\tilde{j}}), \neg\tilde{S})$ as no jobs with smaller release time than r_{j_1} can be part of S. Thus, $F_{\tilde{j}} < (\alpha - \delta)\Gamma - w_{\hat{\mathcal{I}}}([r_{j_1}, r_{\tilde{j}}), \neg\tilde{S}) - \text{overhead}_{\text{BALANCE}}([r_{j_1}, r_{\tilde{j}}), \neg\tilde{S})$. Now, applying the same arguments with $S = S^{\delta}_{\Gamma}(\tilde{j})$ and using Corollary 7.4 instead of Lemma 7.3, we claim that we find a further job with flow time at most $(\alpha - \delta)\Gamma$ (and all jobs processed before have flow time below $\alpha\Gamma$). Iterating this process we will eventually reach the end of the instance without finding a job with flow time at least $\alpha\Gamma$, contradicting that $\mathcal{E}^{q}_{\text{BALANCE}}$ holds. Formally, it remains to prove that the claim $F_{\tilde{j}} \leq (\alpha - \delta)\Gamma$ also holds for later iterations. We introduce the following notations. Denote by j_1^0 and $\tilde{j}^0$ the jobs j_1 and $\tilde{j}$ from the first iteration as before, respectively. For the following iterations, we use the notation j_1^i and $\tilde{j}^i$ for the respective jobs of the i-th iteration. Note that $j_1^i = \tilde{j}^{i-1}$ and we will thus only use j_1^0, but $\tilde{j}^i$ at all other places. Similarly, we denote $\tilde{S}^i$ as the symbol $\tilde{S}$ from the i-th iteration. We define $w_{\hat{\mathcal{I}}}(I, \bigwedge_{i=0}^{\nu} \neg\tilde{S})$ as the natural extension of the prior definition to be the workload of jobs released in interval I and which are not part of any of the subschedules $\tilde{S}^0, \ldots, \tilde{S}^{\nu}$. For $\text{overhead}_{\text{BALANCE}}(I, \bigwedge_{i=0}^{\nu} \neg\tilde{S})$, the extension is defined similarly. We now prove the following claim inductively:

$$F_{\tilde{j}^{\nu}} \leq (\alpha - \delta)\Gamma - w_{\hat{\mathcal{I}}}\left([r_{j_1^0}, r_{\tilde{j}^{\nu}}), \bigwedge_{i=0}^{\nu} \neg\tilde{S}^i\right) - \text{overhead}_{\text{BALANCE}}\left([r_{j_1^0}, r_{\tilde{j}^{\nu}}), \bigwedge_{i=0}^{\nu} \neg\tilde{S}^i\right).$$

For ease of notation, we introduce the combined expression of $\text{com}(I, X) \coloneqq w_{\hat{\mathcal{I}}}(I, X) + \text{overhead}_{\text{BALANCE}}(I, X)$. As we have already seen the induction base, assume the claim is true for $\nu - 1$. By using

$$\begin{aligned}\text{com}\left([r_{j_1^0}, r_{\tilde{j}^{\nu-1}}), \bigwedge_{i=0}^{\nu-1} \neg\tilde{S}^i\right) &= \text{com}\left([r_{j_1^0}, r_{\tilde{j}^{\nu-1}}), \bigwedge_{i=0}^{\nu} \neg\tilde{S}^i\right) \\ &\quad + \text{com}\left([r_{j_1^0}, r_{\tilde{j}^{\nu-1}}), \bigwedge_{i=0}^{\nu-1} \neg\tilde{S}^i \wedge \tilde{S}^i\right)\end{aligned}$$

and $\text{com}\left([r_{j_1^0}, r_{\tilde{j}^{\nu-1}}), \bigwedge_{i=0}^{\nu} \neg\tilde{S}^i \wedge \tilde{S}^i\right) = \text{com}\left([r_{j_1^0}, r_{\tilde{j}^{\nu-1}}), \tilde{S}^i\right)$, we estimate

$$\begin{aligned}F_{\tilde{j}^{\nu}} &\leq F_{\tilde{j}^{\nu-1}} + L \\ &\leq (\alpha - \delta)\Gamma - \text{com}\left([r_{j_1^0}, r_{\tilde{j}^{\nu-1}}), \bigwedge_{i=0}^{\nu-1} \neg\tilde{S}^i\right) \\ &\quad + \text{com}([0, r_{\tilde{j}^{\nu-1}}), \tilde{S}^{\nu}) - \text{com}([r_{\tilde{j}^{\nu-1}}, r_{\tilde{j}^{\nu}}), \neg\tilde{S}^{\nu})\end{aligned}$$

$$
\begin{aligned}
&= (\alpha-\delta)\Gamma - \operatorname{com}\left([r_{j_1^0}, r_{\tilde{j}^{\nu-1}}), \bigwedge_{i=0}^{\nu} \neg\tilde{S}^i\right) - \operatorname{com}\left([r_{j_1^0}, r_{\tilde{j}^{\nu-1}}), \tilde{S}^\nu\right) \\
&\quad + \underbrace{\operatorname{com}([0, r_{j_1^0}), \tilde{S}^\nu)}_{=0} + \operatorname{com}([r_{j_1^0}, r_{\tilde{j}^{\nu-1}}), \tilde{S}^\nu) - \operatorname{com}([r_{\tilde{j}^{\nu-1}}, r_{\tilde{j}^\nu}), \neg\tilde{S}^\nu) \\
&\leq (\alpha-\delta)\Gamma - \operatorname{com}\left([r_{j_1^0}, r_{\tilde{j}^{\nu-1}}), \bigwedge_{i=0}^{\nu} \neg\tilde{S}^i\right) - \operatorname{com}([r_{\tilde{j}^{\nu-1}}, r_{\tilde{j}^\nu}), \bigwedge_{i=0}^{\nu} \neg\tilde{S}^i) \\
&= (\alpha-\delta)\Gamma - \operatorname{com}\left([r_{j_1^0}, r_{\tilde{j}^\nu}), \bigwedge_{i=0}^{\nu} \neg\tilde{S}^i\right).
\end{aligned}
$$

The claim follows. Also, as $\operatorname{com}\left([r_{j_1^0}, r_{\tilde{j}^\nu}), \bigwedge_{i=0}^{\nu} \neg\tilde{S}^i\right)$ is always non-negative, the claim implies $F_{\tilde{j}^\nu} \leq (\alpha-\delta)\Gamma$. □

To finally bound the probability of good events to happen, we need the following lemma.

Lemma 7.18. *Let J be a set of jobs and assume that processing times are perturbed according to a uniform or normal smoothing distribution. With probability at least $\frac{1}{10}$, $w_{\hat{\mathcal{I}}}(J) \geq w_{\mathcal{I}}(J) + \frac{\varepsilon}{5}(\lfloor w_{\mathcal{I}}(J)\rfloor/3)^{0.5}$. Also, with probability at least $\frac{1}{10}$, $w_{\hat{\mathcal{I}}}(J) \leq w_{\mathcal{I}}(J) - \frac{\varepsilon}{5}(\lfloor w_{\mathcal{I}}(J)\rfloor/3)^{0.5}$.*

Proof. We first show the lemma for the case of a uniform smoothing distribution and then continue with the case of a normal distribution.

Uniform Smoothing Distribution. We can describe the perturbed workload as $w_{\hat{\mathcal{I}}}(J) = w_{\mathcal{I}}(J) + X$ where X is the random variable given by $X = \sum_{j\in J} X_j$ where $X_j \sim \mathcal{U}(-\varepsilon p_j, \varepsilon p_j)$. Let $w = \lfloor w_{\mathcal{I}}(J)\rfloor$. We distinguish two cases depending on whether there exists a job $j' \in J$ with $p_{j'} \geq \frac{2}{5}\varepsilon\frac{\sqrt{w}}{\sqrt{3}}$. In the positive case, we have $\Pr[X_{j'} \geq \frac{1}{5}\varepsilon(w/3)^{0.5}] \geq \frac{1}{4}$ and $\Pr[\sum_{j\in J\setminus\{j'\}} X_j \geq 0] \geq \frac{1}{2}$. Hence, in this case the lemma holds. Consider the case in which for all $j \in J$ it holds $p_j < \frac{2}{5}\varepsilon\frac{\sqrt{w}}{\sqrt{3}}$. We then have $\mathbb{E}(X_j) = 0$ and $\mathbb{V}(X_j) = \sigma_j^2 = \frac{1}{3}(\varepsilon p_j)^2 \geq \frac{1}{3}\varepsilon^2$, for all j. Also $\mathbb{E}[|X_j|^3] = \frac{1}{4}(\varepsilon p_j)^3$. We now bound the probability we are interested in by a normal approximation. Let $S = \frac{X_1+\ldots+X_{|J|}}{\sqrt{\sigma_1^2+\ldots+\sigma_{|J|}^2}}$, F be the distribution of S and $\delta = \sup_x |F(x) - \Phi(x)|$, where $\Phi(x)$ is the distribution function of the standard normal distribution. By the central limit theorem as described in Chapter 2 we have

$$
\begin{aligned}
&\Pr\left[X_1 + \ldots + X_{|J|} \leq \frac{\varepsilon}{5}\left(\lfloor w_{\mathcal{I}}(J)\rfloor/3\right)^{0.5}\right] \leq \Pr\left[X_1 + \ldots + X_{|J|} \leq \frac{1}{5}\left(\sum \sigma_i^2\right)^{0.5}\right] \\
\leq &\Phi\left(\frac{1}{5}\right) + \delta \leq 0.57926 + \delta.
\end{aligned}
$$

Also, we can bound δ using standard Berry-Esseen bounds as described in Chapter 2

by

$$\delta \leq 0.56 \left(\sum \sigma_i^2\right)^{-\frac{1}{2}} \cdot \max \frac{\mathbb{E}[|X_i|^3]}{\sigma_i^2} \leq \frac{1}{\varepsilon\sqrt{w/3}} \cdot \frac{3}{4}\varepsilon\frac{2}{5}\sqrt{w/3} < \frac{3}{10}.$$

Together with the symmetry of the uniform distribution, we obtain the lemma for uniform perturbations.

Normal Smoothing Distribution. Recall that $\varepsilon^2 = 2.64\sigma^2$. We can describe the perturbed workload as $w_{\hat{\mathcal{I}}}(J) = w_{\mathcal{I}}(J) + X$ where X is a random variable given by $X = \sum_{j \in J} X_j$ and $X_j \sim \mathcal{N}_{(-p_j,p_j)}(0, (\sigma p_j)^2)$. We have $\mathbb{E}[X_j] = 0$ and $\mathbb{V}[X_j] = (\sigma p_j)^2(1 - \frac{2/\sigma\phi(2/\sigma)}{2\Phi(2/\sigma)-1}) \geq (\sigma p_j)^2(1 - \frac{0.11}{0.95}) \geq 0.88(\sigma p_j)^2$. Also we have

$$\begin{aligned}\mathbb{E}[|X_j|^3] &= \frac{1}{\sqrt{2\pi}\sigma(\Phi(\frac{p_j}{\sigma}) - \Phi(\frac{-p_j}{\sigma}))} \int_{-p_j}^{p_j} |x|^3 \exp(-0.5(\frac{x}{\sigma})^2)\,\mathrm{d}x \\ &\leq \frac{1}{\sqrt{2\pi}\sigma 0.68} 4\sigma^4 \leq 2.35\sigma^3.\end{aligned}$$

By the central limit theorem we have

$$\begin{aligned}&\Pr\left[X_1 + \ldots + X_{|J|} \leq \frac{\varepsilon}{5}\left(\lfloor w_{\mathcal{I}}(J)\rfloor/3\right)^{0.5}\right] \\ =&\Pr\left[X_1 + \ldots + X_{|J|} \leq \frac{1}{5}\left(0.88\sigma^2\lfloor w_{\mathcal{I}}(J)\rfloor\right)^{0.5}\right] \\ \leq&\Pr\left[X_1 + \ldots + X_{|J|} \leq \frac{1}{5}\left(\sum \mathbb{V}[X_i]\right)^{0.5}\right] \leq \Phi\left(\frac{1}{5}\right) + \delta \leq 0.57926 + \delta.\end{aligned}$$

Again, we can bound δ using standard Berry-Esseen bounds by

$$\begin{aligned}\delta &\leq 0.56 \left(\sum \mathbb{V}[X_i]\right)^{-\frac{1}{2}} \cdot \max \frac{\mathbb{E}[|X_i|^3]}{\sigma_i^2} \\ &\leq 0.597\frac{1}{\sqrt{|J|}\sigma} \cdot 2.6705\sigma \leq 1.5942885/\sqrt{|J|} < \frac{3}{10},\end{aligned}$$

if $|J| \geq 29$. Note that we can assume $|J| \geq 29$ as we only apply the bound in Theorem 7.19 and thus under the assumption that $w_{\mathcal{I}}(J) \geq \mu\frac{\Gamma}{4} \geq \mu\frac{c_1\varepsilon^{-2}\log^2 ns}{4\alpha^2}F^* = \Omega(\log^2 n)F^*$, which requires at least $\Omega(\log^2 n)$ many jobs. Together with the symmetry of the normal distribution, we obtain the lemma. □

Theorem 7.19. *The smoothed competitiveness of* Balance *is* $O(\sigma^{-2}\log^2 n)$ *when processing times* p_j *are perturbed independently at random to* $\hat{p}_j = (1 + X_j)p_j$ *where* $X_j \sim \mathcal{U}(-\varepsilon, \varepsilon)$ *or* $X_j \sim \mathcal{N}_{(-1,1)}(0, \sigma^2)$.

Proof. Recall that it only remains to prove $\Pr[\mathcal{E}^q_{\text{Opt}} \wedge \mathcal{E}^q_{\text{Balance}}] \leq \frac{1}{n}$. First consider the case $N_D \geq 7\log n$. For a fixed i we have $\Pr[\mathcal{D}_i^{\text{good}}] \geq \frac{1}{10}$ because of the following reasoning. According to Lemma 7.18, it holds $\Pr[w_{\hat{\mathcal{I}}}(I_i) \geq$

$w_{\mathcal{I}}(I_i) + \frac{\varepsilon}{5}(\lfloor w_{\mathcal{I}}(I_i) \rfloor / 3)^{0.5}] \geq \frac{1}{10}$. By definition of I_i we can bound $\frac{\varepsilon}{5}(\lfloor w_{\mathcal{I}}(I_i) \rfloor / 3)^{0.5} \geq \frac{\varepsilon}{5}(\frac{\mu\Gamma}{12})^{0.5} \geq \varepsilon^2 \frac{\Gamma}{18\sqrt{c_2 s \log n}}$ which implies $\Pr[\mathcal{D}_i^{\text{good}}] \geq \frac{1}{10}$. Because $N_D \geq 7 \log n$, the probability that no event $\mathcal{D}_i^{\text{good}}$ occurs is then upper bounded by $(1 - \frac{1}{10})^{7 \log n} \leq \frac{1}{n}$ and so is $\Pr[\mathcal{E}^q_{\text{OPT}} \wedge \mathcal{E}^q_{\text{BALANCE}}]$ according to Lemma 7.16.

For the case $N_D < 7 \log n$ the same line of arguments gives $\Pr[\mathcal{S}_i^{\text{good}}] \geq \frac{1}{10}$ for each sparse interval I_i. Hence, the probability for a run of events $\mathcal{S}_i^{\text{bad}}$ of length at least $14 \log n$ is at most $\frac{1}{n}$ and so is $\Pr[\mathcal{E}^q_{\text{OPT}} \wedge \mathcal{E}^q_{\text{BALANCE}}]$ by Lemma 7.17. □

CHAPTER 8

Cost-efficient Scheduling on Machines from the Cloud

Cloud computing provides a modern concept for provisioning computing power, usually in the form of virtual machines (VMs), that offers users potential benefits but also poses algorithmic challenges to be addressed. On the positive side, main advantages for users of moving their business to the cloud are manifold: Possessing huge computing infrastructures as well as asset and maintenance costs associated with it are no longer required and hence, costs and risks of large investments are significantly reduced. Instead, users are only charged to the extent they actually use computing power and it can be scaled up and down depending on current demands. In practice, two ways of renting machines from the cloud are typically available: on-demand-instances without long-term commitment, and reserved instances. In the former case, machines can be rented for any (not prespecified) duration and the charging is by the hour (Amazon EC2 [Ama19]), minute or even by the second (Google Cloud [Goo19]); whereas in the latter case, specific (long-term) leases of various lengths are available. We focus on the former case and note that the latter might better be captured within the leasing framework introduced in [Mey05].

Despite the potential benefits, the cloud also bears new challenges in terms of renting and utilizing resources in a (cost-)efficient way. Typically, the cloud offers diverse VM types differing in the provided resources and further characteristics. One might think of different machine types and each machine either being a high-CPU instance, which especially suits the requirements for computation-intensive jobs, or a high-memory instance for I/O-intensive jobs. Another example is the distinction of CPU and GPU instances. Certain computations can be accelerated by the use of GPUs and in [LK11], for instance, it is observed that, for certain workloads, one can significantly improve performance when not only using classical CPU but

additionally GPU instances. Hence, the performance of a job can strongly depend on the VM type on which it is executed and one should account for this in order to be cost-efficient while guaranteeing a good performance. Performance in turn can be expressed in terms of user-defined due dates or deadlines implicitly given by a desired quality of service level.

Also, despite the fact that computing power can be scaled according to current demands, one needs to take into account that this scaling is not instantaneous and it initially could take some time for a VM to be ready for processing workload. A recent study [MH12] shows such setup times to typically be in the range of several minutes for common cloud providers and hence, to be non-ignorable for the overall performance [MLH10; PLM18].

We formally introduce the model we consider in this chapter in Section 8.1 and give an overview of related work and our results in Sections 8.2 and 8.3, respectively. We then highlight the main challenges for online algorithms by giving general lower bounds and upper bounds for naive approaches in Section 8.4. Our main algorithm is designed and analyzed in Section 8.5 and it is based on three major steps: In Section 8.5.1, we show how we can, by losing only a constant factor compared to an optimal solution, structure schedules with respect to rental intervals and sets of jobs processed together on a common machine. We then define a variant of the problem of finding such structured schedules and show how to solve it offline in Section 8.5.2. This "offline helper" is then used as one important ingredient in the design of our online algorithm in Section 8.5.3.

8.1 Model & Notation

The problem CloudScheduling is extracted from the preceding observations and features the three key properties mentioned before: Firstly, setup times for the startup of machines; secondly, (two) heterogeneous machine types[1] and the option for the scheduling algorithm to choose between these types; thirdly, the presence of deadlines to formulate requirements on the finishing time of jobs. We formally specify the problem by defining the machine environment, job characteristics and the objective function in the following.

Machine Environment There are two *types* $\tau \in \{A, B\}$ of machines, which can be rented in order to process workload: A machine M of type τ can be opened at any time a_M and, after a setup taking some non-negligible $s_\tau \in \mathbb{R}_{>0}$ time units was performed, can be closed at any time b_M with $b_M \geq a_M + s_\tau$, for $a_M, b_M \in \mathbb{R}_{\geq 0}$. Note that b_M does not need to be known beforehand and particularly not at a_M when the machine is opened. A machine can process workload during the interval $[a_M + s_\tau, b_M]$ and it can process at most one job at a time. The *rental cost* incurred

[1] In general, considering two types of machines is often a natural and important special case of fully heterogeneous machines. It has been considered theoretically in different settings, for example, in [NHL82; DE92; CYZ13; CMV17; AZM18].

by a machine M is determined by the duration for which it is open and is formally given by $c_\tau \cdot (b_M - a_M)$, for some $c_\tau \in \mathbb{R}_{>0}$. Note that a machine cannot process any workload nor can be closed during the setup. However, one is still charged for the duration of the setup. We refer to the cost $c_\tau \cdot s_\tau$ incurred by a machine during its setup as *setup cost* and to the cost $c_\tau \cdot (b_M - a_M - s_\tau)$ incurred during the remaining time it is open as *processing cost*. Without loss of generality, we assume that $s_B \geq s_A$ and normalize c_A so that $c_A = 1$. Then, the cost ratio c is given by $c \coloneqq \frac{c_B}{c_A} = c_B$. Practical observations show that c is usually in the range of 1 to a few hundred [MH12; Ama19]. We also use τ^+ and τ^- to refer to the more expensive and cheaper machine type, respectively. That is, $\tau^+ = B$ if $c \geq 1$ and $\tau^+ = A$ otherwise, and $\tau^- \in \{A, B\}$ with $\tau^- \neq \tau^+$. We use $c(\tau^x) = c_{\tau^x}$ for $x \in \{+, -\}$.

