Handling Critical Tasks Online

Deadline Scheduling and Convex-Body Chasing

vorgelegt von
Kevin Schewior, M.Sc.
geboren in Duisburg

Von der Fakultät II – Mathematik und Naturwissenschaften
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften
– Dr. rer. nat. –

genehmigte Dissertation

Promotionsausschuss

Vorsitzender:	Prof. Dr. Wolfgang König
Berichter:	Prof. Dr. Nicole Megow
	Prof. Dr. Martin Skutella
	Prof. Clifford Stein, Ph.D.

Tag der wissenschaftlichen Aussprache: 23. Juni 2016

Berlin 2016

Acknowledgements

First and foremost, I would like to thank my advisor Nicole Megow for her support and encouragement, without which this thesis would not have been possible in this form. In particular, I am very happy with the way she made me acquainted with the research community, and I very much enjoy working on the problems to which she introduced me.

Many thanks also go to Martin Skutella, for instance, for being a reviewer of this thesis. In fact, I would like to thank the whole COGA group in Berlin, who made it possible to work in a very productive and relaxed atmosphere. An integral part of the surroundings in Berlin consisted in the research training group "Methods for Discrete Structures", which supported me with a scholarship and which was an important platform for a university-, city-, country-, and worldwide exchange with researchers. Being supported by the graduate school "Berlin Mathematical School" further enhanced this situation. I feel honored that I was able to work in this stimulating environment. I made many friends here, and I am looking forward to visiting my former home.

Further I am very grateful to Kirk Pruhs for warmly welcoming me in Pittsburgh for a research visit, which I truly enjoyed. In addition, he arranged a number of follow-up research weeks, which I appreciate very much.

Moreover, I would like to express my gratitude to Cliff Stein for his willingness to be a reviewer of this thesis and coming to Berlin for the defense.

In addition to the co-authors already mentioned, I really enjoyed the discussions with Lin Chen, which led to co-authoring two papers. Also, I was very fortunate to work with Antonios Antoniadis, Neal Barcelo, Michael Nugent, and Michele Scquizzato on results that have become part of this thesis.

Moreover, I would like to thank Antonios Antoniadis, Miriam Schlöter, Nicole Megow, and Yann Disser for proofreading different parts of this thesis.

Last but not least, I would like to thank my family and friends for their constant support and belief in my abilities.

Berlin, April 2016 *Kevin Schewior*

This research was supported by the German Science Foundation (DFG) within the research training group 'Methods for Discrete Structures' (GRK 1408) and under contract ME 3825/1.

Contents

Chapter 1

Introduction

The outcome of many decisions made in the real world heavily depends on future events. Unfortunately, it is often hard or expensive to obtain information about the future, or it may even be impossible. Thus, to optimize the outcome of the above decisions, one cannot use classical optimization techniques, which typically require extensive knowledge about all parameters influencing the outcome. Instead, one desires optimization techniques that can handle data arriving over time and that are able to make irrevocable decisions already during this process, that is, *online*.

Approximately 30 years ago, Sleator and Tarjan [ST85] first investigated online algorithms in depth, in the context of data structures. They performed what is now called a *competitive analysis*, as coined by Karlin et al. [KMRS88]. Here, the performance of an online algorithm is evaluated by the *competitive ratio*, that is, the worst-case ratio of the cost of the solution the algorithm produces and the cost of an optimal *offline* solution. In contrast to the solution produced by the online algorithm, the offline solution is constructed based on the entire input instance. Taking a different viewpoint, the input sequence is constructed online by an adversary that strives to maximize the above ratio based on the decisions that the algorithm makes.

The adversary that can adaptively react to each decision that the algorithm makes is rather strong: For some problems, reasonable competitive ratios, such as ratios that are constant or (poly-)logarithmic in the input size, are not achievable. This issue is partly solved by considering *randomized online algorithms* and an *oblivious adversary*, a concept introduced by Ben-David et al. [BBK+94]. Here, the algorithm's decisions may involve randomness, and the competitive ratio is measured via the expected cost of the algorithm on the input instance given by the adversary. This adversary is weaker in the sense that it may only design the input sequence based on the specification of the algorithm and not on the outcome of the random events, that is, the adversary is oblivious to the outcome of the random events.