Job Characteristics The workload of an instance is represented by a set J of n jobs, where each job j is characterized by a *release time* $r_j \in \mathbb{R}_{\geq 0}$, a *deadline* $d_j \in \mathbb{R}_{>0}$, and *sizes* $p_{j,\tau} \in \mathbb{R}_{>0}$ for all $\tau \in \{A, B\}$, describing the processing time of job j when assigned to a machine of type τ. Observe that therefore machines of the same type are considered to be identical while those of different types are unrelated (also cf. Chapter 5). Throughout this chapter, we assume by using a suitable time scale that $p_{j,\tau} \geq 1$ for all $j \in J$ and all $\tau \in \{A, B\}$. The jobs arrive online over time at their release times at which the scheduler gets to know the jobs together with their deadlines and sizes. Each job needs to be completely processed by *one* machine before its respective deadline, i.e., to any job j we have to assign a *processing interval* I_j of length $p_{j,\tau}$ that is contained in j's *time window* $[r_j, d_j] \supseteq I_j$ on a machine M of type τ that is open during the entire interval $I_j \subseteq [a_M + s_\tau, b_M]$. A machine M of type τ is called *exclusive* machine for a job j if it only processes j and $b_M - a_M = s_\tau + p_{j,\tau}$. We define the *slack* of a job j as the amount by which j is shiftable in its window: $\sigma_{j,\tau} \coloneqq d_j - r_j - p_{j,\tau}$ for $\tau \in \{A, B\}$.

Objective Function The objective of CloudScheduling is to rent machines and compute a schedule that minimizes the rental cost given by $\sum_M c_{\tau(M)} \cdot (b_M - a_M)$, where $\tau(M)$ denotes the type of machine M.

We analyze the quality of our algorithms in terms of their competitiveness and assume that problem instances are parameterized by their *minimum slack*. An instance is said to have a minimum slack of β if $\max_\tau\{\sigma_{j,\tau}\} \geq \beta$, for all $j \in J$. Then, for a given β, an algorithm is called *ρ-competitive* if, on any instance with minimum slack β, the rental cost is at most by a factor ρ larger than the cost of an optimal offline algorithm. Note that this is a natural parameterized analog of the general definition of competitiveness as given in Chapter 2.

In the following, we denote an optimal schedule as well as the cost it incurs by Opt. Throughout this chapter (in all upper and lower bounds), we assume that Opt is not too small and in particular, $\text{Opt} = \Omega(c(\tau^+) \cdot r_{max})$ holds, where $r_{max} \coloneqq \max_{j \in J} r_j$. (Similar assumptions are made in the literature [Aza+13], where it is argued (for *identically* priced machines) that the case where $\text{Opt} = o(r_{max})$

is of minor interest as workload and costs are very small.) This bound is true, for example, if the optimal solution has to maintain at least one open machine (of the more expensive machine type) during (a constant fraction of) the considered time horizon. This assumption seems to be reasonable for large-scale systems where, at any time, the decision to make is rather concerned with dozens of machines than whether a single machine is rented at all. For example, consider a setting in which our scheduler is not part of a single end-user enterprise but where we assume a *virtual cloud* model [MH11]. In this model there are three roles involved: Cloud providers who own the computing infrastructure, cloud vendors who rent resources from the providers and end-users/clients who request resources from and are served by a cloud vendor. If we assume our scheduling algorithms to be located at the site of a cloud vendor, requests of a huge amount of clients with different needs will be concentrated at the cloud vendor who in turn is concerned with renting a large amount of resources.

8.2 Related Work

Cloud scheduling has attracted the interest of theoretical researchers during the last years. Azar et al. [Aza+13] consider a scheduling problem where jobs arrive online over time and need to be processed by identical machines rented from the cloud. While machines are paid for the exact time they are used, a fixed setup time s is required before the respective machine is available for processing (and hence, the charging model is identical to ours). In this setting, Azar et al. consider a bicriterial optimization problem where the rental cost is to be minimized while guaranteeing a reasonable maximum delay. An online algorithm that is $(1+\varepsilon, O(1/\varepsilon))$-competitive regarding the cost and the maximum delay, respectively, is provided. Precisely, the delay of a job j is defined by the difference between its finishing time and $r_j + p_j$. They propose an online algorithm that, given a budget of $(1+\varepsilon)$ times the minimum possible cost to schedule all jobs (without any assumption on the maximum delay), finds a solution with a maximum delay of $O(s/\varepsilon)$. A different model for cloud scheduling was considered by Saha [Sah13]. She considers jobs that arrive over time and need to be finished before their respective deadlines. To process jobs, identical machines are available and need to be rented from the cloud. The goal is to minimize the rental cost where a machine that is rented for t time units incurs cost of $\lceil t/D \rceil$ for some fixed D. The problem is considered as an offline as well as an online problem and algorithms that guarantee solutions incurring costs of $O(\alpha)\textsc{Opt}$ (where α is the approximation factor of the algorithm for machine minimization in use, see below) and $O(\log(\frac{p_{max}}{p_{min}}))\textsc{Opt}$, respectively, are designed.

A different, but closely related problem is that of *machine minimization*. In this problem, n jobs with release times and deadlines are considered and the objective is to finish each job before its deadline while minimizing the number of machines used. Note that in this model a machine stays open for an unlimited duration once it is opened and one is only interested in the sole *number* of opened machines.

This problem has been studied in online and offline settings for the general and different special cases. The first result is due to [RT87] where an offline algorithm with approximation factor of $O(\log n / \log\log n)$ is given. This was later improved to $O(\sqrt{\log n / \log\log n})$ by Chuzhoy et al. [Chu+04]. Better bounds have been achieved for special cases; if all jobs have a common release date or equal processing times, constant approximation factors are achieved [YZ09]. In the online case, a lower bound of $\Omega(n + \log(\frac{p_{max}}{p_{min}}))$ and an algorithm matching this bound is given in [Sah13]. For jobs of equal size, an optimal e-competitive algorithm is presented in [Dev+14], where e denotes the Euler constant. This problem of machine minimization has also been considered from the perspective of heterogeneity. In [NHL82], Nakajima et al. assume that there are cheap, slow machines and expensive, fast machines and that each job's time window has a length equal to the processing time on the slow machine. They give NP-hardness results for the case that the processing times on the fast machines are arbitrary. They also provide polynomial algorithms when the processing times on fast machines are all 1 and additionally, either all release times are equal or each job has a fixed starting time at its (arbitrary) release time.

A further area of research that studies rental/leasing problems from an algorithmic perspective and which recently gained attention is that of *resource leasing*. Its focus is on infrastructure problems and while classically these problems are modeled such that resources are bought and then available all the time, in their leasing counterparts resources are assumed to be rented only for certain durations. In contrast to our model, in the leasing framework resources can not be rented for arbitrary durations. Instead, there is a given number K of different leases with individual costs and durations for which a resource can be leased. Also, an online algorithm has to decide on the duration for which a resource is leased already at the beginning of the leasing time. The model was introduced by Meyerson [Mey05] and problems like FacilityLeasing or SetCoverLeasing have been studied in [AG07; NW13; KMP12; AMM14; Li+15] afterward.

A last problem worthmentioning here is *scheduling with calibrations* [Ben+13; FS15; Che+19]. Although it does not consider the aspect of minimizing resources and the number of machines is fixed, it is closely related to machine minimization and shares aspects with our model. There are given a set of m (identical) machines and a set of jobs with release times, deadlines and sizes. After a machine is calibrated at a time t, it is able to process workload in the interval $[t, t+T]$, for some fixed T, and the goal is to minimize the number of calibrations. For sufficiently large m, the problem is similar to a problem we need to solve in Section 8.5.2.

8.3 Our Results

We study algorithms for CloudScheduling where jobs need to be scheduled on machines rented from the cloud such that the rental costs are minimized subject to quality of service constraints. Particularly, compared to existing work on cloud scheduling and machine minimization, the problem introduces the possibility for a

scheduler to choose between different machine types being heterogeneous in terms of prices and processing capabilities. It also captures the fact that due to times for preparing machines and acquiring resources, available computing power does not scale instantaneously.

Our results show the competitiveness to heavily depend on the minimum slack β guaranteed by all jobs. Therefore, we perform a parameterized analysis and study the competitiveness depending on β. While no finite competitiveness is possible for $\beta < s_B$, a naive rule achieves a competitiveness of $\Theta((c + 1/c)s_B)$ for $\beta = (1 + \varepsilon)s_B$, $0 \leq \varepsilon < 1/s_B$, which is in general optimal.

As our main result, for $1/s_B \leq \varepsilon \leq 1$, we present an algorithm with a competitiveness of $O(c/\varepsilon + 1/\varepsilon^3)$ and $O(1/c\varepsilon^2 + 1/\varepsilon^3)$ for $c \geq 1$ and $c < 1$, respectively. These bounds are complemented by lower bounds for online algorithms which are $\Omega(c/\varepsilon)$ and $\Omega(1/c\varepsilon)$, respectively.

8.4 Simple Lower and Upper Bounds

In this section, we give some simple lower bounds on the achievable competitiveness for CloudScheduling and take a look at the quality of an extremely naive algorithm. The lower bounds are designed such that they all fulfill our assumption $\text{Opt} = \Omega(c(\tau^+) \cdot r_{max})$. We begin our study by showing that setup times are hard to cope with for any online algorithm when the minimum slack is below the setup time s_B. This is the case because an online algorithm needs to hold machines ready at any time for arriving jobs with small deadlines.

Proposition 8.1. *If $\beta < s_B$, no online algorithm has a finite competitiveness.*

Proof. We first prove that an online algorithm must have an open machine at any time. Assume to the contrary that there is a time $t \geq s_B$ at which the algorithm has no open machine. Then, the adversary releases one job j with $r_j = t$, $d_j = r_j + p_{j,B} + \beta$ and an arbitrary size $p_{j,B} \geq c(\tau^+)t$ and $p_{j,A} > p_{j,B} + \beta$. To finish this job without violating its deadline, it needs to be started not after $t + \beta$ on a machine of type B. However, this is not possible since there is no open machine at time t and the setup of a type B machine needs time $s_B > \beta$.

Hence, any online algorithm has to rent a machine of type B all the time to be able to guarantee feasibility of its schedule. This clearly yields an unbounded competitiveness. More formally, in this case the adversary can release only one job with $r_1 = 0$, $p_{1,\tau^+} = 1$, $p_{1,\tau^-} > s_{\tau^+} + 1 + \beta$, $d_1 = s_{\tau^+} + 1 + \beta$. An optimal offline algorithm can schedule this job on a machine of type τ^+ with cost $c(\tau^+) \cdot (s_{\tau^+} + 1)$. On the other hand, the online algorithm has an open machine during the whole interval $[0, t']$, for any t'. This implies cost of at least $c(\tau^-)t'$ and hence an unbounded competitiveness as it grows in t' and thus, without bound. □

Due to this impossibility result, we restrict ourselves to cases where the minimum slack is $\beta = (1 + \varepsilon)s_B$ for some $\varepsilon \geq 0$. It will turn out in Lemma 8.4 that for very

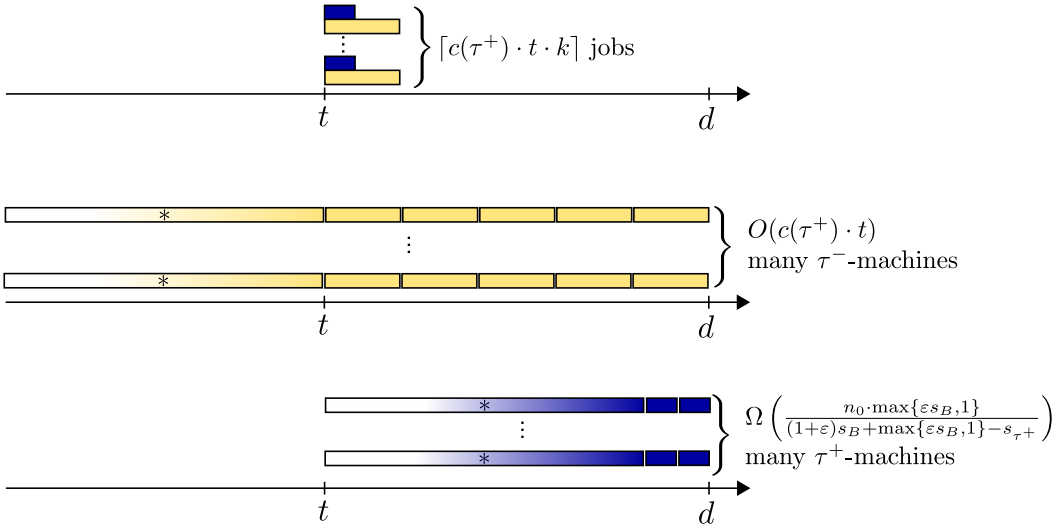

Figure 8.1: Illustration of the construction in the proof of Lemma 8.2 for the case $s_A = s_B$. From top to bottom it shows (relevant parts of) the instance, an optimal schedule and the schedule of an online algorithm. In the instance, each pair of rectangles defines a job j with the blue (dark) rectangle representing p_{j,τ^+} and the yellow (bright) one p_{j,τ^-}. A blue-white (dark) rectangle labeled with an asterisk represents the setup of a machine of type τ^+. A yellow-white (bright) rectangle labeled with an asterisk represents the setup of a machine of type τ^-. The online algorithm cannot place a yellow (bright) rectangle after a setup started at t since the job would be finished slightly after d.

small values of ε, $0 \leq \varepsilon < 1/s_B$, there is a high lower bound on the competitiveness, which depends on s_B, and essentially, we cannot do better than processing each job on its own, exclusive machine. However, for larger values of ε, the situation clearly improves as shown in Lemma 8.2 and leaves room for designing non-trivial online algorithms.

The idea of the proof of Lemma 8.2 is as follows: The adversary releases a whole bunch of jobs as soon as the online algorithm has no machine available. The processing times and deadlines are chosen in such a way that the online algorithm can only process these jobs on machines of the unfavorable type as setting up machines of the other type takes too long. In contrast, an optimal offline algorithm can open machines of the desirable type in advance and can thereby be prepared upon arrival of the jobs.

Lemma 8.2. *If $\beta = (1+\varepsilon)s_B$, there is a lower bound on the competitiveness of $\Omega\left(\frac{c(\tau^+)}{c(\tau^-)}\left(1 + \frac{s_A}{(1+\varepsilon)s_B + \max\{\varepsilon s_B, 1\} - s_A}\right)\right)$.*

Proof. Let s_B and s_A be arbitrary. At $t = 0$ the adversary starts the instance by releasing the first job with $p_{1,A} = p_{1,B} = 1$ and $d_1 = s_B + 1 + \beta$. Afterward, no job

is released until a time $t \geq \max\{1, s_B\}$ at which the online algorithm does not have any open machine. Note that t must exist as otherwise the online algorithm cannot admit bounded competitiveness (also cf. Proposition 8.1). Let $n_0 \coloneqq c(\tau^+) \cdot t \cdot k$, where $k \coloneqq \frac{(1+\varepsilon)s_B + \max\{\varepsilon s_B, 1\}}{\varepsilon s_B + \max\{\varepsilon s_B, 1\} + \max\{0, s_{\tau^+} - s_{\tau^-}\} + \delta}$ for some sufficiently small $\delta > 0$. At time t, the adversary releases $\lceil n_0 \rceil \leq 2n_0$ many identical jobs each with processing times $p_{\tau^+} = \max\{\varepsilon s_B, 1\}$ and $p_{\tau^-} = \varepsilon s_B + \max\{\varepsilon s_B, 1\} + \max\{0, s_{\tau^+} - s_{\tau^-}\} + \delta$ and deadline $d = t + p_{\tau^+} + \beta$. For an illustration see Figure 8.1.

We first bound the cost of the online algorithm. Note that if processed on a machine of type τ^-, any of the jobs released at time t needs to be started not later than $d - p_{\tau^-} = t + p_{\tau^+} + (1+\varepsilon)s_B - p_{\tau^-} = t + \max\{\varepsilon s_B, 1\} + (1+\varepsilon)s_B - (\varepsilon s_B + \max\{\varepsilon s_B, 1\} + \max\{0, s_{\tau^+} - s_{\tau^-}\} + \delta) < t + s_{\tau^-}$. Since at time t no machine is available, no job can be processed on a machine of type τ^- as the job could only be started after a setup at $t + s_{\tau^-}$. On a machine of type τ^+ opened at time t the online algorithm can process jobs during the interval $[t + s_{\tau^+}, d]$. Thus, on each such machine at most $\left\lfloor \frac{(1+\varepsilon)s_B + \max\{\varepsilon s_B, 1\} - s_{\tau^+}}{\max\{\varepsilon s_B, 1\}} \right\rfloor$ many can be processed. Hence, the online algorithm has to rent $\Omega\left(\frac{n_0 \cdot \max\{\varepsilon s_B, 1\}}{(1+\varepsilon)s_B + \max\{\varepsilon s_B, 1\} - s_{\tau^+}}\right)$ machines of type τ^+, each at least open for a duration of $\max\{s_{\tau^+}, s_B - s_{\tau^+}\}$ and hence, incurring cost of $\Omega\left(\left(\frac{n_0 \cdot \max\{\varepsilon s_B, 1\}}{(1+\varepsilon)s_B + \max\{\varepsilon s_B, 1\} - s_{\tau^+}}\right) \cdot c(\tau^+) s_B\right)$.

Now we take a look at the cost of an optimal offline algorithm. The job released at time 0 can be scheduled with cost $c(\tau^-)(s_{\tau^-} + 1)$. On a machine of type τ^- opened at time $t - s_{\tau^-}$ it can process jobs released at time t during the interval $[t, d] = [t, t + (1+\varepsilon)s_B + \max\{\varepsilon s_B, 1\}]$ and thus, $\lfloor k \rfloor \geq \max\{1, \frac{1}{2}k\}$ many. Hence, it can process all $\lceil n_0 \rceil = \lceil c(\tau^+)tk \rceil$ jobs released at time t on $O(c(\tau^+)t)$ machines each open during the interval $[t - s_{\tau^-}, t + (1+\varepsilon)s_B + \max\{\varepsilon s_B, 1\}]$. The overall cost is then bounded by $O(c(\tau^+)t \cdot c(\tau^-)((1+\varepsilon)s_B + \max\{\varepsilon s_B, 1\} + s_{\tau^-})) = O(c(\tau^+)t \cdot c(\tau^-)s_B)$. Taken together, this yields a competitiveness of $\Omega\left(\frac{c(\tau^+)}{c(\tau^-)}\left(1 + \frac{s_A}{(1+\varepsilon)s_B + \max\{\varepsilon s_B, 1\} - s_A}\right)\right)$. □

Corollary 8.3. *Depending on the power of the adversary, for* $1/s_B \leq \varepsilon \leq 1$*, the lower bound from Lemma 8.2 can be bounded by*

1. $\Omega\left(\frac{c(\tau^+)}{c(\tau^-)}\right)$ *if the adversary is restricted and cannot choose* s_A *and* s_B*, and*
2. $\Omega\left(\frac{c(\tau^+)}{c(\tau^-)\varepsilon}\right)$ *otherwise.*

While the construction in Lemma 8.2 exploits a situation in which the online algorithm has to use machines of the unfavorable type, the next lower bound is based on the following idea: A bunch of jobs that can only be processed on machines of type B are released at a time $t \geq s_B$ at which the online algorithm has no open machine of type B. The online algorithm can consequently only process jobs after time $t + s_B$ while an optimal solution can also process jobs during the interval $[t, t + s_B]$ and therefore, needs significantly less machines.

Lemma 8.4. *If $\beta = (1+\varepsilon)s_B$, $0 \leq \varepsilon < 1/s_B$, there is a lower bound on the competitiveness of $\Omega(s_B)$.*

Proof. Let s_A and s_B be arbitrary with $s_B \geq 1$ (since otherwise the lower bound is meaningless anyway). At $t = 0$ the adversary starts the instance by releasing the first job with $p_{1,A} = p_{1,B} = 1$ and $d_1 = s_B + 1 + \beta$. Afterward, the adversary does not release any job until time $t \geq s_B$ at which the online algorithm does not have any open machine. At time t, the adversary releases $\lceil c(\tau^+)t \rceil$ identical jobs with processing times $p_B = 1$ and $p_A > s_B + p_B + 1$ and deadline $d = t + p_B + \beta$. First we can observe that the online algorithm cannot process any job on a machine of type A because (even without setup) the job cannot be finished before $t + s_B + p_B + 1 > t + s_B + p_B + \varepsilon s_B = d$. A machine of type B opened at time t can only process jobs during the interval $[t + s_B, d]$ and hence, only one job. Therefore, the cost of the online algorithm is $\Omega(c(\tau^+)tc(s_B + 1))$. An optimal offline algorithm can schedule all jobs released at time t using only $O(c(\tau^+)t/s_B)$ machines, each processing $\Theta(s_B)$ jobs by opening the respective machines at $t - s_B$ and processing jobs in the interval $[t, d] = [t, t + p_B + \beta]$. The job released at time 0 can be scheduled with cost $c(\tau^-)(s_{\tau^-} + 1)$. Hence, the competitiveness is lower bounded by $\Omega\left(\frac{c(\tau^+)tc(s_B+1)}{c(\tau^+)tc/s_B \cdot s_B}\right) = \Omega(s_B)$. □

Corollary 8.5. *Depending on the power of the adversary, for $0 \leq \varepsilon < 1/s_B$, the lower bound from Lemmas 8.2 and 8.4 can be bounded by*

1. $\Omega\left(\frac{c(\tau^+)}{c(\tau^-)} + s_B\right)$ *if the adversary is restricted and cannot choose s_A, s_B, and*
2. $\Omega\left(\frac{c(\tau^+)}{c(\tau^-)} s_B\right)$ *otherwise.*

Throughout this chapter, when referring to lower bounds (or optimality) the bounds given for an unrestricted adversary are meant.

8.4.1 Simple Approaches

As a first step of studying algorithms in our model we discuss some natural approaches. One of the doubtlessly most naive rules simply decides on the machine type to process a job based on the cost the job incurs on this type. Define the algorithm A_1 such that it assigns each job to its own, exclusive machine and chooses this machine to be of type A if $s_A + p_{j,A} \leq c(s_B + p_{j,B})$ and $\sigma_{j,A} \geq s_A$, or if $\sigma_{j,B} < s_B$. Otherwise, it chooses the machine to be of type B. Then, we have the following bound on the competitiveness of A_1.

Proposition 8.6. *A_1 is $O\left(\frac{c(\tau^+)}{c(\tau^-)} \frac{s_A}{(1+\varepsilon)s_B+1-s_B} + s_B\right)$ and $\Omega(s_B)$-competitive.*

For $0 \leq \varepsilon < 1/s_B$, it has an optimal competitiveness of $\Theta\left(\frac{c(\tau^+)}{c(\tau^-)} s_B\right)$.

For $1/s_B \leq \varepsilon \leq 1$, it has a suboptimal competitiveness of $O\left(\frac{c(\tau^+)}{c(\tau^-)\varepsilon}\right) + \Theta(s_B)$.

Proof. For $\tau, \tau' \in \{A, B\}$, let $J_{\tau,\tau'} \subseteq J$ be the set of jobs assigned to machines of type τ by A_1 and to machines of type τ' by OPT. As a trivial lower bound for OPT we have $\text{OPT} \geq \sum_{j \in J_{A,A} \cup J_{B,A}} p_{j,A} + c(\sum_{j \in J_{A,B} \cup J_{B,B}} p_{j,B})$. Also we know that if a job j is assigned to the more cost-efficient machine type τ by A_1, we have for $\tau' \neq \tau$

$$c_\tau(s_\tau + p_{j,\tau}) \leq c_{\tau'}(s_{\tau'} + p_{j,\tau'}). \tag{8.1}$$

If a job j is not assigned to its more cost-efficient machine type τ it holds

$$p_{j,\tau} \geq p_{j,\tau'} + \beta - s_\tau. \tag{8.2}$$

This directly follows from the fact that $\sigma_{j,\tau} < s_\tau$ and hence, $\sigma_{j,\tau'} \geq \beta$. Then, we can bound the competitiveness as follows

$$\begin{aligned}&\frac{\sum_{j \in J_{A,A}}(p_{j,A} + s_A)}{\sum_{j \in J_{A,A}} p_{j,A}} + \frac{c(\sum_{j \in J_{B,B}}(p_{j,B} + s_B))}{c(\sum_{j \in J_{B,B}} p_{j,B})} \\ +&\frac{\sum_{j \in J_{A,B}}(p_{j,A} + s_A)}{c(\sum_{j \in J_{A,B}} p_{j,B})} + \frac{c(\sum_{j \in J_{B,A}}(p_{j,B} + s_B))}{\sum_{j \in J_{B,A}} p_{j,A}}.\end{aligned}$$

We can bound the first fraction as $\frac{\sum_{j \in J_{A,A}}(p_{j,A}+s_A)}{\sum_{j \in J_{A,A}} p_{j,A}} \leq \frac{\sum_{j \in J_{A,A}} p_{j,A}}{\sum_{j \in J_{A,A}} p_{j,A}} + \frac{|J_{A,A}|s_A}{|J_{A,A}|} = O(s_A)$, where we used $p_{j,A} \geq 1$. By an analogous statement the second fraction is upper bounded by $O(s_B)$. For the third fraction let $J^1_{A,B} \subseteq J_{A,B}$ be the jobs which are assigned to machines of type A as they are more cost-efficient on these machines and let $J^2_{A,B} = J_{A,B} \setminus J^1_{A,B}$. Then we can bound

$$\begin{aligned}&\frac{\sum_{j \in J^1_{A,B}}(p_{j,A} + s_A)}{c(\sum_{j \in J_{A,B}} p_{j,B})} + \frac{\sum_{j \in J^2_{A,B}}(p_{j,A} + s_A)}{c(\sum_{j \in J_{A,B}} p_{j,B})} \\ \leq&\frac{c\sum_{j \in J^1_{A,B}}(p_{j,B} + s_B)}{c(\sum_{j \in J_{A,B}} p_{j,B})} + \frac{\sum_{j \in J^2_{A,B}}(p_{j,A} + s_A)}{c(\sum_{j \in J_{A,B}}(p_{j,A} + (1+\varepsilon)s_B - s_B))} \\ \leq&1 + \frac{c\sum_{j \in J^1_{A,B}} s_B}{c|J^1_{A,B}|} + \frac{1}{c} + \frac{\sum_{j \in J^2_{A,B}} s_A}{c|J^2_{A,B}|(1 + (1+\varepsilon)s_B - s_B)} \\ =&O\left(s_B + \frac{1}{c} + \frac{s_A}{c(1 + (1+\varepsilon)s_B - s_B)}\right),\end{aligned}$$

where we used Equation (8.1) and Equation (8.2) in the first inequality and $p_{j,A} \geq 1$ in the second.

Using analogous arguments, the last fraction can then be upper bounded by $O\left(s_A + c + \frac{cs_A}{(1+(1+\varepsilon)s_B - s_A)}\right)$, proving the claimed upper bound.

The s_B in the bound on the competitiveness can easily be seen to be necessary. Consider an instance where all jobs need to be processed on machines of type B with sizes $p_{j,B} = 1 \ \forall j \in J$ and let them arrive such that all can be assigned to

Algorithm 6 Description of the class GreedyFit.

(1) At each time t at which jobs $J_t = \{j \in J : r_j = t\}$ arrive, Alg processes them in *any* order $j_1, j_2, \ldots$ and one by one.

(1.1) If j_i cannot *reasonably* be processed on any open machine, Alg opens *some* machine.

(1.2) Alg assigns j_i to *some* open machine.

(2) Alg *may* close a machine as soon as it is about to idle.

one machine. Then the above term $\frac{c(\sum_{j \in J_{B,B}} (p_{j,B} + s_B))}{c(s_B + \sum_{j \in J_{B,B}} p_{j,B})}$ is lower bounded by $\frac{1}{2} s_B$ for $n \geq s_B$. □

While this trivial rule is optimal for $0 \leq \varepsilon < 1/s_B$, for larger values of ε the dependence of the competitiveness on the setup time is undesired as it can be quite high and in particular, it is sensitive to the time scale (and recall that we chose a time scale such that the smallest processing time of any job is at least 1). Therefore, the rest of this chapter is devoted to finding an algorithm with a competitiveness being independent of s_B and narrowing the gap to the lower bound for $1/s_B \leq \varepsilon \leq 1$.

One shortcoming of A_1 is the fact that jobs are never processed together on a common machine. A simple idea to fix this and to batch jobs might be to extend A_1 by an AnyFit rule as known from bin packing problems (see, e.g., [Sga14]). Such an algorithm dispatches the jobs one by one and only opens a new machine if the job to be assigned cannot be processed on any already open machine. Then, the job is assigned to some machine it fits into. It turns out that the competitiveness of this approach still depends on the setup time s_B, which we show by proving a slightly more general statement. Consider the class GreedyFit consisting of all deterministic algorithms Alg fitting into the framework as given in Algorithm 6. The terms *some*, *any* and *may* above should be understood as to be defined by the concrete algorithm. A job is said to be *reasonably* processable on a machine M if it does not violate any deadline and the processing cost it incurs does not exceed the cost for setting up and processing the job on a new machine.

Unfortunately, one can still easily get such an algorithm to open a machine on which all upcoming jobs are processed although opening a machine of the other type would be more reasonable.

Proposition 8.7. *If* $1/s_B \leq \varepsilon \leq 1$*, any* GreedyFit *algorithm has a competitiveness of* $\Omega\left(\frac{c(\tau^+)}{c(\tau^-)}\left(1 + \frac{s_A}{(1+\varepsilon)s_B + \varepsilon s_B - s_A}\right) + s_B\right)$.

Proof. First, the lower bound from Lemma 8.2 directly implies the first summand in the competitiveness of GreedyFit. Thus, we focus on the second summand s_B and assume $s_B \geq 1$ (as otherwise this part of the lower bound is meaningless anyway). Fix an arbitrary algorithm Alg belonging to GreedyFit and arbitrary s_B, s_A and

$c \geq 1$. At time $t = 0$ release a job j_0 with $d_{j_0} = s_B + \lceil s_B \rceil \lceil cs_B \rceil$ and $p_{j_0,A} = \lceil cs_B \rceil$. Since the considered algorithm is deterministic and only $p_{j_0,B}$ is not yet fixed, we can now determine and distinguish the following two cases: a) $\exists x \geq 1$ such that if $p_{j_0,B} = x$, ALG opens a machine of type A, and the alternative case b) such an x does not exist.

We start with case a) and set $p_{j_0,B} = x$. At each time $t_i = i$, for $i = 1, 2, \ldots, \lceil s_B \rceil - 1$, a job j with $p_{j,A} = \lceil cs_B \rceil$, $p_{j,B} = 1$ and $d_j = s_B + \lceil s_B \rceil \lceil cs_B \rceil$ is released. By definition of case a) ALG opens a machine of type A at time 0 and then schedules all jobs on it finishing the last one at time $s_A + \lceil s_B \rceil \lceil cs_B \rceil \leq d_j$. Note that ALG does not open a machine of type B as it is assumed to belong to the class GREEDYFIT and at any time t_i, opening a machine of type B and processing the respective job j on it incurs cost of $c(s_B + 1)$ while processing it on the open machine of type A incurs cost of $\lceil cs_B \rceil \leq cs_B + 1$. Therefore, the overall cost of ALG on the given instance is $s_A + \lceil cs_B \rceil \lceil s_B \rceil$.

In case b), ALG opens a machine of type B for processing job j_0. We set $p_{j_0,B} = \lceil s_B \rceil \lceil cs_B \rceil$ so that ALG finishes the job by $s_B + \lceil s_B \rceil \lceil cs_B \rceil \leq d_{j_0}$. Therefore, the cost of ALG is $c(s_B + \lceil s_B \rceil \lceil cs_B \rceil)$.

OPT can always schedule job j_0 on a machine of type A finishing it not later than $s_A + \lceil cs_B \rceil \leq d_{j_0}$ and the remaining jobs on one machine of type B finishing the last one at $s_B + \lceil s_B - 1 \rceil \leq d_j$. The cost incurred by OPT is at most $s_A + \lceil cs_B \rceil + cs_B + c(\lceil s_B \rceil - 1) \leq 4\lceil cs_B \rceil$. In both cases, this gives a lower bound of $\Omega(s_B)$ on the competitiveness of ALG, which concludes the proof. □

8.5 A Batch-Style Competitive Algorithm

Due to the discussed observations, it seems that decisions on the type of a machine to open as well as finding a good assignment of jobs requires more information than given by a single job. Therefore, the general strategy of our algorithm is to collect jobs over some specified period of time and then decide on which machines to rent and how to schedule based on the entire bunch of collected jobs. Thereby it is crucial to specify this period carefully so that jobs do not wait too long before being scheduled but still sufficiently long to base renting decisions on enough jobs. The rough outline of our approach is as follows: In a first step, we prove some structural properties that we can assume (by losing a constant factor) the optimal solution to have (Section 8.5.1). These properties are quite helpful as they restrict the possibilities an algorithm has to take into account when searching for a solution. Based on these structural insights, we formulate an (offline) variant of CLOUDSCHEDULING that we can solve by providing an Integer Linear Program (ILP) (Section 8.5.2). We then use solutions computed based on an enhanced variant of this ILP and computed at regular points in time on subsets of already arrived jobs to define our online algorithm (Section 8.5.3).

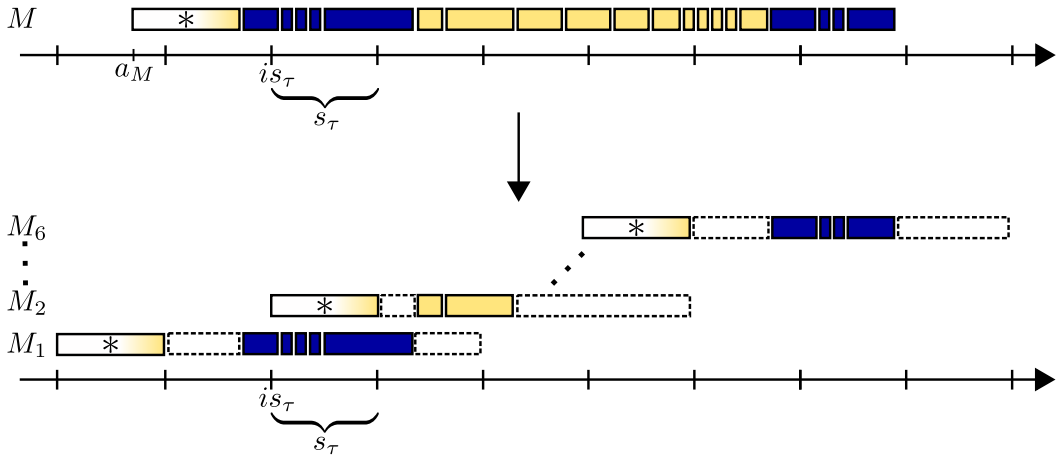

Figure 8.2: Illustration of Property 2. Solid rectangles have the same meaning as in Figure 8.1, dashed rectangles represent idle time added in the construction and $i \in \mathbb{N}$. The left block of blue (dark) jobs equals T_1, the right one equals T_6.

8.5.1 Structuring Schedules

We first show a fundamental and main lemma that provides a way of suitably batching the processing of jobs and structuring the rental intervals. We define intervals of the form $[is_\tau, (i+1)s_\tau)$, for $i \in \mathbb{N}_0$ and $\tau \in \{A, B\}$, as the *i-th τ-interval*. Intuitively, we can partition the considered instance into subinstances each consisting only of jobs released during one B-interval and the times during which machines are open are aligned with these τ-intervals.

Lemma 8.8. *By losing a constant factor, we may assume that* Opt *fulfills the following properties:*

1. *Each job j that is processed on a machine of type τ and fulfills $p_{j,\tau} \geq s_\tau$ is assigned to an exclusive machine,*
2. *each remaining machine M of type τ is open for exactly five τ-intervals, i.e., $[a_M, b_M) = [is_\tau, (i+5)s_\tau)$ for some $i \in \mathbb{N}_0$, and*
3. *if it is opened at $a_M = (i-1)s_\tau$, the beginning of the $(i-1)$-th τ-interval, it only processes jobs released during the i-th τ-interval.*

Proof. We prove the lemma by describing how to modify Opt to establish the three properties while increasing its cost only by a constant factor. Consider a fixed type $\tau \in \{A, B\}$ and let $J_\tau \subseteq J$ be the set of jobs processed on machines of type τ in Opt. Let J_M be the set of jobs processed by a machine M.

Property 1. Any job $j \in J_\tau$ with $p_{j,\tau} \geq s_\tau$ is moved to a new exclusive machine of type τ if not yet scheduled on an exclusive machine in Opt. This increases the cost due to an additional setup and the resulting idle time by $s_\tau + p_{j,\tau} \leq 2p_{j,\tau}$.

Applying this modification to all jobs j with $p_{j,\tau} \geq s_\tau$ therefore increases the cost of Opt by a factor of at most three and establishes the desired property. From now on, we will assume for simplicity that all jobs j scheduled on a machine of type τ fulfill $p_{j,\tau} < s_\tau$.