Some problems, however, are even more difficult in the sense that there does not even exist a randomized algorithm whose competitive ratio against an oblivious adversary can be reasonably bounded. This is the case, for example, for some single-machine scheduling problems. To address this problem, *resource augmentation* was proposed by Kalyanasundaram and Pruhs [KP00]: In this setting, the online algorithm may use more resources, such as higher machine speed, than the offline solution. One can then either consider a trade-off between the competitive ratio and the amount of used resources, or

one can try to achieve a competitive ratio of 1 and minimize the amount of resources needed for that.

In this thesis, we consider two different online problems, online deadline scheduling and convex-body chasing. In most of the cases, it turns out that reasonable bounds according to one of the above measures can be obtained.

1.1 Considered Problems and Contribution

There are important real-world applications in which one faces tasks that are released over time and that are time-critical, meaning that they *have* to be finished either immediately or before a certain deadline: In a hospital, patients that come in may need to be treated within a certain time frame. Another example is an on-board computer on an aeroplane or in a car that has to perform safety-relevant tasks such as checking for obstacles. In this thesis, we consider two abstract problems that are similar in the sense that they are both given a sequence of such tasks as input, and these tasks have to be finished either immediately or before a certain deadline.

Another property that unifies the two problems that we consider in this thesis is their fundamentality: Both problems have a very simple formulation, but they have not yet been understood well. In fact, understanding them seems to be at the core of understanding the online features of a more general class of problems.

1.1.1 Online Deadline Scheduling

In classical online scheduling problems, one is typically given the same machines (resources) as the offline solution, and one tries to be competitive w.r.t. a certain objective function such as the latest completion time as considered by Graham [Gra69]. Being competitive given the same resources as the offline solution is, however, impossible in *online deadline scheduling* as considered in the seminal work by Phillips et al. [PSTW02]. The problem is stated as follows.

We are given an integer m, and jobs arrive over time at their release dates and require to be processed for a respective processing time until their respective deadline. In this chapter, we will only consider the model where jobs can be started at any time on any machine, preempted at any time, and continued at any later time. In the *migratory setting*, the job may be continued on any machine; in the *non-migratory setting*, it must be continued on the machine it was started on. We assume that the input instance has a feasible (offline) schedule on m machines in the sense that all the jobs are completed by their deadlines.

As shown by Dertouzos and Mok [DM89], there is no online algorithm that computes feasible schedules on m machines for all instances. Since we do want to meet the deadlines of *all* jobs, there is no natural and suitable concept of competitive ratio here. Instead, we use resource augmentation and wish to minimize the amount of resources that we use to meet the deadlines of all jobs.

In Chapter 2, we consider augmenting the number of unit-speed machines, that is, we consider online algorithms that use $m' > m$ machines. It was shown by Phillips et al. [PSTW02] that we must have $m' > 5m/4$ for $m \geq 2$ so that the online algorithm can

meet all deadlines (in this case, we call the algorithm an *m'-machines algorithm*). Nevertheless, a wide open question mentioned in [CLT05, Pru10] was whether *any* constant m' suffices even when $m = 2$ for the migratory as well as the non-migratory setting. It was already ruled out that simple algorithms which prioritize jobs by their deadlines or their slack achieve this.

Contributions of Chapter 2. For the migratory setting, we present an $\mathcal{O}(m \log m)$-machines online algorithm. For the non-migratory setting, however, we show that no $f(m)$-machines algorithm exists for any function f. This may be surprising since any migratory schedule on m machines meeting all deadlines can be transformed (offline) into a non-migratory schedule meeting all deadlines on $\mathcal{O}(m)$ machines as shown by Kalyanasundaram and Pruhs [KP01]; also, Chan, Lam, and To [CLT05] showed that, in the non-migratory setting, an $\mathcal{O}(m)$-machines online algorithm exists when the speed on each machine is also augmented by a factor of $1 + \varepsilon$, for an arbitrarily small $\varepsilon > 0$.

The crux of designing a migratory $f(m)$-machines algorithm turns out to be handling jobs whose processing time is large compared to their lifespan (*tight* jobs). We handle these jobs via a new balanced delaying scheme, which we analyze using a new necessary condition for the existence of a feasible schedule for tight jobs. Using a similar proof technique, we are able to show that the same algorithm is an $\mathcal{O}(m)$-machine algorithm for special cases w.r.t. the interval structure induced by the jobs' time windows.

Our lower bound for non-migratory online algorithms relies on a novel construction which recursively forces the algorithm to spread jobs over machines in such a bad way that releasing an additional jobs forces the algorithm to open a new machine. Leveraging some of the aforementioned results, we are able to give $f(m)$-machines algorithms for suitable functions f at least for some special cases, that is, jobs that are not tight and special cases w.r.t. the interval structures induced by the time windows.