Property 2. Next, we establish the property that each remaining machine is open for exactly four τ-intervals; when establishing the third property, this is extended to five intervals as claimed in the lemma. For an illustration see Figure 8.2. Consider a fixed machine M. Partition the time during which M is open into intervals of length s_τ by defining $I_k \coloneqq [a_M + ks_\tau, a_M + (k+1)s_\tau)$ for $k \in \{1, \ldots, \lceil \frac{b_M - a_M}{s_\tau} - 1 \rceil\}$. Let t_j be the starting time of job $j \in J_M$ and let $T_k \coloneqq \{j \in J_M : t_j \in I_k\}$ partition the jobs of J_M with respect to the interval during which they are started. We replace machine M by $\lceil \frac{b_M - a_M}{s_\tau} - 1 \rceil$ machines $M_1, M_2, \ldots$ such that M_i processes the jobs from T_i keeping the jobs' starting times as given by Opt and setting $a_{M_i} \coloneqq \min_{j \in T_i} t_j - s_\tau$ and $b_{M_i} \coloneqq \max_{j \in T_i}(t_j + p_{j,\tau})$. Observe that the cost originally incurred by M is $c_\tau(b_M - a_M)$ while those incurred by the replacing machines $M_1, M_2, \ldots$ are at most $c_\tau(b_M - a_M) + c_\tau(\lceil \frac{b_M - a_M}{s_\tau} - 1 \rceil)s_\tau \leq 2c_\tau(b_M - a_M)$ where the additional term stems from the additional setups. Since it holds that $p_{j,\tau} < s_\tau$ for all $j \in J_M$, we conclude that $b_{M_i} - a_{M_i} \leq 3s_\tau$ for all M_i. For each M_i, we can now simply decrease a_{M_i} and increase b_{M_i} such that M_i is open for exactly four τ-intervals. This maintains the feasibility and increases the cost incurred by each machine M_i from at least $c_\tau s_\tau$ to at most $4c_\tau s_\tau$. Applying these modifications to all machines M, establishes the claimed property while increasing the overall cost by a factor of at most eight.

Property 3. It remains to prove the third property. Again consider a fixed machine M. Let $N_{i,M} \coloneqq \{j \in J_M : r_j \in [is_\tau, (i+1)s_\tau)\}$, $i \in \mathbb{N}_0$, be the (sub-)set of jobs released during the i-th τ-interval and processed by M. Furthermore, let $\#(M) \coloneqq |\{i : N_{i,M} \neq \emptyset\}|$ be the number of τ-intervals from which M processes jobs. If $\#(M) \leq 3$, we replace M by $\#(M)$ machines M_i each processing only jobs from $N_{i,M}$. To define the schedule for M_i, let $N_{i,M} = \{j_{i_1}, j_{i_2}, \ldots\}$ such that $t_{i_1} < t_{i_2} < \ldots$ holds. We reset the starting times of the jobs in $N_{i,M}$ by setting $t_{i_1} \coloneqq r_{i_1}$ and $t_{i_k} \coloneqq \max\{r_{i_k}, t_{i_{k-1}} + p_{i_{k-1},\tau}\}$, for $k > 1$, and set $a_{M_i} \coloneqq (i-1)s_\tau$ and $b_{M_i} \coloneqq a_{M_i} + 5s_\tau$. This gives a feasible schedule for $N_{i,M}$ since no starting time is increased and according to Property 2 we have $\sum_{j \in N_{i,M}} p_{j,\tau} \leq 3s_\tau$ and hence, $\max_{j \in N_{i,M}} r_j + \sum_{j \in N_{i,M}} p_{j,\tau} < (i+4)s_\tau = b_{M_i}$. Also, the replacing machines fulfill the three properties and the cost is increased by a factor less than four.

For the complementary case where $\#(M) > 3$, we argue as follows. For an illustration see Figure 8.3. Observe that due to the already established Property 2., M is open for exactly four τ-intervals and hence, $N_{i,M} = \emptyset$ for all $i \geq \frac{a_M}{s_\tau} + 4$. Note that $\frac{a_M}{s_\tau} \in \mathbb{N}_0$ due to Property 2. Also, for each $i \in \{\frac{a_M}{s_\tau} - 1, \frac{a_M}{s_\tau}, \frac{a_M}{s_\tau} + 2, \frac{a_M}{s_\tau} + 3\}$, we can move the jobs from $N_{i,M}$ to new machines by an argument analogous to the one given for the case $\#(M) \leq 3$ above. Hence, we have established the property that M only processes jobs $j \in \bigcup_{i=0}^{a_M/s_\tau - 2} N_{i,M} \cup N_{a_M/s_\tau + 1, M}$ and applying the modification to all machines M from the set $\mathcal{M}$ of machines fulfilling $\#(M) > 3$, increases the cost by a constant factor.

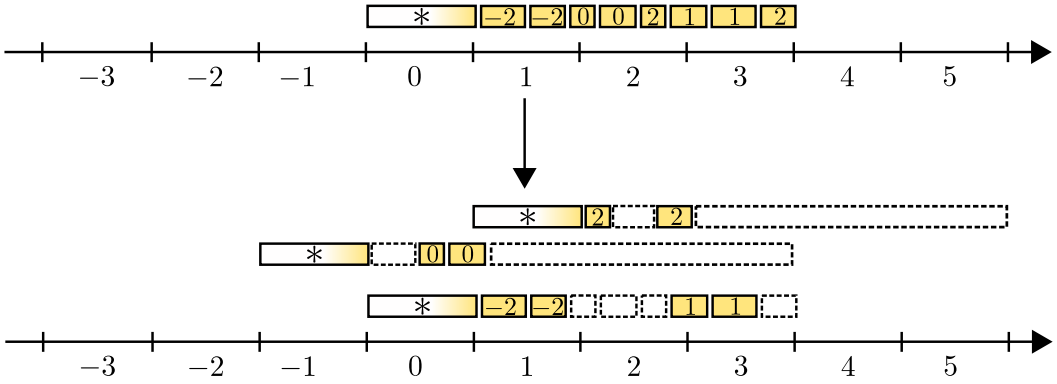

Figure 8.3: Illustration of Property 3. The interval annotated with i represents the interval $[a_M + is_\tau, a_M + (i+1)s_\tau)$. Numbers within rectangles denote the interval during which the respective job is released so that all jobs labeled with j belong to $N_{i,M}$ with $i = \frac{a_M}{s_\tau} + j$. A machine cannot process any job with a label ≥ 4. By losing a constant factor, we can assume that it only processes jobs labeled with 1 or some value ≤ -2. Since jobs with a label ≤ -2 have a deadline not left to the interval 1, they can easily be rescheduled (in intervals annotated with -1 and 0) such that finally each machine only processes jobs directly released in the interval after its setup (with label 1).

In order to finally establish the third property, our last step proves how we can reassign all those jobs j which are processed on a machine $M \in \mathcal{M}$ with $j \in \bigcup_{i=0}^{a_M/s_\tau - 2} N_{i,M}$. Note that jobs belonging to $N_{a_M/s_\tau+1,M}$ for some $M \in \mathcal{M}$ can remain on machine M as this does not violate Property 3. Let $J' = \bigcup_{M\in\mathcal{M}} \bigcup_{i=0}^{a_M/s_\tau-2} N_{i,M}$ be the set containing these jobs and partition them according to their release times by defining $N_i \coloneqq (\bigcup_{M\in\mathcal{M}} N_{i,M}) \cap J'$. Note that for any $j \in N_i$ processed on a machine M it holds $d_j \geq a_M + s_\tau \geq (i+3)s_\tau$, for all $i \in \mathbb{N}_0$. Let $w_i \coloneqq \sum_{j\in N_i} p_{j,\tau}$. We can assign all jobs from N_i to new machines fulfilling the three properties as follows: We open $\lceil \frac{w_i}{s_\tau} \rceil$ new machines at time $(i-1)s_\tau$ and keep them open for exactly five τ-intervals. Due to the fact that for all jobs $j \in N_i$ it holds $r_j \leq (i+1)s_\tau$ and $d_j \geq (i+3)s_\tau$, we can accommodate a workload of at least s_τ in the interval $[(i+1)s_\tau, (i+3)s_\tau]$ on each machine by assigning jobs from N_i to it in any order. By these modifications the cost increase due to cases where $w_i \geq s_\tau$ is given by an additive of at most $c_\tau \sum_{i:w_i\geq s_\tau} \lceil \frac{w_i}{s_\tau} \rceil 5s_\tau$, which is $O(\textsc{Opt})$ since $\textsc{Opt} \geq c_\tau \sum_{i:w_i\geq s_\tau} w_i$. The overall increase in the cost due to the cases where $0 < w_i < s_\tau$ is given by an additive of $c_\tau \sum_{0\leq i\leq \lfloor \frac{r_{max}}{s_\tau} \rfloor : 0<w_i<s_\tau} 5s_\tau$. In case the sum is larger than 0, it is upper bounded by $c_\tau(5r_{max} + 5s_\tau) = O(\textsc{Opt})$ due to our assumption $\textsc{Opt} = \Omega(c_\tau \cdot r_{max})$ and since $\textsc{Opt} = \Omega(c_\tau s_\tau)$ as it rents at least one machine of type τ. □

Here we want to note that the last step of the proof of Lemma 8.8 is the only place

where we use our assumption that $\textsc{Opt} = \Omega(c \cdot r_{max})$. Without this assumption, the cost of a schedule with the properties of Lemma 8.8 could (only) be bounded by $O((1 + \frac{c \cdot r_{max}}{\textsc{Opt}})\textsc{Opt})$ instead of $O(\textsc{Opt})$. Also note that in a schedule of Lemma 8.8, machines of type τ processing jobs released in the interval $[0, s_\tau)$ are opened at $t = -s_\tau$. Although this is not actually possible, it simplifies the presentation and is fine for our purposes: Throughout this chapter, the way we use the result makes sure that in the actual schedules that we finally compute such situations will not occur.

By Lemma 8.8, we can partition any instance into subinstances such that the i-th subinstance consists of those jobs released during the interval $[is_B, (i+1)s_B)$ and solve them separately. In the rest of this chapter, we assume for simplicity and without loss of generality that the entire instance only consists of jobs released during the interval $[0, s_B)$.

Furthermore, we can deduce two additional statements from Lemma 8.8 as given in Lemmas 8.9 and 8.10. To this end, we divide the jobs into three sets according to their sizes. Let

$$\begin{aligned} J_1 &\coloneqq \{j \in J : p_{j,A} \geq s_A \wedge p_{j,B} \geq s_B\}, \\ J_2 &\coloneqq \{j \in J : p_{j,A} < s_A \wedge p_{j,B} < s_B\}, \\ J_3 &\coloneqq J \setminus (J_1 \cup J_2). \end{aligned}$$

That is, J_1 contains those jobs for which the processing cost dominates the setup cost on both machine types. The set J_2 contains those jobs for which the processing cost does not dominate the setup cost on either of the two machine types and J_3 all remaining jobs. We also define $J_3 = J_{3,1} \cup J_{3,2} = \{j \in J_3 : p_{j,A} \geq s_A \wedge p_{j,B} < s_B\} \cup \{j \in J_3 : p_{j,A} < s_A \wedge p_{j,B} \geq s_B\}$. Based on these definitions and by Lemma 8.8 we directly obtain the following two lemmas.

Lemma 8.9. *For an optimal schedule for jobs from J_2, we may assume each machine of type $\tau \in \{A, B\}$ to be open for exactly five τ-intervals. If a job $j \in J_2$ is processed on a machine of type τ and $r_j \in [is_\tau, (i+1)s_\tau)$, $i \in \mathbb{N}_0$, its processing interval is completely contained in the interval $[is_\tau, (i+4)s_\tau]$.*

Lemma 8.10. *For an optimal schedule for jobs from $J_{3,1}$, we may assume each machine of type B to be open for exactly five B-intervals. If a job $j \in J_{3,1}$ is processed on a machine of type A, it is processed on an exclusive machine and if j is processed on a machine of type B and $r_j \in [is_B, (i+1)s_B)$, $i \in \mathbb{N}_0$, its processing interval is completely contained in the interval $[is_B, (i+4)s_B]$.*

Analogous statements hold for jobs from $J_{3,2}$.

8.5.2 Tentative Subschedules

We now use the results from the previous section to describe and solve the following relaxed offline variant, denoted OfflineCloudScheduling, of our problem: All

$$\min \quad 5s_A \sum_i z_i + 5cs_B \cdot z_B + \sum_{\substack{x(I,j):I\in\mathcal{I}_A(j)\\ j\in J_1\cup J_{3,1}}} x(I,j)p_{j,A} + c \sum_{\substack{x(I,j):I\in\mathcal{I}_B(j)\\ j\in J_1\cup J_{3,2}}} x(I,j)p_{j,B}$$

$$\text{s.t.} \quad \sum_{\substack{j\in J_2\cup J_{3,2}:\\ r_j\in[(i-1)s_A,is_A)}} \sum_{I\in\mathcal{I}_A(j):t\in I} x(I,j) \le z_i \qquad \forall t\in L_i, i\in\{1,\dots,{}^{s_B}/_{s_A}\} \quad (8.3)$$

$$\sum_{j\in J_2\cup J_{3,1}} \sum_{I\in\mathcal{I}_B(j):t\in I} x(I,j) \le z_B \qquad \forall t\in L_B \quad (8.4)$$

$$\sum_{I\in\mathcal{I}_A(j)\cup\mathcal{I}_B(j)} x(I,j) = 1 \qquad \forall j\in J \quad (8.5)$$

$$x(I,j)\in\{0,1\} \qquad \forall j\in J, I\in\mathcal{I}_A(j)\cup\mathcal{I}_B(j) \quad (8.6)$$

Figure 8.4: Integer Linear Program for OfflineCloudScheduling.

jobs are known in advance and setups do not take any time but still incur the respective cost as before.

By Lemma 8.9 and Lemma 8.10, we can formulate OfflineCloudScheduling (which is NP-hard by the NP-hardness of classical BinPacking) as an ILP as given in Figure 8.4. We use $\mathcal{I}_\tau(j)$ to denote the set of all possible processing intervals of job j on a machine of type $\tau\in\{A,B\}$. Intuitively, $\mathcal{I}_A(j)\cup\mathcal{I}_B(j)$ describes all possible ways how job j can be scheduled. Note that $\mathcal{I}_A(j)$ and $\mathcal{I}_B(j)$ are built under the assumptions from Lemmas 8.8 to 8.10. For each $I\in\mathcal{I}_A(j)\cup\mathcal{I}_B(j)$, the indicator variable $x(I,j)$ states if job j is processed in interval I. We use L_B to denote all left endpoints of intervals in $\bigcup_{j\in J_2\cup J_{3,1}}\mathcal{I}_B(j)$ and L_i to denote all left endpoints of intervals in $\bigcup_{j\in J_2\cup J_{3,2}:r_j\in[(i-1)s_A,is_A)}\mathcal{I}_A(j)$. Additionally, we use a variable z_B to denote the number of (non-exclusive) machines of type B that we rent. The variable z_i describes the number of machines of type A that we rent and that process jobs released during the $(i-1)$-th A-interval. For simplicity we assume that s_B is an integer multiple of s_A.

We are now asked to minimize the cost for machines of type B plus those for machines of type A, taking into account that each machine of type τ is either open for a duration of exactly $5s_\tau$ time units (first two summands of the objective function) or is an exclusive machine (last two summands). The constraints given by Equation (8.3) and Equation (8.4) ensure that, at any point in time, the number of jobs processed on (non-exclusive) machines does not exceed the number of open machines. Additionally, constraints given by Equation (8.5) and Equation (8.6) ensure that each job is completely processed by exactly one machine in a contiguous interval.

Observe that in general this ILP has an infinite number of variables since each $\mathcal{I}_\tau(j)$ contains all possible processing intervals of j on a machine of type τ. Yet,

as we prove in Lemma 8.11, there is an efficient way (adapted from [Chu+04]) to reduce the number of variables that need to be considered to $O(|J|^2)$ such that afterward a solution only being by a constant factor larger than the optimal one of the original formulation exists. Note that we can then solve the resulting ILP optimally (though it may take non-polynomial time) yielding $O(1)$-approximate solutions to the original formulation.

Lemma 8.11. *By losing a constant factor, we can assume that* $|\bigcup_{j\in J}(\mathcal{I}_A(j) \cup \mathcal{I}_B(j))| = O(|J|^2)$ *holds.*

Proof. The following proof is an extension of one from [Chu+04], which studies a related issue for the problem of machine minimization.

We show how to reduce the number $|\bigcup_{j\in J}(\mathcal{I}_A(j) \cup \mathcal{I}_B(j)|$ of *job intervals* to $O(|J|^2)$. Note that due to Lemmas 8.8 to 8.10, it holds

$$\left|\bigcup_{\tau\in\{A,B\}}\bigcup_{j\in J_1}\mathcal{I}_\tau(j)\right| = O(|J|), \left|\bigcup_{j\in J_{3,1}}\mathcal{I}_A(j)\right| = O(|J|), \left|\bigcup_{j\in J_{3,2}}\mathcal{I}_B(j)\right| = O(|J|).$$

For the sake of simplicity, we only argue that $|\bigcup_{j\in J_2}(\mathcal{I}_A(j) \cup \mathcal{I}_B(j))| = O(|J|^2)$ in the following. The reasoning for the remaining sets $J_{3,1}$ and $J_{3,2}$ is analogous but even simpler as it is a more restricted case than the one for J_2 due to the aforementioned bounds on the number of job intervals on one of the two machine types.

We now describe the construction and afterward prove the claimed bound on the possible loss. Recall Lemmas 8.8 and 8.9. Let M_0 be a machine of type B such that it is open until $4s_B$ (which corresponds to machines represented by z_B in Figure 8.4) and for each $i \in \{1, \ldots, \frac{s_B}{s_A}\}$ let M_i be a machine of type A which can process jobs during the interval $[(i-1)s_A, (i+3)s_A)$ (which corresponds to machines represented by z_i in Figure 8.4). We first construct sets $D_i \subseteq J_2$, $i \in \{0, 1, \ldots, \frac{s_B}{s_A}\}$, such that all jobs from D_i can be scheduled on machine M_i. To do so, perform the following steps for each machine M_i separately: Consider the set J_2 of all jobs. At each time machine M_i gets idle, we greedily assign the job that will finish earliest among all jobs that are not yet scheduled on M_i and that can still meet their deadlines. Let P_i be the set which contains all right endpoints of processing intervals of jobs scheduled on M_i together with all release times and deadlines of the remaining jobs. Note that $|P_i| = O(|J|)$, for all $i \in \{0, 1, \ldots, \frac{s_B}{s_A}\}$. Now we reduce the number of job intervals of any job j as follows. Each left endpoint of a job interval in $\mathcal{I}_B(j)$ is rounded down to its nearest point in P_0 and each right endpoint is rounded up to its nearest point in P_0. Similarly, each left endpoint of a job interval in $\mathcal{I}_A(j)$ is rounded down and each right endpoint is rounded up to its nearest point in its corresponding P_i. By this construction, the number of different job intervals becomes $O(|J|^2)$.

Next we prove the bound on the cost of the modified instance. Consider an optimal solution S for the original instance. If a job $j \in D_0$ is scheduled on a machine of type A in S, remove it from D_0 and similarly, if a job $j \in D_i$, $i \in \{1, \ldots, \frac{s_B}{s_A}\}$,

is scheduled on a machine of type B in S, remove it from D_i. Also, for each job $j \in D_i$, remove this job from S. Now, all processing intervals of jobs $j \in S$ must contain at least one point from the respective set P_i by construction as otherwise j would belong to D_i. Thus, after performing the modifications of the release times and deadlines, the jobs $j \in S$ can always be scheduled by twice the number of type A and type B machines used in S. By definition, the remaining jobs in the sets D_i can be assigned to one machine for each set. Note that due to the removal of jobs from D_0, this set is empty if the original solution S had no machine of type B. Similarly, for any fixed $i \in \{1, 2, \ldots, \frac{s_B}{s_A}\}$, D_i is empty if the original solution S had no machine of type A processing jobs released during the interval $[(i-1)s_A, is_A)$. Hence, there is a schedule for the modified instance with at most three times the cost of the original schedule. □

In the following we will use solutions to OfflineCloudScheduling in our online algorithm. To this end, we will compute ILP solutions at regular points in time for the set of jobs arrived after the last computation and before the current time t. As these solutions do not include setup times and resulting schedules may require a job to be started at some time between its release time and time t (at which we only compute the schedule), they cannot directly be realized as computed and the online algorithm has to care for the feasibility when defining the actual schedule. Therefore, to emphasize its character of not being final, we call a schedule computed as a solution to OfflineCloudScheduling a *tentative schedule*.