We provide a summary of the state-of-the-art results on online deadline scheduling with machine augmentation in Table 2.1.

In Chapter 3, we consider augmenting the speed on each machine instead of the number of (unit-speed) machines in the migratory setting. Known results include that the simple earliest-deadline-first (EDF) algorithm is a speed-$(2-1/m)$ algorithm, that is, a speed of $(2-1/m)$ on each machine suffices for it to always produce feasible schedules, as shown by Phillips et al. [PSTW02]. While this result is tight for EDF, the same upper bound holds for the natural greedy algorithm LLF, which at each time prefers those unfinished jobs that have the smallest slack left. The best known lower and upper bounds on the speed requirement were shown by Lam and To [LT99], and they are (asymptotically for $m \to \infty$) 1.207 and $2 - 2/(m+1)$.

Contributions of Chapter 3. In this chapter, we show that the algorithm YSS proposed by Anand, Garg, and Megow [AGM11] is not a speed-$e/(e-1) \approx 1.58$ algorithm. We propose the new algorithm YS-LLF, which fixes issues of YSS by using ideas from LLF. Indeed, YS-LLF is a candidate to be a speed-$e/(e-1)$ algorithm. While we are not able to provide a proof, we give some justification.

We also consider LLF separately, and we show a lower bound of $2-1/m$ on the speed requirement of this algorithm, which surprisingly matches the speed requirement for the

significantly simpler algorithm EDF. In defense of LLF, we prove that it is a speed-1 algorithm for instances with a single release date while EDF already shows its worst-case behavior on such an instance. In fact, our lower-bound construction for LLF requires an unbounded number of release dates, and we conjecture that, for every fixed number of release dates, it is a speed-$(2-\varepsilon)$ algorithm for some $\varepsilon > 0$.

1.1.2 Convex-Body Chasing

The second problem that we consider is a special case of the fundamental and very general problem of *metrical task systems*, which is a classical online problems due to Borodin, Linial, and Saks [BLS92]. In this problem, an online server starts at a point of a metric space and receives a sequence of functions that map from that metric space to $\mathbb{R}_+$. In response to each function, the server may move to a new point in the metric space. The cost in this step is the distance moved plus the function value at the point at which the server ends up after the optional move. The goal is to be competitive w.r.t. the aggregate cost.

Our focus is on the following special case of this problem, called *convex-body chasing* and already considered by Friedman and Linial [FL93]: The considered metric space is $\mathbb{R}^d$ equipped with the standard Euclidean metric. Further, each function that arrives has value 0 within some convex body and ∞ (or some very large value) outside the convex body. This makes the task of moving to the convex body critical, that is, the online server *has* to move to it so as to stay competitive. The main result by Friedman and Linial is an online algorithm for convex-function chasing with constant competitive ratio when $d = 2$.

We also consider two more special cases of metrical task systems which are more general than convex-body chasing: In *lazy convex-body chasing*, a non-negative slope is presented along with each convex body, and the corresponding function is not ∞ outside the convex body any more but grows linearly at the respective slope as one moves away from the convex body. Further, *convex-function chasing* is the special case of metrical task systems where the metrical space is again $\mathbb{R}^d$ and each presented function is convex.

We provide an overview of these problems in Table 4.1.

Contributions of Chapter 4. Our main result is a new online algorithm with competitive ratio $2^{\mathcal{O}(d)}$ for hyperplane chasing in d dimensions, that is, the special case of convex-body chasing where all convex bodies are hyperplanes. This settles an open problem, which was posed by Friedman and Linial [FL93]. Our result rests on a new reduction from lazy convex-body chasing to (non-lazy) convex-body chasing, both in the same dimension. This means that, if there is an online algorithm with constant competitive ratio for convex-body chasing, such an algorithm also exists for lazy convex-body chasing. Our proof relates the slope of a convex body in lazy convex-body chasing with the probability with which it appears in a convex-body chasing instance.

To obtain the competitiveness result for hyperplane chasing, we apply the latter reduction alternatingly with another reduction from hyperplane chasing in d dimensions to lazy hyperplane chasing in $d-1$ dimensions from Friedman and Linial [FL93]. For $d = 2$, that is, line chasing in two dimensions, we simplify and slightly improve the algorithm due

to Friedman and Linial [FL93] and then generalize it to line-segment chasing in arbitrary dimensions.