8.5.3 The BatchedDispatch Algorithm

In this section, we describe and analyze our online algorithm. It is essentially based on several observations concerning how to restrict tentative schedules so that they can be transformed into feasible solutions afterward. We first show Lemma 8.12, which relates the cost of an optimal schedule to the cost of one where (1) jobs are finished at least s_B time units before their deadline and (2) no jobs released during different intervals $[i\Delta, (i+1)\Delta)$ share a machine, where $\Delta := \frac{1}{2}\varepsilon s_B$. Intuitively, (1) gives us the chance to delay jobs by s_B for performing setups without violating the original deadlines. By (2) we can schedule jobs from different such intervals independently from each other achieving the cost bound as given in the lemma. Throughout the following description we assume that ε is known to the algorithm in advance. However, afterward we show how this assumption can easily be dropped. For simplicity we define $\tilde{d}_j := d_j - s_B$ and $\tilde{\sigma}_{j,\tau} := \tilde{d}_j - r_j - p_{j,\tau}$, for all $\tau \in \{A, B\}$ and all $j \in J$. We then have the following property.

Property 1. For all $j \in J$ it holds $\max\{\tilde{\sigma}_{j,A}, \tilde{\sigma}_{j,B}\} \geq 2\Delta$ and $\tilde{d}_j > r_j + 2\Delta$.

Proof. We have $\max\{\tilde{\sigma}_{j,A}, \tilde{\sigma}_{j,B}\} = \max\{\sigma_{j,A}, \sigma_{j,B}\} - s_B \geq (1+\varepsilon)s_B - s_B = \varepsilon s_B = 2\Delta$ for all $j \in J$. It also implies $\tilde{d}_j > r_j + 2\Delta$ for all $j \in J$. □

Lemma 8.12. *There is a schedule that (1) meets all deadlines $\tilde{d}_j$, (2) in which no machine processes any two jobs j, j' with $r_j \in [i\Delta, (i+1)\Delta)$ and $r_{j'} \in [i'\Delta, (i'+1)\Delta)$ with $i \neq i'$ and $i, i' \in \mathbb{N}_0$ and (3) which has cost $O\left(c(\tau^+)/c(\tau^-)\varepsilon + 1/\varepsilon^2\right)$ Opt.*

Furthermore, the jobs from $j \in \bar{J} \coloneqq \{j : \exists \tau \in \{A, B\}$ such that $\tilde{\sigma}_{j,\tau} < 0\}$ can be scheduled with cost $O\left(c(\tau^+)/c(\tau^-)\varepsilon + 1/\varepsilon^2\right)$ Opt and each job $j \in \bar{J}$ with $\tilde{\sigma}_{j,\tau} < 0$ is processed on a machine of type $\tau' \neq \tau$. The jobs from $j \in J \setminus \bar{J}$ can be scheduled with cost $O(1/\varepsilon^2)$Opt.

Proof. We show how to modify an optimal schedule such that the desired properties and bounds on the cost hold.

First, let $E_\tau \subseteq \bar{J}$ be the set of jobs $j \in \bar{J}$ that are processed on a machine of type τ in Opt. By increasing the cost by a factor of $O\left(\frac{c(\tau^+)}{c(\tau^-)\varepsilon}\right)$ we can assume, for all $\tau \in \{A, B\}$, that each job $j \in E_\tau$ is processed on an exclusive machine of type $\tau' \neq \tau$. This is true since $\sigma_{j,\tau'} \geq \beta$ implying $p_{j,\tau} \geq p_{j,\tau'} + \varepsilon s_B$. Hence, processing j on an exclusive machine of type τ' incurs cost of $c(\tau')(s_{\tau'} + p_{j,\tau'})$ while it incurs cost of at least $c(\tau)p_{j,\tau}$ in Opt, proving the claimed factor. Consequently, all jobs from $E_A \cup E_B = \bar{J}$ fulfill the desired properties.

Therefore, consider the remaining set of jobs $J \setminus \bar{J}$. We first establish Property (2) for these jobs. By Lemmas 8.9 and 8.10 each machine is open for a duration of exactly $5s_\tau$ and hence, can process jobs from $O(\frac{1}{\varepsilon})$ many different intervals $[i\Delta, (i+1)\Delta)$. Hence, by assuming that each machine only processes jobs released during a common interval $[i\Delta, (i+1)\Delta)$, $i \in \mathbb{N}_0$, the costs are increased by a factor of $O(1/\varepsilon)$, proving Property (2).

To establish Property (1), consider any machine M^i processing only jobs released in the interval $[i\Delta, (i+1)\Delta)$. Let $J_k^{M^i}$ be the set of jobs finished in the interval $[k\Delta, (k+1)\Delta)$ on machine M^i, $k \in \mathbb{N}_0$. Note that $J_k^{M^i} \neq \emptyset$ for at most $O(1/\varepsilon)$ different k's, which again follows from Lemmas 8.9 and 8.10. Therefore, by losing an additional factor of $O(1/\varepsilon)$ we can replace M^i by machines M_k^i such that M_k^i processes exactly those jobs from $J_k^{M^i}$ that are started and finished in $[k\Delta, (k+1)\Delta)$. It remains to show how we can decrease the starting times of all jobs on any fixed machine M_k^i so that all deadlines $\tilde{d}_j$ are met. Let $k_1, k_2, \ldots$ be the jobs processed on machine M_k^i such that for the starting times t_j it holds $t_{k_1} < t_{k_2} < \ldots$ We now distinguish three cases depending on when the jobs on M_k^i are finished compared to their release times. In case $k\Delta \geq s_B + (i+1)\Delta$, let $d = t_{k_1} - (i+1)\Delta$ and reset the starting time t_j of job j on M_k^i to $t_j = t_j - d$. Observe that $t_j \geq (i+1)\Delta \geq r_j$ for all $j \in \{k_1, k_2, \ldots\}$ and each job j is finished by $t_j - d + p_j \leq d_j - d \leq d_j - s_B \leq \tilde{d}_j$.

In case $(i+1)\Delta \leq k\Delta < s_B + (i+1)\Delta$, all jobs $k_1, k_2, \ldots$ can be processed one after another starting with k_1 at $(i+1)\Delta$. Then no job is started before its release time and since the workload of all of them is not larger than Δ, each job $j \in \{k_1, k_2, \ldots\}$ is finished by $(i+1)\Delta + \Delta \leq r_j + 2\Delta \leq \tilde{d}_j$, where we used Property 1 in the last inequality.

In case $k\Delta < (i+1)\Delta$, it follows $k\Delta \leq i\Delta$. Then, all jobs $j \in \{k_1, k_2, \ldots\}$ are finished by $(k+1)\Delta \leq i\Delta + \Delta \leq r_j + \Delta \leq \tilde{d}_j$. Hence, we have established a schedule as desired by the lemma. □

We are now ready to describe our final algorithm. The formal description is given in Algorithm 7. The algorithm relies on Δ and its relation to the (modified) slack $\tilde{\sigma}_{j,\tau}$ of jobs and uses the following partition of J into the sets:

$$J'_1 = \{j : \tilde{\sigma}_{j,B} \geq 2\Delta \wedge (\tilde{\sigma}_{j,A} \geq 2\Delta \vee p_{j,A} \leq \Delta)\},$$
$$J'_2 = \{j : \tilde{\sigma}_{j,B} \geq 2\Delta \wedge (\tilde{\sigma}_{j,A} < 2\Delta \wedge p_{j,A} > \Delta)\},$$
$$J'_3 = \{j : \tilde{\sigma}_{j,B} < 2\Delta \wedge \tilde{\sigma}_{j,A} \geq 2\Delta\}.$$

Note that $\{j : \tilde{\sigma}_{j,B} < 2\Delta \wedge \tilde{\sigma}_{j,A} < 2\Delta\} = \emptyset$ by Property 1 and hence, that the three sets actually form a partition of the set of jobs J. Before proving the correctness and bounds on the cost, we shortly describe the high level ideas of our algorithm skipping technical details, which should become clear during the descriptions for the individual job sets and the analysis. The algorithm proceeds in phases, where each phase is devoted to scheduling jobs released during an interval of length Δ. At the end of a given phase, we compute tentative schedules for each set of jobs which arrived in this phase using the modified deadlines $\tilde{d}_j$ so that the schedules can be delayed and extended by the necessary setup times without violating any original deadline (cf. Step (2.2)). In order to be able to guarantee that the tentative schedules can be turned into feasible solutions, we have to define several additional restrictions on starting times and machines to use (cf. Step (2.1)). These restrictions are carefully designed depending on the characteristics of jobs concerning their slacks. This approach ensures that we can turn solutions into feasible schedules while guaranteeing that costs are not increased too much. We also precautionarily open machines (and possibly close them without using them) for jobs that are required to be started early and hence, for which we cannot extend the tentative schedule by the necessary setups (cf. Step (1.2)). The feasibility is crucial since tentative schedules are not only delayed due to the added setups but also because of computing the schedules only at the end of a phase where jobs may have already been available for Δ time units.

Note that by Lemma 8.12, the overall costs of all tentative schedules, *if each was computed without the additional restrictions* of Step (2.1), would be bounded by $O((c(\tau^+)/c(\tau^-)\varepsilon + 1/\varepsilon^2)\textsc{Opt})$. Hence, it is sufficient to show that the schedules for a single phase are feasible and the costs are increased not too much by posing the additional restrictions on the tentative schedules. Lemmas 8.14 to 8.16 prove that both properties hold for all three sets of jobs. As used in these proofs, we first show that jobs having a sufficiently large slack on the machine they are processed on, can be assumed not to be started too early after they are released: Intuitively, if jobs that have a slack of at least 2Δ and are released in the interval $[(i-1)\Delta, i\Delta)$ are processed on a machine before $i\Delta$, the large slack allows us to shift them together to a later point in time on a new machine.

Lemma 8.13. *Consider a fixed $\tau \in \{A, B\}$ and a set J' of jobs such that $r_j \in [(i-1)\Delta, i\Delta)$ for all $j \in J'$. With constant loss we can assume that in a schedule fulfilling Lemma 8.12 all jobs $j \in J'$ fulfilling $\tilde{\sigma}_{j,\tau} \geq 2\Delta \vee p_{j,\tau} \leq \Delta$ are not started*

Algorithm 7 Description of the algorithm BatchedDispatch(ε).

Let $C = (s_B \geq s_A + \Delta \wedge \frac{s_A}{c} > \Delta)$.
In phase $i \in \mathbb{N}$ process all jobs j with $r_j \in [(i-1)\Delta, i\Delta)$:

(1) Upon arrival of a job j at time t

(1.1) Classify j to belong to the set J'_1, J'_2 or J'_3.
If C holds, also define $J'_{2,1} \coloneqq J'_2$ and $J'_{2,2} \coloneqq \emptyset$.
Otherwise, $J'_{2,1} \coloneqq \{j \in J'_2 : cp_{j,B} > s_A\}$ and $J'_{2,2} \coloneqq J'_2 \setminus J'_{2,1}$.

(1.2) If C does not hold, open a machine of type A at t for each $j \in J'_{2,1}$.

(2) At time $i\Delta$

(2.1) Compute tentative schedules for the job sets using the modified deadlines $\tilde{d}_j$ and these additional restrictions:

- Starting times of jobs from $J'_1, J'_{2,2}$ and J'_3 are at least $i\Delta$,
- starting times of jobs from $J'_{2,1}$ on machines of type B are at least $i\Delta$ and
- jobs from $J'_{2,2}$ (J'_3) only use machines of type B (type A).

(2.2) Realize the tentative schedules by increasing the starting times of jobs from

- J'_1 by s_τ on machines of type τ,
- $J'_{2,1}$ by s_B on machines of type B and by $s_A + \Delta$ or s_A depending on whether C holds or not on machines of type A,
- $J'_{2,2}$ by s_B and
- J'_3 by s_A,

and doing the respective setups at time $i\Delta$. Use machines from Step (1.2) for $J'_{2,1}$ if necessary and otherwise close them after finishing the setups.

before $i\Delta$ if processed on a machine of type τ.

Proof. Consider a schedule S fulfilling Lemma 8.12. Consider an arbitrary machine M of type τ in S and let $J_M = \{j_1, j_2, \ldots\}$ denote the jobs processed on M such that for their starting times $t_{j_1} < t_{j_2} < \ldots$ holds. By losing a factor of two we can assume that J_M only processes jobs j that fulfill $\tilde{\sigma}_{j,\tau} \geq 2\Delta \vee p_{j,\tau} \leq \Delta$. Let $k \in \mathbb{N}$ be chosen such that $t_{j_k} + p_{j_k,\tau} < i\Delta$ and $t_{j_{k+1}} + p_{j_{k+1},\tau} \geq i\Delta$, that is, j_k is the last job finished on M before $i\Delta$. Now, we can leave all jobs $j \in \{j_{k+2}, j_{k+3}, \ldots\}$ unaffected (i.e., still start them at the respective t_j) and we can process j_{k+1} on an exclusive machine started at time $t_{j_{k+1}} = i\Delta > r_{j_{k+1}}$. This is feasible since j_{k+1} is finished at time $i\Delta + p_{j_{k+1},\tau} \leq \tilde{d}_{j_{k+1}}$, where, in case $\tilde{\sigma}_{j_{k+1},\tau} \geq 2\Delta$ holds, the last inequality follows from $\tilde{\sigma}_{j_{k+1},\tau} = \tilde{d}_{j_{k+1}} - r_{j_{k+1}} - p_{j_{k+1},\tau} \geq 2\Delta$ and $r_{j_{k+1}} \geq (i-1)\Delta$.

Otherwise, it follows because $p_{j_{k+1},\tau} \leq \Delta$ and $\tilde{d}_{j_{k+1}} \geq 2\Delta + r_{j_{k+1}} \geq (i+1)\Delta$ by Property 1 and $r_{j_{k+1}} \geq (i-1)\Delta$. All remaining jobs $j \in \{j_1, \ldots, j_k\}$ can be moved to a new machine M' of type τ using new starting times t'_j with $t'_j = t_j + \Delta$. Each job $j \in \{j_1, \ldots, j_k\}$ is then finished at time $t'_j + p_{j,\tau} = t_j + p_{j,\tau} + \Delta \leq \tilde{d}_j$. The last inequality holds since $\tilde{d}_j \geq 2\Delta + r_j \geq (i+1)\Delta \geq t_j + p_{j,\tau} + \Delta$, where we used Property 1, $r_j \geq (i-1)\Delta$ and in the last inequality the definition of k and the fact that $j \in \{j_1, \ldots, j_k\}$.

Hence, we obtain a feasible schedule in which no job $j \in J_M$ is started before $i\Delta$ and the cost at most triples since the two additional machines need not run longer than M. Applying the argument to all machines M concludes the proof. □

We now take a brief look at how jobs from J'_1 are scheduled and afterward formally prove the cost bound and feasibility in Lemma 8.14. For jobs scheduled in the i-th phase (i.e., released in the interval $[(i-1)\Delta, i\Delta)$), BatchedDispatch(ε) computes a tentative schedule at time $i\Delta$ using deadlines $\tilde{d}_j$ and with the additional restriction that no job is started before $i\Delta$. It then shifts the starting times of all jobs (as given by the tentative schedule) on machines of type τ by s_τ in order to be able to perform the respective setups before the jobs are processed. Roughly speaking, this leads to a feasible schedule since any job is delayed by at most s_B so that no deadline $d_j = \tilde{d}_j + s_B$ is violated. The costs are not increased compared to the bound given in Lemma 8.12 due to Lemma 8.13.

Lemma 8.14. *For jobs from J'_1,* BatchedDispatch(ε) *produces a feasible schedule with rental cost $O(c(\tau^+)/c(\tau^-)\varepsilon + 1/\varepsilon^2)$*Opt.

Proof. Let S^{BD} be the schedule produced by BatchedDispatch(ε) for J'_1 and let S^{NR} and S^R be the schedule given by the union of the tentative schedules for J'_1 if computed without and with the restrictions (of Step (2.1)), respectively. We first prove the bound on the cost of S^{BD} and afterwards its feasibility. As the cost of S^{BD} are not larger than the cost of S^R and by Lemma 8.12 S^{NR} has cost $O(c(\tau^+)/c(\tau^-)\varepsilon + 1/\varepsilon^2)$Opt, we only have to bound how much S^R and S^{NR} differ. Recall that we restrict the starting times of jobs from J'_1 scheduled during the i-th phase to be at least $i\Delta$. As by definition all jobs from J'_1 fulfill the requirements of Lemma 8.13, this restriction only increases the cost by a constant factor. Hence, the cost of S^R and S^{NR} differ by a constant factor so that S^R and thus also S^{BD} satisfy the claimed bound.

It remains to reason about the feasibility of S^{BD}. Fix an arbitrary phase i and define $N_i := \{j \in J'_1 : r_j \in [(i-1)\Delta, i\Delta)\}$ to be the set of jobs released during the i-th phase. Let S^R_i be the tentative schedule for N_i and let t^R_j denote the starting time of job $j \in N_i$ in S^R_i. First of all, the tentative schedule S^R_i provides a solution with $t^R_j \geq i\Delta$ and $t^R_j + p_{j,\tau} \leq \tilde{d}_j$ for all $j \in N_i$ processed on a machine of type τ. Since the starting times are then increased in S^{BD} to $t_j = t^R_j + s_\tau$, each job j processed on a machine of type τ is not started before $i\Delta + s_\tau$ and is finished by $t_j + p_{j,\tau} \leq t^R_j + s_\tau + p_{j,\tau} \leq \tilde{d}_j + s_\tau \leq d_j$. Therefore, by starting the setups at $i\Delta$ and applying the argument to all phases, S^{BD} is a feasible solution. □

Scheduling jobs from J'_2 is slightly more involved. The very rough idea for jobs from $J'_{2,1}$ is as follows: At the end of the i-th phase at time $i\Delta$ BatchedDispatch(ε) computes a tentative schedule for jobs released during this phase. The starting times on machines of type B are restricted to be at least $i\Delta$, which (exactly as in the previous lemma) allows us to easily setup machines of type B in time by delaying the starting times by s_B while increasing the cost only by some constant factor. The starting times on machines of type A in the tentative schedule need not be restricted because of the following insight: In case C holds, s_A is sufficiently small compared to s_B and the starting times of the tentative schedule can be increased (by $s_A + \Delta$) so that no job is started before $i\Delta$, the setups can be performed before the new starting times and no deadline $d_j = \tilde{d}_j + s_B$ is violated. In case C does not hold, a machine is opened precautionarily for each job upon its arrival. This ensures that the job can be started sufficiently early after computing the tentative schedule while not being too expensive (as the setup of a machine of type A has lower cost than processing a job from $J'_{2,1}$ on a machine of type B).

The idea for jobs from $J'_{2,2}$ is as follows: At the end of the i-th phase a tentative schedule is computed in which all jobs are processed on machines of type B. On the one hand, by this restriction it is (as in the previous descriptions) easy to guarantee feasibility. On the other hand, restricting ourselves to machines of type B cannot increase the cost too much because for all jobs it holds that processing it on a machine of type B incurs less cost than a setup of a machine of type A (by definition of $J'_{2,2}$) and due to one of the following observations, which show the setup cost to be not too high: On a machine of type A only a small constant number of jobs can be processed per machine (since $p_{j,A} \geq \Delta$ and $\tilde{\sigma}_{j,A} < 2\Delta$) while on a machine of type B sufficiently many jobs can be processed in case $\frac{s_A}{c} \leq \Delta$ (since together with the definition of $J'_{2,2}$ we have $p_{j,B} \leq \Delta$). Otherwise, $c < \frac{s_A}{\Delta}$ and $s_B < s_A + \Delta$ (since C does not hold if $J'_{2,2} \neq \emptyset$) and hence, machines of type B are not much more expensive and also the setup does not take much longer, so that in both cases the restriction to machines of type B does not increase the cost too much.

Lemma 8.15. *For jobs from J'_2,* BatchedDispatch(ε) *produces a feasible schedule with rental cost $O(c(\tau^+)/c(\tau^-)\varepsilon + 1/\varepsilon^3)$*Opt.

Proof. First of all, note that restricting the starting times of jobs from J'_2 released in the i-th phase to be at least $i\Delta$ if processed on a machine of type B, only increases the cost (compared to the schedule without this restriction) by a constant factor according to Lemma 8.13. By Lemma 8.12 we then would have a cost bound of $O(c(\tau^+)/c(\tau^-)\varepsilon + 1/\varepsilon^2)$Opt for the schedule of BatchedDispatch(ε) for jobs from $J'_{2,1}$ as well as from $J'_{2,2}$ if only this restriction was posed. Also, by the same line of arguments as in the previous lemma, the resulting schedule is feasible with respect to machines of type B. Hence, in the following we only have to analyze the influence of the further restrictions on the cost and prove the feasibility of the schedule with respect to machines of type A. We argue about $J'_{2,1}$ and $J'_{2,2}$ separately.

Scheduling jobs from $J'_{2,1}$. Fix an arbitrary phase i and let $N_i := \{j \in J'_{2,1} : r_j \in [(i-1)\Delta, i\Delta)\}$. Let S_i^R be the tentative schedule for N_i and let t_j^R denote

the starting time of job $j \in N_i$ in S_i^R. Let S_i^{BD} be the schedule produced by BatchedDispatch for N_i and t_j^{BD} be the starting time of job j in S_i^{BD}. Let S^R and S^{BD} be the union of the schedules S_i^R and S_i^{BD}, respectively. Let $N_i^A \subseteq N_i$ be the set of jobs processed on machines of type A in S_i^R. We distinguish two cases depending on whether C holds.

In case C holds, the cost bound of $O(c(\tau^+)/c(\tau^-)\varepsilon + 1/\varepsilon^2)\text{Opt}$ from Lemma 8.12 holds for S^{BD} as we do not have any restrictions besides on the starting times on machines of type B as already discussed at the beginning of the proof. Concerning the feasibility we can argue as follows. Consider any job $j \in N_i^A$ for some i. For the starting time of job j we have $t_j^{BD} = t_j^R + \Delta + s_A \geq i\Delta + s_A$. Hence, we are able to realize the respective schedule for jobs from N_i^A by starting the respective setup processes at time $i\Delta$. Also, all deadlines are met since $\Delta + s_A \leq s_B$ by the fact that C holds and thus, job j is finished by $t_j^{BD} + p_{j,A} \leq t_j^R + \Delta + s_A + p_{j,A} \leq \tilde{d}_j + \Delta + s_A \leq \tilde{d}_j + s_B \leq d_j$. Hence, S^{BD} is also feasible with respect to machines of type A.

In case C does not hold, recall that at each arrival of a job $j \in J'_{2,1}$, we open a new machine of type A at time r_j. This ensures that a machine is definitely available for the respective job $j \in N_i^A$ at time $r_j + s_A \leq t_j^{BD}$. Also, j is finished by $r_j + p_{j,A} + s_A \leq \tilde{d}_j + s_A \leq d_j$. Therefore, we obtain a feasible schedule and it only remains to prove the bound on the cost. Note that the additional setup cost in S_i^{BD} compared to S_i^R is upper bounded by $|J'_{2,1}| \cdot s_A$, which we can upper bound as follows. If a job $j \in J'_{2,1}$ is processed on a machine of type B in S_i^R it incurs cost of $cp_{j,B} \geq s_A$ by definition of $J'_{2,1}$. On the other hand, S_i^R can process at most three jobs on a machine of type A (since $\tilde{\sigma}_{j,A} = \tilde{d}_j - r_j - p_{j,A} < 2\Delta$ implying $\tilde{d}_j < 2\Delta + p_{j,A} + r_j$ and because $p_{j,A} > \Delta$ by the definition of J'_2). Taken together, this shows that the additional setup cost of $|J'_{2,1}| \cdot s_A$ is upper bounded by three times the cost of S_i^R. Therefore, BatchedDispatch(ε) produces a feasible schedule for jobs from $J'_{2,1}$ and the cost bound for S^R as discussed at the beginning of the proof holds for S^{BD} as well. This concludes the proof for jobs from $J'_{2,1}$.

Scheduling jobs from $J'_{2,2}$. Let $N_i \coloneqq \{j \in J'_{2,2} : r_j \in [(i-1)\Delta, i\Delta)\}$ and let S_i^{NR} be the tentative schedule for N_i if computed without the restriction of only using machines of type B. Let S_i^{BD} be the schedule produced by BatchedDispatch(ε) for jobs from N_i. Let S^{NR} and S^{BD} be the union of the schedules S_i^{NR} and S_i^{BD}, respectively. We have to analyze the influence of the restriction of only using machines of type B. To do so, we show the claim that shifting any jobs processed on machines of type A in S_i^{NR} (call them N_i^A) only increases the cost by $O(1/\varepsilon)$. Note that this is sufficient to prove the bound in the lemma as $N_i^A \cap \{j : \exists \tau \in \{A, B\}\ \tilde{\sigma}_{j,\tau} < 0\} = \emptyset$ and thus according to Lemma 8.12, all jobs from $\bigcup_i N_i^A$ can be scheduled with cost $O(1/\varepsilon^2)\text{Opt}$. Observe that $N_i^A \cap \{j : \exists \tau \in \{A, B\}\ \tilde{\sigma}_{j,\tau} < 0\} = \emptyset$ holds since we have $\tilde{\sigma}_{j,B} \geq 2\Delta$ for all jobs $j \in J'_2$ and thus, j can only be in $N_i^A \cap \{j : \exists \tau \in \{A, B\}\ \tilde{\sigma}_{j,\tau} < 0\}$ if $\tilde{\sigma}_{j,A} < 0$. However, in this case by the statement for jobs from $\bar{J}$ in Lemma 8.12 we can assume that $j \notin N_i^A$.

First, we show the claim if $\frac{s_A}{c} \leq \Delta$ holds. On the one hand, any machine of type A

in S_i^{NR} can process at most three jobs j with $j \in N_i^A$ (since $\tilde{\sigma}_{j,A} = \tilde{d}_j - r_j - p_{j,A} < 2\Delta$ implying $\tilde{d}_j < 2\Delta + p_{j,A} + r_j$ and since $p_{j,A} > \Delta$ by the definition of J_2'), leading to (setup) cost of $\Omega(|N_i^A|s_A)$. On the other hand, we can schedule all jobs from N_i^A on $O\left(\frac{|N_i^A|s_A}{\Delta c}\right)$ machines of type B each open for $O(s_B)$ time units and thus, with cost of $O\left(\frac{|N_i^A|s_A}{\Delta c} c s_B\right)$. To see why this is true, observe that for all jobs $j \in N_i^A$ it holds $[r_j, \tilde{d}_j] \supseteq [i\Delta, (i+1)\Delta) =: I_i$ since $r_j \leq i\Delta$ and $\tilde{\sigma}_{j,B} \geq 2\Delta$. Because all jobs $j \in N_i^A$ fulfill $p_{j,B} \leq \frac{s_A}{c}$ by definition of $J_{2,2}'$, we can accommodate $\lfloor \frac{c\Delta}{s_A} \rfloor \geq 1$ many jobs from N_i^A in I_i. Hence, by only using machines of type B we lose a factor of $O(\frac{cs_B}{\Delta c}) = O(1/\varepsilon)$, proving the claim.

Finally, it needs to be proven that the claim also holds for the case $\frac{s_A}{c} > \Delta$. Recall that $J_{2,2}' \neq \emptyset$ only if C does not hold and hence, that this case also implies $s_B < s_A + \Delta$. Let $M_1, \ldots, M_m$ denote the machines of type A used by S_i^{NR} and let $\kappa_1, \ldots, \kappa_m$ be the durations for which they are open. Since $p_{j,B} \leq p_{j,A}$ for all $j \in J_{2,2}'$ by definition of J_2', we can replace each machine $M \in \{M_1, \ldots, M_m\}$ by a machine M' of type B using the same schedule on M' as on M. We increase the cost by a factor of at most $\frac{\sum_{i=1}^m c(s_B+\kappa_i)}{\sum_{i=1}^m (s_A+\kappa_i)} \leq \frac{cs_B}{s_A} + c \leq \frac{c(\Delta+s_A)}{s_A} + c \leq \frac{c\Delta}{s_A} + 2c = O(\frac{s_A}{\Delta}) = O(1/\varepsilon)$, where the second last bound holds due to $\frac{s_A}{c} > \Delta$. This proves the claim and concludes the proof. □

Lemma 8.16. *For jobs from J_3',* BatchedDispatch(ε) *produces a feasible schedule with rental cost $O(c(\tau^+)/c(\tau^-)\varepsilon + 1/\varepsilon^3)$Opt if $\tau^+ = B$ and $O(c(\tau^+)/c(\tau^-)\varepsilon^2 + 1/\varepsilon^3)$Opt otherwise.*

Proof. If $c \geq 1$ (i.e., $\tau^+ = B$) then all jobs from J_3' can be assumed to be scheduled on machines of type A without any loss in the cost since machines of type B are at least as expensive as machines of type A and all jobs from J_3' are smaller on machines of type A. Hence by Lemma 8.13 we obtain a bound on the cost as given in Lemma 8.12.

Consider the case $c < 1$. Recall that we schedule all jobs on machines of type A. By the same line of arguments as given in the previous lemma for jobs from $J_{2,2}'$ and switching the roles of machines of type A and type B, we get the desired bound (as in the case $\frac{s_A}{c} > \Delta$ of Lemma 8.15, we lose a factor of at most $\frac{\sum_{i=1}^m (s_A+\kappa_i)}{\sum_{i=1}^m c(s_B+\kappa_i)} = O\left(\frac{1}{c}\right) = O\left(\frac{c(\tau^+)}{c(\tau^-)}\right)$ compared to the schedule with cost $O\left(\frac{1}{\varepsilon^2}\right)$ from Lemma 8.12). □

Taking Lemmas 8.14 to 8.16 together, we obtain the following theorem.

Theorem 8.17. *Let $\beta = (1+\varepsilon)s_B$, $1/s_B \leq \varepsilon \leq 1$. For $c \geq 1$* BatchedDispatch(ε) *is $O(c/\varepsilon + 1/\varepsilon^3)$-competitive. For $c < 1$* BatchedDispatch(ε) *is $O(1/c\varepsilon^2 + 1/\varepsilon^3)$-competitive.*

In a parameterized analysis one usually does not want the algorithm to depend on the knowledge of the value of the parameter but only wants to refer to the parameter

Algorithm 8 Description of the algorithm BatchedDispatch.

Let $\hat{\varepsilon} = 1/2$.

(1) Start a new instance BatchedDispatch($\hat{\varepsilon}$) handling arriving jobs **until** the arrival of a job j with $\hat{\varepsilon} > \max_\tau \sigma_{j,\tau} =: \sigma$. Then,
- reset $\hat{\varepsilon} = \min\{\hat{\varepsilon}/2, \sigma\}$ and
- go back to Step 1.

in the analysis and resulting performance bound. Therefore, we do not want to assume the minimum slack β to be known in advance and we show how one can easily get rid of this assumption. The formal description is given in Algorithm 8. Essentially, the algorithm maintains a guess on ε and whenever this guess turns out to be too large, it is halved. This approach does not substantially increase the cost compared to the case where ε is known in advance as proven in the next theorem.

Theorem 8.18. BatchedDispatch *achieves the same bounds as given in Theorem 8.17 for* BatchedDispatch*(*ε*)* .

Proof. Denote the parameters $\hat{\varepsilon}$ used in the calls of BatchedDispatch($\hat{\varepsilon}$) by $\hat{\varepsilon}_1, \hat{\varepsilon}_2, \ldots, \hat{\varepsilon}_k$ (in this order). Let ε be the actual value describing the slack of the considered instance. We have $2^{k-1}\hat{\varepsilon}_k \leq \hat{\varepsilon}_{k-1}/2 \leq \ldots 2\hat{\varepsilon}_2 \leq \hat{\varepsilon}_1 = 1/2$. Also we have that $\hat{\varepsilon}_k \geq \varepsilon/2$. Hence, $\hat{\varepsilon}_i \geq 2^{k-i-1}\varepsilon$. By Theorem 8.17 BatchedDispatch($\hat{\varepsilon}_i$) has cost $O\left(\left(\frac{c}{\hat{\varepsilon}_i} + \frac{1}{\hat{\varepsilon}_i^3}\right) \textsc{Opt}\right)$ for the case $c \geq 1$. Hence, the overall costs of BatchedDispatch for this case are upper bounded by

$$\begin{aligned} O(\textsc{Opt}) \cdot \sum_{i=1}^{k} \left(\frac{c}{\hat{\varepsilon}_i} + \frac{1}{\hat{\varepsilon}_i^3}\right) &= O(\textsc{Opt}) \left(\frac{c}{\varepsilon} \sum_{i=1}^{k} \frac{1}{2^{k-i-1}} + \frac{1}{\varepsilon^3} \sum_{i=1}^{k} \frac{1}{2^{3(k-i-1)}}\right) \\ &= O(\textsc{Opt}) \left(\frac{4c}{\varepsilon} + \frac{10}{\varepsilon^3}\right) . \end{aligned}$$

The case $c < 1$ follows analogously. □

We want to conclude this chapter by noting that neither in our algorithm design nor in our analysis we tried to optimize any constants involved in the achieved competitiveness. Also, computed solutions will, in general, rent a very high number of machines and for quite short periods of time, which is rather inefficient. In actual implementations one would, of course, add improvements such as keeping a machine open if it is clear that it can, instead of being closed, be reused for the schedule of a later phase.

CHAPTER 9

Conclusion & Outlook

Scheduling with setup times is a large research area considering models with many natural applications. However, as discussed in the introduction of this thesis, Allahverdi et al. highlighted in their survey [AGA99] that especially theoretical results based on worst-case analysis are very scarce in this domain. In the light of this observation, the goal of this thesis was to contribute to the research area on scheduling with setup times from the perspective of approximation and competitive online algorithms. In the course of this work, we have considered two different questions. On the one hand, the question how we can reasonably schedule jobs of different classes when a switch from processing jobs of one class to jobs of a different class requires reconfigurations with non-negligible setup times. On the other hand, we considered the allocation of and scheduling on machines rented from the cloud when a (virtual) machine incurs a non-ignorable starting time before being ready for processing. In the following, we discuss some open questions that directly result from our findings as well as some possible future research directions.

9.1 Scheduling on Machines from the Cloud

In Chapter 8, we considered the problem in which jobs, which arrive over time, are to be scheduled on machines that need to be rented from the cloud. The goal is to minimize the rental cost while meeting all jobs' deadlines. We were able to achieve $O(c/\varepsilon + 1/\varepsilon^3)$-competitive and $O(1/c\varepsilon^2 + 1/\varepsilon^3)$-competitive solutions for $c \geq 1$ and $c < 1$, respectively, when the smallest slack is at least $(1 + \varepsilon)s_B$.

Embedded in the research context, these results can be seen to complement and extend a variant of the problem of scheduling in the cloud [Aza+13] as discussed in Section 8.2. Recall that in this work, the authors study a problem addressing the minimization of cost and maximum delay in a model where identical machines with setup times of length s need to be rented from the cloud. Precisely, they define

the delay of a job j as the difference between its finishing time and $r_j + p_j$. They designed an online algorithm that, given a budget of $(1 + \varepsilon)$ times the minimum cost to schedule all jobs (without any assumption on the maximum delay), finds a solution with a maximum delay of $O(s/\varepsilon)$. It seems to be a natural question to study the problem when the roles of costs and delay are interchanged in the sense that the maximum delay becomes the restriction (instead of the optimization goal) and the cost becomes the optimization goal (instead of the restriction). Observe that in this case, the delay and the slack of a job essentially describe the same aspect: The maximum amount of time by which the processing of a job might be delayed after its release. Hence, if the maximum delay should be bounded by d, we can set the maximum slack of each job j to d (by defining $d_j = r_j + \min_\tau\{p_{j,\tau}\} + d$). For only one type of machines, our algorithm then achieves a solution with maximum delay of $(1 + \varepsilon)s$ and costs $O(1/\varepsilon^2 \textsc{Opt})$, giving a solution for the problem of [Aza+13] when the maximum delay is considered as a restriction and the cost as an optimization goal. In the same sense, one could extend this to two types of machines, in which case our algorithm provides solutions with cost $O((c/\varepsilon + 1/\varepsilon^3)\textsc{Opt})$ and $O((1/c\varepsilon^2 + 1/\varepsilon^3)\textsc{Opt})$ for $c \geq 1$ and $c < 1$, respectively.

In the light of these considerations, our results are in line with other research (e.g. [Aza+13; Sah13]) and in this regard it is a first step toward models for scheduling machines from the cloud addressing the heterogeneity of machines. Although the considered setting with $\kappa = 2$ types of machines seems to be restrictive, it turned out to be challenging and there is still a gap between our lower and upper bound. For future work, however, it would be interesting to study more general models for $\kappa > 2$ types of machines. As our approach is partly based on manually designed decisions concerning which type of machine to use for jobs with certain characteristics (see Section 8.5.3), algorithms for $\kappa > 2$ probably require new techniques. On the one hand, while it might be possible to design decision rules for deciding which type of machine to use for certain jobs, an increasing number of types might let the number of decision rules grow even more rapidly. On the other hand, it is not clear how to automate such decisions. Indeed it is possible to extend the linear program to a larger number of machine types, but this will only work for jobs having a large slack on all types (as it was the case for the job set J_1' in Section 8.5.3). For other jobs it would be necessary to restrict the machine types that might process a job in advance since otherwise it cannot be guaranteed that tentative schedules can be turned into an actual feasible one.

9.2 Scheduling with Setup Times

For the model introduced in Chapter 3 and then considered in Chapters 4 to 7, we have shown several positive results in terms of small constant factor approximations concerning the makespan for the classical case of identical, uniformly related and special cases of unrelated machines. For the general case of unrelated machines we have shown matching upper and lower bounds, which are logarithmic in the

number n of jobs and number m of machines. For minimizing the maximum flow time on a single machine, we have shown a constant factor approximation and have analyzed a very simple and intuitive online algorithm, which turned out to be $\Theta(\sqrt{n})$-competitive. In a smoothed competitive analysis, we have shown that this competitiveness improves to be polylogarithmic in n for uniform and gaussian noise.

In the following we name and discuss some questions that result from our findings and that might be interesting for future work.

Makespan on Heterogeneous Machines. Our results show a clear separation between the classical two models for heterogeneous machines: While the case of uniformly related machines can be approximated well, our matching upper and lower bounds for unrelated machines are unsatisfactorily high. Therefore, it might be interesting to seek for positive results for relevant special cases of unrelated machines, that is, cases lying in between the two aforementioned models for heterogeneity. The negative results are, unarguable, due to the mere generality of the considered model and the resulting relation to the SetCover problem. However, there might be practically interesting special cases where much better approximation factors can be achieved. One reasonable direction could be to consider unrelated machines of few types. Here, all machines of a given type are identical, while machines of different types are unrelated (similar to the cloud setting considered in Chapter 8). Such settings are, for example, motivated by heterogeneous systems comprising of a few different processing units such as CPUs, GPUs and FPGAs. In case without setup times, this case is already much easier as there is an EPTAS known if the number of types is constant [JM17] (compared to the $3/2$ lower bound for the general problem [LST90]).