Finally, for function chasing in one dimension, we provide a new lower bound on the achievable competitive ratio, leaving a gap between 1.86 and 2.

Chapter 2

Online Deadline Scheduling with Machine Augmentation

Bibliographic information. Most of the results presented in this section are published in [CMS16a] or [CMS16b].

This section deals with the fundamental problem of online deadline scheduling with machine augmentation, as motivated and informally introduced in Section 1.1. We formally define the problem in Section 2.1, and we name known as well as new results in Sections 2.2 and 2.3, respectively. Our results are presented in Sections 2.4–2.7. We conclude with open problems in Section 2.8.

2.1 Preliminaries

We first define the machine-augmentation problem, which was introduced by Phillips et al. [PSTW02]. Afterwards we give notations and terms that will be used throughout the chapter.

Problem Defintion. We consider the following online scheduling problem, called *online deadline scheduling*. As input, we are given an integer m and a set of *jobs* $J = \{1, \dots, n\}$, where each job $j \in J$ has a *release date* $r_j \in \mathbb{N}$, a *processing time* $p_j \in \mathbb{N}$, and a *deadline* $d_j \in \mathbb{N}$. In a *schedule* of J on m' machines, each job $j \in J$ may be started on any machine of the m' machines at any time at or after r_j, and we consider different settings regarding the continuation of the job:

- We consider the *migratory setting* in which j may be preempted at any time. It can be continued at any later time on any machine, that is, it may be *migrated* onto another machine.
- In the *non-migratory setting*, a job may also be preempted and continued at any point in time, but it may only be continued on the machine it was started on. That is, it may not be migrated.
- Once a job j is started in the *non-preemptive* setting, it must run for p_j time units. Here, it will be easy to see that it does not matter if migration is allowed or not.

In each of the settings and at all times, each machine may only work on at most one job, and each job may only be processed by at most one machine. We say that a job j is

finished at time t if it has run for p_j time units (in aggregate) until t, and we say that its deadline is *met* if the job is finished at d_j. We assume that the input instance is *feasible* (on m machines) in the sense that it has a schedule that meets all deadlines (a *feasible schedule*) on m machines. Throughout this chapter, we will use m to denote the number of machines given as input.

An online scheduler learns about each job only at the job's release date. The task in this case is also to produce a feasible schedule. As shown in the literature [DM89], there is no online algorithm that is guaranteed to produce a feasible schedule using only m machines (see Section 2.2 for a more detailed discussion). This result motivates increasing (augmenting) the number of machines that the online algorithm uses and the following notion: We call an online algorithm an *m'-machines algorithm* if it is guaranteed to produce feasible schedules on m' machines.

The problem defined above can be viewed as a *semi-online* problem in the sense that the input consists not only of an instance J but also of m. We will also consider the *fully online* problem where m is not known to the online scheduler. Here, we can assume that m is the minimum number of machines needed to schedule J feasibly, which we will also denote by $m(J)$ if the instance is not clear from the context. In this context, it also makes sense to speak about the competitive ratio (as introduced in Chapter 1), where the cost of the schedule is the number of machines it uses: We call an algorithm *c-competitive* if it is guaranteed to produce feasible schedules on cm machines. We will, however, show in Section 2.5 that both settings are essentially equivalent, and, unless stated differently, we assume in the following that m is known to the online scheduler.

Instead of increasing the number of machines w.r.t. the offline schedule, one can also increase the *speed* on each machine from 1 to $s \in \mathbb{R}$, meaning that each job j is already finished when it has run for p_j/s (instead of p_j) time units. We call an online algorithm a speed-s algorithm if it is guaranteed to produce feasible schedules on m speed-s machines. We will only need this notion occasionally in this chapter. For a more rigorous definition and an in-depth discussion, see Chapter 3.

Further Notation. For a job j, the remaining processing time at time t is denoted by $p_j(t)$. The *laxity* of j at t is the maximum duration j can remain unprocessed from time t until it has to be started in order to meet its deadline, and this value is denoted by $\ell_j(t) \coloneqq d_j - r_j - p_j(t)$. In case it will not be clear which algorithm A the remaining processing time or laxity refers to, we will add A as superscript. The *initial laxity* (sometimes simply laxity) of j is $\ell_j \coloneqq \ell_j(r_j)$. For all job parameters (such as release dates, processing times, etc.) x_j, we set $x_{\min} \coloneqq \min_{j \in J} x_j$ and $x_{\max} \coloneqq \max_{j \in J} x_j$.

The *processing interval* of j is $I(j) \coloneqq [r_j, d_j)$, and j is said to *cover* each $t \in I(j)$ or be *available* at each $t \in I(j)$. For a set of jobs S, we define $I(S) = \bigcup_{j \in S} I(j)$. For a union of intervals $I = \bigcup_{i=1}^{k} [a_i, b_i)$ where $[a_1, b_1), \ldots, [a_k, b_k)$ are pairwise disjoint, we define the *length* of I to be $|I| \coloneqq \sum_{i=1}^{k} (b_i - a_i)$. The *contribution* of j to I is the minimum processing that j has to receive in I (ignoring all other jobs) and denoted by $C(j, I) := \max\{|I \cap I(j)| - \ell_j, 0\}$. The contribution of a set of jobs S to I is the sum of all individual contributions of jobs in S to I, that is, $C(S, I) \coloneqq \sum_{j \in S} C(j, I)$.

Throughout this chapter we assume w.l.o.g. that jobs are indexed in increasing order of release dates, and we break ties in decreasing order of deadlines. Hence, for any two

jobs j, j' with $j < j'$ one of these three cases holds: (i) $r_j < r_{j'}$, (ii) $r_j = r_{j'}$ and $d_j > d_{j'}$, or (iii) $I(j) = I(j')$.

Simple Algorithms. For arbitrary m', we define two algorithms that produce schedules on m' machines. They are both *busy* in the sense that, they only run $k < m'$ at times when there are only k unfinished jobs available. At any integer time when there are more than m' unfinished jobs available, the algorithm *Earliest Deadline First* (EDF) runs m' of them with minimum deadline (breaking ties arbitrarily). Likewise, at any integer time t when there are more than m' unfinished jobs available, the algorithm *Least Laxity First* (LLF) picks m' jobs with minimum laxity at time t (again breaking ties arbitrarily) and runs each of them within the interval $[t, t+1)$. In the context of speed augmentation, we will give a formulation of LLF that avoids tie breaking (Section 3.4).

Special Instances. Given a parameter $\alpha \in [0, 1]$, a job j is called *α-loose* if its processing time is at most an α-fraction of the length of its feasible time window $[r_j, d_j)$, that is, if $p_j/(d_j - r_j) \leq \alpha$. Otherwise we call the job *α-tight*.

We also classify instances on the base of their interval structure. An instance J is *laminar* if, whenever the feasible time windows of two jobs intersect, they must be contained in one another. Formally, for all jobs $j, j' \in J$, we have

$$I_j \cap I_{j'} \neq \emptyset \Rightarrow I_j \subseteq I_{j'} \vee I_{j'} \subseteq I_j.$$

Somewhat complimentarily, in *agreeable* instances, whenever the feasible time windows of two jobs intersect, they can only be contained in one another if they coincide in one of their endpoints (that is, the jobs' release dates or deadlines). Equivalently, for all jobs $j, j' \in J$, we have

$$r_j < r_{j'} \Rightarrow d_j \leq d_{j'}.$$

2.2 Related Work

We first consider the migratory online setting. As shown by Dertouzos and Mok [DM89], both EDF and LLF are 1-machine algorithms when $m = 1$, but there is no m-machines algorithm for any $m \geq 2$. In other words, the setting where the online algorithm and the offline adversary

(i) are provided the same resources
(ii) and both have to meet the deadlines of all jobs exactly

is hopeless. In the following, we discuss different approaches to overcoming this issue.

One approach would be to relax Requirement (ii). For instance, one could replace the goal by maximizing the number of jobs which are completed by their deadlines as done for a single machine by Kalyanasundaram and Pruhs [KP98], in which setting an $\mathcal{O}(1)$-competitive algorithm is obtained. In our approach, however, Requirement (ii) is kept and Requirement (i) is relaxed, that is, we use resource augmentation, which was first considered by Kalyanasundaram and Pruhs [KP00] in the context of various single-machine scheduling problems. In their seminal paper, Phillips et al. [PSTW02]

first applied resource augmentation to our setting and considered m'-machine algorithms for $m' > m$. For a comprehensive review of their work on speed augmentation, see Section 3.2.