A second case that might admit constant factor approximations are problems with processing set restrictions [LL08; LL16], a special case of unrelated machines. Note that the case of identical machines with processing set restrictions is equivalent to the restricted assignment problem. Here, it is known for the classical variant (without setup times) that certain structural properties concerning the set of eligible machines of jobs can lead to much better results than those possible for the general case. As an example take the case where the set of eligible machines is hierarchical, that is, there is an ordering of machines such that any M_j is a suffix of it. Such a model might have practical motivations, for example, in settings where restrictions result from lack of memory or equipment with specific components. Compared to the impossibility result of $3/2$ for the general restricted assignment problem, [EL11] designs a PTAS for the case of hierarchical restrictions. It is not too hard to see that the case of hierarchical restrictions also allows better approximation factors for our problem of restricted assignment with setup times. Compared to the $o(\log n + \log m)$ impossibility result for the general restricted assignment problem with setup times, first approaches already show that constant factor approximations are possible for hierarchical restrictions using an approach similar to Section 5.4.

Minimizing Maximum Flow Time. When it comes to minimizing the maximum flow time on a single machine, the exact approximability of the problem is still open. While we improved the approximation factor to a smaller constant, an impossibility result is only known in terms of the NP-hardness for computing optimal solutions [DS11]. Therefore, the question of better approximations or stronger lower bounds arises.

Concerning the (non-clairvoyant) online variant, the capabilities of online algorithms, which are not restricted to be greedy-like as in Chapter 7, need to be further investigated. It is not clear whether or not a constant competitiveness is possible. Our lower bound only holds for greedy-like algorithms and does not carry over to generally unrestricted online algorithms. A potential non-constant lower bound construction would have to respect two aspects: First, the non-clairvoyance of an online algorithm would have to be exploited explicitly (which our construction does not make use of at all) as for clairvoyant approaches a constant competitiveness is achievable [DS11]. Second, as a potential lower bound would also have to hold for our algorithm, the optimal flow time would have to be in $o(\sqrt{n})$ and the difference in the number of setups our online algorithm performs compared to the optimal solution has to be in $\omega(1)$. Taken together, an adversary would have to exploit the non-clairvoyance to let an online algorithm perform unnecessary setups. While it is not clear how this could be achieved, we do not know how to handle the non-clairvoyance algorithmically to get positive results either. One important ingredient that is used in [DS11] to obtain a constant competitive algorithm for the clairvoyant case is, roughly speaking, based on the fact that one can collect jobs of the same class until their workload reaches a threshold that justifies a new setup for the respective class. Such an approach is no longer possible in the non-clairvoyant case.

It would be very interesting for future work to further investigate if it is possible to improve our smoothed analysis. Although we were not able to show such a result, it is possible that the actual smoothed competitiveness is independent of n. Though, proving such a result would probably require a different approach; the $\log n$ term in our result seems to be inherent to our analysis as it relies on the length of the longest run of bad events, each occurring with constant probability. On the other hand, it is also evident that our $\Omega(\sqrt{n})$ lower bound for greedy-like algorithms is robust against very small amounts of random noise. For noise from an interval $[a, b]$ with $a, b \in O(\frac{1}{\sqrt{n}})$, a small adaptation of the construction in Theorem 7.12 proves this fact: Increasing the setup time of each class from 1 to 2 and having 5 instead of 2 time steps without a job release at the end of each phase. By this, an optimal solution will remain to have a constant flow time as no phases interfere (even if all job sizes are increased due to the random noise); the online solution will still increase the flow time in each phase as (even if all job sizes are decreased due to the random noise) the workload processed by the online algorithm is by at least 1 larger than the length of the phase. Similarly, one should be able to obtain analogous results for gaussian noise. A rough back-of-the-envelope calculation already shows

that for $\sigma \leq \frac{1}{\sqrt{n \log n}}$ we obtain, with constant probability, the same situation as described above where all $X_j \in [a, b]$ with $a, b \in O(\frac{1}{\sqrt{n}})$ (which follows from the fact that $\Phi^{-1}(1 - \frac{1}{n}) \approx \sqrt{2 \log n}$ and if $X_j \sim \mathcal{N}(0, \sigma^2)$ then $X_j \in [-\sigma z, \sigma z]$ with probability p for $z = \Phi^{-1}(\frac{p+1}{2})$ [Das08]).

An extension to parallel machines could be another next step. However, compared to the case where we do not have setup times and where the FIFO strategy was carried over to parallel machines [Mas04], setups make the problem more challenging in two ways: We not only have to take a look at the increase in the number of setups an online or approximation algorithm performs compared to an optimal solution due to the exact way how jobs are batched per machine; we also need to take into account the influence of the distribution of jobs among several machines. Intuitively, decreasing the number of machines used for a class also decreases the number of setups but, on the other hand, it decreases the potential of parallelizing jobs of a class. While this was already the case for the makespan objective, it becomes even more challenging here because a class will usually have several (instead of at most one) setups per machine. It is not obvious and seems to be non-trivial how to tackle this aspect of the parallel machines case, that is, how to keep the number of setups small and how to bound this.

9.3 Future Directions

On a more general level, that is, not directly linked to our particular results, we will discuss three future directions.

Smoothed Competitive Analysis. An interesting research direction seems to be the application of the notion of smoothed competitiveness as one approach in the realm of beyond worst-case analysis [Rou19]. Although this kind of analysis looks quite demanding, it can provide meaningful bounds carrying much more information about the performance of an algorithm than a usual competitive analysis. Also, an interplay of algorithm design and smoothed competitive analysis might have a special charm as it might lead to practically good and, at the same time, simpler and more intuitive algorithms compared to algorithms tailored to the worst case.

On the other hand, one should also be aware that smoothed competitive analysis is not a universal tool that is simply applicable to any problem to mitigate worst cases and to get considerably improved bounds. The question of whether random noise can lead to a benefit highly depends on the studied problem. For example, we have seen some positive result for smoothed competitiveness in Chapter 7. However, in the course of the development of this thesis, we also encountered problems where worst cases are more robust against random noise and small perturbations as in Chapter 7 seem to have no significant impact on the achievable competitiveness. One such problem is a variant of the problem from Chapter 7 in which the objective is the minimization of the *average* flow time. This problem has, even in the clairvoyant case without setups, a lower bound of $\Omega(n)$ for deterministic and $\Omega(\sqrt{n})$ [SV02]

for randomized online algorithms, which seem to be robust against small random perturbations. A similar observation seems to hold true for the lower bound for the problem of machine minimization [Sah13], which we mentioned in the discussion of related work in Section 8.2.

Sequence-Dependent Setups. With respect to the modeling of scheduling with setup times, it might be appealing to consider generalizations in terms of sequence-dependent setup times [Bru07]. In this case, setup times (related to individual jobs, or related to classes) might not only depend on what is executed next on a machine but on what was executed before as well. Such models have natural applications as reported in [AGA99; All+08; All15]. For example, setup times might represent the cleaning of a machine, which can depend on the sequence of operations. For instance, when cleaning a printing machine between the usage of different colors (white after red requires more cleaning time than black after white); or setup times might represent the change of tools where the question of which tools are to be removed or added depends on past *and* future. However, like in the case of sequence-independent setups, sequence-dependencies have been considered in the literature but hardly from the perspective of worst-case analysis. The challenge of having sequence-dependent setup times lies in the following observation, which indicates that minimizing the makespan becomes much harder even on a single machine. Recall that in the model considered in this thesis, minimizing the makespan on a single machine is trivial as there is exactly one batch per class and batches are ordered arbitrarily. In contrast, handling sequence-dependencies implies solving a travelling salesperson problem (even if all jobs have size 0 and under the assumption that we introduce a dummy class for which a final setup has to be performed).

Composed Jobs. Finally, another interesting direction could be the study of models that consider composed jobs. Motivated by, for example, the SFB 901 [Hap+13], or in general micro-service based architectures, one could consider settings where a job consists of several operations. Then the completion time of a job is determined by the completion time of its latest operation and a natural objective is the minimization of the average completion time of jobs. Also, setup times might be associated with classes of operations. Such a model has also been motivated in further possible domains [LLP05] but is still barely understood. First steps included studying the complexity of the problem and the NP-hardness of the single machine case [NCY02] as well as the polynomial solvability of a special case [Ger+99] are known. Intuitively and compared to the makespan objective, this problem is no longer trivial even on a single machine mainly because the order in which classes are processed influences the total completion time. We initiated research on approximation algorithms for this problem and considered the case of a single machine in [MMP19]. Based on solving a simplified variant and transforming solutions back to the original problem, we were able to design $(1 + \sqrt{2})$- and $2(1 + \sqrt{2})$-approximation algorithms for a constant and non-constant number of classes, respectively.

Bibliography

[AG07] Barbara M. Anthony and Anupam Gupta. “Infrastructure Leasing Problems”. In: *Proceedings of the 12th International Conference on Integer Programming and Combinatorial Optimization (IPCO)*. Vol. 4513. Lecture Notes in Computer Science. Springer, 2007, pp. 424–438.

[AGA99] Ali Allahverdi, Jatinder N.D. Gupta, and Tariq Aldowaisan. “A review of scheduling research involving setup considerations”. In: *Omega* vol. 27, no. 2 (1999), pp. 219–239.

[Alb03] Susanne Albers. “Online algorithms: a survey”. In: *Mathematical Programming* vol. 97, no. 1-2 (2003), pp. 3–26.

[All+08] 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* vol. 187, no. 3 (2008), pp. 985–1032.

[All15] Ali Allahverdi. “The third comprehensive survey on scheduling problems with setup times/costs”. In: *European Journal of Operational Research* vol. 246, no. 2 (2015), pp. 345–378.

[Alo+98] Noga Alon, Yossi Azar, Gerhard J. Woeginger, and Tal Yadid. “Approximation schemes for scheduling on parallel machines”. In: *Journal of Scheduling* vol. 1, no. 1 (1998), pp. 55–66.

[Ama19] Amazon EC2. `https://aws.amazon.com/de/ec2/`. Accessed: July 1, 2019.

[AMM14] Sebastian Abshoff, Christine Markarian, and Friedhelm Meyer auf der Heide. “Randomized Online Algorithms for Set Cover Leasing Problems”. In: *Proceedings of the 8th International Conference on Combinatorial Optimization and Applications (COCOA)*. Vol. 8881. Lecture Notes in Computer Science. Springer, 2014, pp. 25–34.

[AMS06] Noga Alon, Dana Moshkovitz, and Shmuel Safra. “Algorithmic construction of sets for k-restrictions”. In: *ACM Transactions on Algorithms* vol. 2, no. 2 (2006), pp. 153–177.

[Ana+17] S. Anand, Karl Bringmann, Tobias Friedrich, Naveen Garg, and Amit Kumar. “Minimizing Maximum (Weighted) Flow-Time on Related and Unrelated Machines”. In: *Algorithmica* vol. 77, no. 2 (2017), pp. 515–536.

[AS08] Ali Allahverdi and H. M. Soroush. "The significance of reducing setup times/setup costs". In: *European Journal of Operational Research* vol. 187, no. 3 (2008), pp. 978–984.

[Aza+13] Yossi Azar, Naama Ben-Aroya, Nikhil R. Devanur, and Navendu Jain. "Cloud Scheduling with Setup Cost". In: *Proceedings of the 25th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. ACM, 2013, pp. 298–304.

[AZM18] Massinissa Ait Aba, Lilia Zaourar, and Alix Munier. "Approximation Algorithm for Scheduling Applications on Hybrid Multi-core Machines with Communications Delays". In: *Proceedings of the 2018 IEEE International Parallel and Distributed Processing Symposium Workshops, (IPDPS Workshops)*. 2018, pp. 36–45.

[BC16] Nikhil Bansal and Bouke Cloostermans. "Minimizing Maximum Flow-Time on Related Machines". In: *Theory of Computing* vol. 12, no. 1 (2016), pp. 1–14.

[BCM98] Michael A. Bender, Soumen Chakrabarti, and S. Muthukrishnan. "Flow and Stretch Metrics for Scheduling Continuous Job Streams". In: *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 1998, pp. 270–279.

[BD78] John L. Bruno and Peter J. Downey. "Complexity of Task Sequencing with Deadlines, Set-Up Times and Changeover Costs". In: *SIAM Journal on Computing* vol. 7, no. 4 (1978), pp. 393–404.

[BE05] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 2005.

[Bec+06] Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, Guido Schäfer, and Tjark Vredeveld. "Average-Case and Smoothed Competitive Analysis of the Multilevel Feedback Algorithm". In: *Mathematics of Operations Research* vol. 31, no. 1 (2006), pp. 85–108.

[Ben+13] Michael A. Bender, David P. Bunde, Vitus J. Leung, Samuel McCauley, and Cynthia A. Phillips. "Efficient scheduling to minimize calibrations". In: *Proceedings of the 25th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. ACM, 2013, pp. 280–287.

[BJS74] John L. Bruno, Edward G. Coffman Jr., and Ravi Sethi. "Scheduling Independent Tasks to Reduce Mean Finishing Time". In: *Commununi-cations of the ACM* vol. 17, no. 7 (1974), pp. 382–387.

[BNR03] Allan Borodin, Morten N. Nielsen, and Charles Rackoff. "(Incremental) Priority Algorithms". In: *Algorithmica* vol. 37, no. 4 (2003), pp. 295–326.

[BPS17] Allan Borodin, Denis Pankratov, and Amirali Salehi-Abari. “On Conceptually Simple Algorithms for Variants of Online Bipartite Matching”. In: *Revised Selected Papers of the 15th International Workshop on Approximation and Online Algorithms (WAOA)*. Vol. 10787. Lecture Notes in Computer Science. Springer, 2017, pp. 253–268.

[Bru+14] Tobias Brunsch, Heiko Röglin, Cyriel Rutten, and Tjark Vredeveld. “Smoothed performance guarantees for local search”. In: *Mathematical Programming* vol. 146, no. 1-2 (2014), pp. 185–218.

[Bru07] Peter Brucker. *Scheduling algorithms*. Vol. 3. Springer, 2007.

[Bur14] John Burkardt. “The truncated normal distribution”. In: *Department of Scientific Computing Website, Florida State University* (2014).

[Che+19] Lin Chen, Minming Li, Guohui Lin, and Kai Wang. “Approximation of Scheduling with Calibrations on Multiple Machines (Brief Announcement)”. In: *Proceedings of the 31st ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. ACM, 2019, pp. 237–239.

[Che93] Bo Chen. “A Better Heuristic for Preemptive Parallel Machine Scheduling with Batch Setup Times”. In: *SIAM Journal on Computing* vol. 22, no. 6 (1993), pp. 1303–1318.

[Chu+04] Julia Chuzhoy, Sudipto Guha, Sanjeev Khanna, and Joseph Naor. “Machine Minimization for Scheduling Jobs with Interval Constraints”. In: *Proceedings of the 45th Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society, 2004, pp. 81–90.

[CMV17] Louis-Claude Canon, Loris Marchal, and Frédéric Vivien. “Low-Cost Approximation Algorithms for Scheduling Independent Tasks on Hybrid Platforms”. In: *Proceedings of the 23rd International Conference on Parallel and Distributed Computing (Euro-Par)*. Vol. 10417. Lecture Notes in Computer Science. Springer, 2017, pp. 232–244.

[Cor+15] 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: *Mathematical Programming* vol. 154, no. 1-2 (2015), pp. 305–328.

[CV97] Bo Chen and Arjen P. A. Vestjens. “Scheduling on identical machines: How good is LPT in an on-line setting?” In: *Operations Research Letters* vol. 21, no. 4 (1997), pp. 165–169.

[CVV16] José R. Correa, Victor Verdugo, and José Verschae. “Splitting versus setup trade-offs for scheduling to minimize weighted completion time”. In: *Operations Research Letters* vol. 44, no. 4 (2016), pp. 469–473.

[CYZ13] Lin Chen, Deshi Ye, and Guochuan Zhang. “Online Scheduling on a CPU-GPU Cluster”. In: *Proceedings of the 10th International Conference on Theory and Applications of Models of Computation (TAMC)*. Vol. 7876. Lecture Notes in Computer Science. Springer, 2013, pp. 1–9.

[Das08] Anirban DasGupta. *Asymptotic theory of statistics and probability*. Springer Science & Business Media, 2008.

[DE92] V. R. Dondeti and Hamilton Emmons. "Fixed Job Scheduling with Two Types of Processors". In: *Operations Research* vol. 40, no. Supplement-1 (1992), S76–S85.

[Dev+14] Nikhil R. Devanur, Konstantin Makarychev, Debmalya Panigrahi, and Grigory Yaroslavtsev. "Online Algorithms for Machine Minimization". In: *CoRR* vol. abs/1403.0486 (2014). URL: `http://arxiv.org/abs/1403.0486`.

[DJ19] Max A. Deppert and Klaus Jansen. "Near-Linear Approximation Algorithms for Scheduling Problems with Batch Setup Times". In: *Proceedings of the 31st ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. ACM, 2019, pp. 155–164.

[DS00] Srikrishna Divakaran and Michael Saks. *An online scheduling problem with job set-ups*. Tech. rep. DIMACS Technical Report, 2000.

[DS08] Srikrishnan Divakaran and Michael E. Saks. "Approximation algorithms for problems in scheduling with set-ups". In: *Discrete Applied Mathematics* vol. 156, no. 5 (2008), pp. 719–729.

[DS11] Srikrishnan Divakaran and Michael E. Saks. "An Online Algorithm for a Problem in Scheduling with Set-ups and Release Times". In: *Algorithmica* vol. 60, no. 2 (2011), pp. 301–315.

[EKS14] Tomás Ebenlendr, Marek Krcál, and Jirí Sgall. "Graph Balancing: A Special Case of Scheduling Unrelated Parallel Machines". In: *Algorithmica* vol. 68, no. 1 (2014), pp. 62–80.

[EL11] Leah Epstein and Asaf Levin. "Scheduling with processing set restrictions: PTAS results for several variants". In: *International Journal of Production Economics* vol. 133, no. 2 (2011), pp. 586–595.

[ERV14] Matthias Englert, Heiko Röglin, and Berthold Vöcking. "Worst Case and Probabilistic Analysis of the 2-Opt Algorithm for the TSP". In: *Algorithmica* vol. 68, no. 1 (2014), pp. 190–264.

[Fel66] William Feller. *An introduction to probability theory and its applications*. Vol. 2. John Wiley & Sons, Inc., 1966.

[FS15] Jeremy T. Fineman and Brendan Sheridan. "Scheduling Non-Unit Jobs to Minimize Calibrations". In: *Proceedings of the 27th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. ACM, 2015, pp. 161–170.

[Ger+99] Alex E. Gerodimos, Celia A. Glass, Chris N. Potts, and Thomas Tautenhahn. "Scheduling multi-operation jobs on a single machine". In: *Annals OR* vol. 92 (1999), pp. 87–105.

[GG64] Paul C Gilmore and Ralph E Gomory. “Sequencing a one state-variable machine: A solvable case of the traveling salesman problem”. In: *Operations research* vol. 12, no. 5 (1964), pp. 655–679.

[GLS81] Martin Grötschel, László Lovász, and Alexander Schrijver. “The ellipsoid method and its consequences in combinatorial optimization”. In: *Combinatorica* vol. 1, no. 2 (1981), pp. 169–197.

[GM12] Ravindra Gokhale and M Mathirajan. “Scheduling identical parallel machines with machine eligibility restrictions to minimize total weighted flowtime in automobile gear manufacturing”. In: *The International Journal of Advanced Manufacturing Technology* vol. 60, no. 9-12 (2012), pp. 1099–1110.

[GMW07] Martin Gairing, Burkhard Monien, and Andreas Woclaw. “A faster combinatorial approximation algorithm for scheduling unrelated parallel machines”. In: *Theoretical Computer Science* vol. 380, no. 1-2 (2007), pp. 87–99.

[Gol08] Oded Goldreich. “Computational complexity: a conceptual perspective”. In: *ACM Sigact News* vol. 39, no. 3 (2008), pp. 35–39.

[Goo19] Google Cloud. `https://cloud.google.com/`. Accessed: July 1, 2019.

[Gra69] Ronald L. Graham. “Bounds on Multiprocessing Timing Anomalies”. In: *SIAM Journal of Applied Mathematics* vol. 17, no. 2 (1969), pp. 416–429.

[Gup+12] Anupam Gupta, Sungjin Im, Ravishankar Krishnaswamy, Benjamin Moseley, and Kirk Pruhs. “Scheduling heterogeneous processors isn’t as easy as you think”. In: *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2012, pp. 1242–1253.

[Hal+97] Leslie A. Hall, Andreas S. Schulz, David B. Shmoys, and Joel Wein. “Scheduling to Minimize Average Completion Time: Off-Line and On-Line Approximation Algorithms”. In: *Mathematics of Operations Research* vol. 22, no. 3 (1997), pp. 513–544.

[Hap+13] Markus Happe, Friedhelm Meyer auf der Heide, Peter Kling, Marco Platzner, and Christian Plessl. “On-the-fly computing: A novel paradigm for individualized it services”. In: *16th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC)*. IEEE. 2013, pp. 1–10.

[HR87] Micha Hofri and Keith W. Ross. “On the Optimal Control of Two Queues with Server Setup Times and its Analysis”. In: *SIAM Journal on Computing* vol. 16, no. 2 (1987), pp. 399–420.

[HS76] Ellis Horowitz and Sartaj Sahni. “Exact and Approximate Algorithms for Scheduling Nonidentical Processors”. In: *Journal of the ACM* vol. 23, no. 2 (1976), pp. 317–327.

[HS87] Dorit S. Hochbaum and David B. Shmoys. “Using dual approximation algorithms for scheduling problems theoretical and practical results”. In: *Journal of the ACM* vol. 34, no. 1 (1987), pp. 144–162.

[HS88] Dorit S. Hochbaum and David B. Shmoys. “A Polynomial Approximation Scheme for Scheduling on Uniform Processors: Using the Dual Approximation Approach”. In: *SIAM Journal on Computing* vol. 17, no. 3 (1988), pp. 539–551.

[HV12] Benjamin Hiller and Tjark Vredeveld. “Probabilistic alternatives for competitive analysis”. In: *Computer Science - R&D* vol. 27, no. 3 (2012), pp. 189–196.

[Im+17] Sungjin Im, Benjamin Moseley, Kirk Pruhs, and Clifford Stein. “Minimizing Maximum Flow Time on Related Machines via Dynamic Posted Pricing”. In: *25th Annual European Symposium on Algorithms (ESA)*. Vol. 87. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017, 51:1–51:10.

[Jan+19] Klaus Jansen, Kim-Manuel Klein, Marten Maack, and Malin Rau. “Empowering the Configuration-IP - New PTAS Results for Scheduling with Setups Times”. In: *Proceedings of the 10th Innovations in Theoretical Computer Science Conference (ITCS)*. Vol. 124. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019, 44:1–44:19.

[Jan10] Klaus Jansen. “An EPTAS for Scheduling Jobs on Uniform Processors: Using an MILP Relaxation with a Constant Number of Integral Variables”. In: *SIAM Journal on Discrete Mathematics* vol. 24, no. 2 (2010), pp. 457–485.

[JL16] Klaus Jansen and Felix Land. “Non-preemptive Scheduling with Setup Times: A PTAS”. In: *Proceedings of the 22nd International Conference on Parallel and Distributed Computing (Euro-Par)*. Vol. 9833. Lecture Notes in Computer Science. Springer, 2016, pp. 159–170.

[JM17] Klaus Jansen and Marten Maack. “An EPTAS for Scheduling on Unrelated Machines of Few Different Types”. In: *Proceedings of the 15th International Symposium on Algorithms and Data Structures (WADS)*. Vol. 10389. Lecture Notes in Computer Science. Springer, 2017, pp. 497–508.

[JMM19] Klaus Jansen, Marten Maack, and Alexander Mäcker. “Scheduling on (Un-)Related Machines with Setup Times”. In: *Proceedings of the 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE Computer Society, 2019, pp. 145–154.

[Joh54] Selmer Martin Johnson. “Optimal two-and three-stage production schedules with setup times included”. In: *Naval Research Logistics (NRL)* vol. 1, no. 1 (1954), pp. 61–68.

[JR17a] Klaus Jansen and Lars Rohwedder. “A Quasi-Polynomial Approximation for the Restricted Assignment Problem”. In: *Proceedings of the 19th International Conference on Integer Programming and Combinatorial Optimization (IPCO)*. Vol. 10328. Lecture Notes in Computer Science. Springer, 2017, pp. 305–316.

[JR17b] Klaus Jansen and Lars Rohwedder. “On the Configuration-LP of the Restricted Assignment Problem”. In: *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2017, pp. 2670–2678.

[Kar84] Narendra Karmarkar. “A new polynomial-time algorithm for linear programming”. In: *Combinatorica* vol. 4, no. 4 (1984), pp. 373–396.

[KB15] Hessam Kooti and Eli Bozorgzadeh. “Reconfiguration-Aware Task Graph Scheduling”. In: *Proceedings of the 13th IEEE International Conference on Embedded and Ubiquitous Computing*. IEEE Computer Society, 2015, pp. 163–167.

[Kim+02] Dong-Won Kim, Kyong-Hee Kim, Wooseung Jang, and F Frank Chen. “Unrelated parallel machine scheduling with setup times using simulated annealing”. In: *Robotics and Computer-Integrated Manufacturing* vol. 18, no. 3-4 (2002), pp. 223–231.

[KLS10] Samir Khuller, Jian Li, and Barna Saha. “Energy Efficient Scheduling via Partial Shutdown”. In: *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2010, pp. 1360–1372.

[KMP12] Peter Kling, Friedhelm Meyer auf der Heide, and Peter Pietrzyk. “An Algorithm for Online Facility Leasing”. In: *Revised Selected Papers of the 19th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*. Vol. 7355. Lecture Notes in Computer Science. Springer, 2012, pp. 61–72.

[Kok05] Ravindranath Kokku. “Share: run-time system for high-performance virtualized routers”. PhD thesis. 2005.

[Kov10] Annamária Kovács. “New Approximation Bounds for Lpt Scheduling”. In: *Algorithmica* vol. 57, no. 2 (2010), pp. 413–433.

[KP00] Elias Koutsoupias and Christos H. Papadimitriou. “Beyond Competitive Analysis”. In: *SIAM Journal on Computing* vol. 30, no. 1 (2000), pp. 300–317.

[Li+15] Shouwei Li, Alexander Mäcker, Christine Markarian, Friedhelm Meyer auf der Heide, and Sören Riechers. “Towards Flexible Demands in Online Leasing Problems”. In: *Proceedings of the 21st International Conference on Computing and Combinatorics (COCOON)*. Vol. 9198. Lecture Notes in Computer Science. Springer, 2015, pp. 277–288.

[LK11] Gunho Lee and Randy H. Katz. “Heterogeneity-Aware Resource Allocation and Scheduling in the Cloud”. In: *Proceedings of the 3rd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*. USENIX Association, 2011.

[LL08] Joseph Y.-T. Leung and Chung-Lun Li. “Scheduling with processing set restrictions: A survey”. In: *International Journal of Production Economics* vol. 116, no. 2 (2008), pp. 251–262.

[LL16] Joseph Y.-T. Leung and Chung-Lun Li. “Scheduling with processing set restrictions: A literature update”. In: *International Journal of Production Economics* vol. 175 (2016), pp. 1–11.

[LLP05] Joseph Y.-T. Leung, Haibing Li, and Michael Pinedo. “Order scheduling models: an overview”. In: *Multidisciplinary scheduling: theory and applications*. Springer, 2005, pp. 37–53.

[Lóp16] Alejandro López-Ortiz. “Alternative Performance Measures in Online Algorithms”. In: *Encyclopedia of Algorithms*. Springer, 2016, pp. 67–72.

[LRS11] Lap Chi Lau, Ramamoorthi Ravi, and Mohit Singh. *Iterative methods in combinatorial optimization*. Vol. 46. Cambridge University Press, 2011.

[LS90] Hanoch Levy and Moshe Sidi. “Polling systems: applications, modeling, and optimization”. In: *IEEE Transactions on Communications* vol. 38, no. 10 (1990), pp. 1750–1760.

[LST90] Jan Karel Lenstra, David B. Shmoys, and Éva Tardos. “Approximation Algorithms for Scheduling Unrelated Parallel Machines”. In: *Mathematical Programming* vol. 46 (1990), pp. 259–271.

[LZ16] Song-Song Li and Yu-Zhong Zhang. “On-Line Scheduling on Parallel Machines to Minimize the Makespan”. In: *Journal of Systems Science & Complexity* vol. 29, no. 2 (2016), pp. 472–477.

[Mäc+15] Alexander Mäcker, Manuel Malatyali, Friedhelm Meyer auf der Heide, and Sören Riechers. “Non-preemptive Scheduling on Machines with Setup Times”. In: *Proceedings of the 14th International Symposium on Algorithms and Data Structures (WADS)*. Vol. 9214. Lecture Notes in Computer Science. Springer, 2015, pp. 542–553.

[Mäc+16] Alexander Mäcker, Manuel Malatyali, Friedhelm Meyer auf der Heide, and Sören Riechers. “Cost-Efficient Scheduling on Machines from the Cloud”. In: *Proceedings of the 10th International Conference on Combinatorial Optimization and Applications (COCOA)*. Vol. 10043. Lecture Notes in Computer Science. Springer, 2016, pp. 578–592.

[Mäc+17] Alexander Mäcker, Manuel Malatyali, Friedhelm Meyer auf der Heide, and Sören Riechers. “Non-clairvoyant Scheduling to Minimize Max Flow Time on a Machine with Setup Times”. In: *Revised Selected Papers of the 15th International Workshop on Approximation and Online Algorithms (WAOA)*. Vol. 10787. Lecture Notes in Computer Science. Springer, 2017, pp. 207–222.

[Mäc+18] Alexander Mäcker, Manuel Malatyali, Friedhelm Meyer auf der Heide, and Sören Riechers. “Cost-Efficient Scheduling on Machines from the Cloud”. In: *Journal of Combinatorial Optimization* vol. 36, no. 4 (2018). (A preliminary version has appeared in: Proceedings of the 10th International Conference on Combinatorial Optimization and Applications (COCOA) [Mäc+16]), pp. 1168–1194.

[Mäc18] Alexander Mäcker. “Approximating Maximum Flow Time on a Machine with Setup Times”. Unpublished Manuscript. 2018.

[Mas04] Monaldo Mastrolilli. “Scheduling To Minimize Max Flow Time: Off-Line And On-Line Algorithms”. In: *International Journal on Foundations of Computer Science* vol. 15, no. 2 (2004), pp. 385–401.

[Mey05] Adam Meyerson. “The parking permit problem”. In: *Proceedings of the 46th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2005, pp. 274–282.

[MH11] Sheheryar Malik and Fabrice Huet. “Virtual Cloud: Rent Out the Rented Resources”. In: *Proceedings of the 2011 International Conference on Internet Technology and Secured Transactions (ICITST)*. IEEE, 2011, pp. 536–541.

[MH12] Ming Mao and Marty Humphrey. “A Performance Study on the VM Startup Time in the Cloud”. In: *Proceedings of the 2012 IEEE 5th International Conference on Cloud Computing (CLOUD)*. IEEE, 2012, pp. 423–430.

[MLH10] Ming Mao, Jie Li, and Marty Humphrey. “Cloud auto-scaling with deadline and budget constraints”. In: *Proceedings of the 2010 11th IEEE/ACM International Conference on Grid Computing (GRID)*. IEEE Computer Society, 2010, pp. 41–48.

[MMP19] Alexander Mäcker, Friedhelm Meyer auf der Heide, and Simon Pukrop. “Approximating Weighted Completion Time for Order Scheduling with Setup Times”. (Submitted). 2019.

[MP89] Clyde L. Monma and Chris N. Potts. “On the Complexity of Scheduling with Batch Setup Times”. In: *Operations Research* vol. 37, no. 5 (1989), pp. 798–804.

[MP93] Clyde L. Monma and Chris N. Potts. “Analysis of Heuristics for Preemptive Parallel Machine Scheduling with Batch Setup Times”. In: *Operations Research* vol. 41, no. 5 (1993), pp. 981–993.

[MPT94] Rajeev Motwani, Steven Phillips, and Eric Torng. “Non-Clairvoyant Scheduling”. In: *Theoretical Computer Science* vol. 130, no. 1 (1994), pp. 17–47.

[MR11] Bodo Manthey and Heiko Röglin. “Smoothed Analysis: Analysis of Algorithms Beyond Worst Case”. In: *it - Information Technology* vol. 53, no. 6 (2011), pp. 280–286.

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[MU05] Michael Mitzenmacher and Eli Upfal. *Probability and computing - randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.

[NCY02] C. T. Ng, T. C. Edwin Cheng, and J. J. Yuan. “Strong NP-hardness of the single machine multi-operation jobs total completion time scheduling problem”. In: *Information Processing Letters* vol. 82, no. 4 (2002), pp. 187–191.

[NHL82] Katsuto Nakajima, S. Louis Hakimi, and Jan Karel Lenstra. “Complexity Results for Scheduling Tasks in Fixed Intervals on Two Types of Machines”. In: *SIAM Journal on Computing* vol. 11, no. 3 (1982), pp. 512–520.

[NW13] Chandrashekhar Nagarajan and David P. Williamson. “Offline and online facility leasing”. In: *Discrete Optimization* vol. 10, no. 4 (2013), pp. 361–370.

[Pin05] Michael L. Pinedo. *Planning and scheduling in manufacturing and services*. Springer Series in Operations Research. Springer, 2005.

[Pin16] Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, 2016.

[Pla+06] C. Greg Plaxton, Yu Sun, Mitul Tiwari, and Harrick M. Vin. “Reconfigurable resource scheduling”. In: *Proceedings of the 18th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. ACM, 2006, pp. 93–102.

[PLM18] Jose Antonio Pascual, José Antonio Lozano, and José Miguel-Alonso. “Effects of Reducing VMs Management Times on Elastic Applications”. In: *Journal of Grid Computing* vol. 16, no. 3 (2018), pp. 513–530.

[Pot91] Chris N. Potts. “Scheduling two job classes on a single machine”. In: *Computers & OR* vol. 18, no. 5 (1991), pp. 411–415.

[PST04] Kirk Pruhs, Jirí Sgall, and Eric Torng. “Online Scheduling”. In: *Handbook of Scheduling - Algorithms, Models, and Performance Analysis.* Chapman and Hall/CRC, 2004.

[Res+03] Javier Resano, Daniel Mozos, Diederik Verkest, Serge Vernalde, and Francky Catthoor. “Run-Time Minimization of Reconfiguration Overhead in Dynamically Reconfigurable Systems”. In: *Proceedings of the 13th International Conference on Field Programmable Logic and Application (FPL)*. 2003, pp. 585–594.

[Rou19] Tim Roughgarden. “Beyond Worst-case Analysis”. In: *Commununications of the ACM* vol. 62, no. 3 (2019), pp. 88–96.

[RT87] Prabhakar Raghavan and Clark D. Thompson. “Randomized rounding: a technique for provably good algorithms and algorithmic proofs”. In: *Combinatorica* vol. 7, no. 4 (1987), pp. 365–374.

[Sah13] Barna Saha. “Renting a Cloud”. In: *Proceedings of the Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*. Vol. 24. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013, pp. 437–448.

[Sah72] Vinod K. Sahney. “Single-Server, Two-Machine Sequencing with Switching Time”. In: *Operations Research* vol. 20, no. 1 (1972), pp. 24–36.

[Sga14] Jirí Sgall. “Online Bin Packing: Old Algorithms and New Results”. In: *Proceedings of the 10th Conference on Computability in Europe (CiE)*. Vol. 8493. Lecture Notes in Computer Science. Springer, 2014, pp. 362–372.

[Sga96] Jirí Sgall. “On-line Scheduling”. In: *Online Algorithms, The State of the Art*. Ed. by Amos Fiat and Gerhard J. Woeginger. Vol. 1442. Lecture Notes in Computer Science. Springer, 1996, pp. 196–231.

[She10] IG Shevtsova. “An improvement of convergence rate estimates in the Lyapunov theorem”. In: *Doklady Mathematics*. Vol. 82. 3. Springer, 2010, pp. 862–864.

[Sri99] Aravind Srinivasan. “Approximation algorithms via randomized rounding: a survey”. In: *Series in Advanced Topics in Mathematics, Polish Scientific Publishers PWN* (1999), pp. 9–71.

[SS05] Guido Schäfer and Naveen Sivadasan. “Topology matters: Smoothed competitiveness of metrical task systems”. In: *Theoretical Computer Science* vol. 341, no. 1-3 (2005), pp. 216–246.

[SSS06] Mark Scharbrodt, Thomas Schickinger, and Angelika Steger. “A New Average Case Analysis for Completion Time Scheduling”. In: *Journal of the ACM* vol. 53, no. 1 (2006), pp. 121–146.

[ST00] Hadas Shachnai and Tami Tamir. “Polynominal time approximation schemes for class-constrained packing problem”. In: *Proceedings of the 3rd International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX)*. Vol. 1913. Lecture Notes in Computer Science. Springer, 2000, pp. 238–249.

[ST04] Daniel A. Spielman and Shang-Hua Teng. “Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time”. In: *Journal of the ACM* vol. 51, no. 3 (2004), pp. 385–463.

[ST09] Daniel A. Spielman and Shang-Hua Teng. “Smoothed analysis: an attempt to explain the behavior of algorithms in practice”. In: *Commununications of the ACM* vol. 52, no. 10 (2009), pp. 76–84.

[ST93] David B. Shmoys and Éva Tardos. “An approximation algorithm for the generalized assignment problem”. In: *Mathematical Programming* vol. 62 (1993), pp. 461–474.

[SV02] Leen Stougie and Arjen P. A. Vestjens. “Randomized algorithms for on-line scheduling problems: how low can’t you go?” In: *Operations Research Letters* vol. 30, no. 2 (2002), pp. 89–96.

[SV05] Evgeny V. Shchepin and Nodari Vakhania. “An optimal rounding gives a better approximation for scheduling unrelated machines”. In: *Operations Research Letters* vol. 33, no. 2 (2005), pp. 127–133.

[Sve12] Ola Svensson. “Santa Claus Schedules Jobs on Unrelated Machines”. In: *SIAM Journal on Computing* vol. 41, no. 5 (2012), pp. 1318–1341.

[SW99] Petra Schuurman and Gerhard J. Woeginger. “Preemptive Scheduling with Job-Dependent Setup Times”. In: *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 1999, pp. 759–767.

[SWW91] David B. Shmoys, J. Wein, and D.P. Williamson. “Scheduling Parallel Machines On-line”. In: *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science (FOCS)*. 1991, pp. 131–140.

[Vaz01] Vijay V. Vazirani. *Approximation algorithms*. Springer, 2001.

[WS11] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

[YZ09] Guosong Yu and Guochuan Zhang. “Scheduling with a minimum number of machines”. In: *Operations Research Letters* vol. 37, no. 2 (2009), pp. 97–101.