Extending the result of Dertouzos and Mok, Phillips et al. [PSTW02] also showed that, for even m, there is no cm-machines algorithm for any $c \leq 5/4$. They also showed that, even for $m = 2$, neither EDF nor LLF is an m'-machines algorithm for any m'. Considering the maximum ratio $\Delta \coloneqq p_{\max}/p_{\min}$ between two processing times, they have shown that LLF is an $\mathcal{O}(\log \Delta)$-machines algorithm, while EDF is an $\Omega(\Delta)$-machines algorithm. Nevertheless, again even for $m = 2$, it remained a famous open problem whether an m'-machine algorithm exists for some m' [Pru10], but this problem will be solved in this this chapter. In the presence of any additional speed, the problem was already solved before: Lam and To [LT99] proved that EDF, for any $\varepsilon > 0$ and m, produces feasible schedules on $\mathcal{O}(m)$ machines of speed $1 + \varepsilon$.

The existence of an $\mathcal{O}(m)$-machines algorithm with speed $1 + \varepsilon$ was also shown in the non-migratory setting by Chan, Lam, and To [CLT05]. This is achieved by only assigning a job to a machine if the machine has no work assigned to it left that must be finished until the job's deadline. Then EDF is run on each machine. In the same paper, the same problem without additional speed is mentioned, but no results have been provided for this problem in the literature. There has, however, been work on the related problem of online interval coloring: Intervals arrive one by one, and upon arrival they have to be colored such that no two intersecting intervals receive the same color. The goal is to be competitive w.r.t. the number of used colors. When the intervals arrive in order of their leftmost points, the problem is equivalent to our problem with zero-laxity jobs with m unknown, where intervals correspond to jobs and colors to machines. Kierstead and Trotter [KT81] provide a 3-competitive algorithm, which is best possible.

The non-preemptive variant of the problem is quite hopeless in the sense that there is no $o(n)$-machines algorithm, even for $m = 1$, as noticed by Saha [Sah13]. Concerning Δ, the machine requirement for an online algorithm was settled to be $\Theta(\log \Delta)$ in the same work. This is based on the standard technique of assigning jobs to buckets corresponding to exponentially increasing processing times; low- and high-laxity jobs in each bucket are then treated separately. The special case of jobs with unit processing times that can only be started at discrete time points received some attention in a series of papers [KNST99, KCRW12, DMPY14] and independently in the context of energy minimization [BKP07]. While the problem is trivially solvable by EDF when m is known, there is a best-possible e-competitive algorithm for unknown m [DMPY14, BKP07].

We mention known results on the offline problem, in which all jobs are known in advance, and the number of used machines is to be minimized. If preemption and migration are allowed, the problem can be solved optimally in polynomial time [Hor74]. The solution of the natural linear-programming formulation can be rounded in a straightforward way by assigning fractionally assigned workload in a round-robin fashion over all machines within each time unit.

The offline problem gets substantially harder when migration (or even preemption) is not allowed: The computational problem of deciding feasibility of an instance can be shown to be strongly $\mathcal{NP}$-hard by a straightforward reduction from 3-Partition [GJ79]. In the non-migratory setting, there is a 12-approximation as shown implicitly by Kalyanasundaram and Pruhs [KP01]: They show that any feasible migratory schedule can be

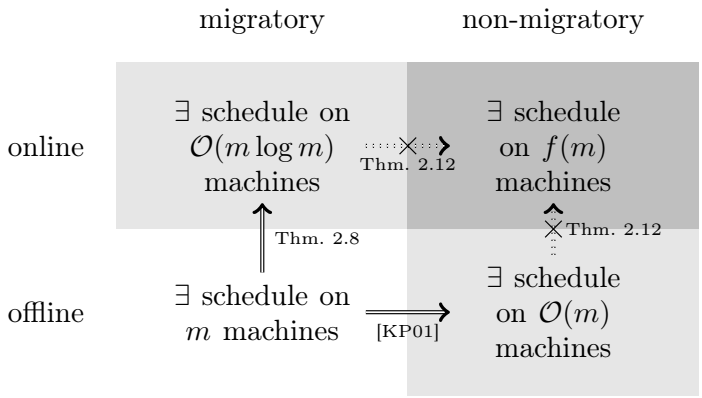

Figure 2.1: The main contribution of this chapter. Suppose there exists a feasible migratory schedule on a constant number of machines for some instance. Then, for the same instance and a (sufficiently large) constant number of machines, a feasible non-migratory schedule exists and a feasible migratory schedule can be constructed online. However, the machine requirement of a non-migratory algorithm is unbounded.

transformed into a feasible non-migratory one in polynomial time at the cost of an additional factor of 12 in the number of machines. For a purely structural result, that is, when the running time does not need to be bounded, this factor can be reduced to 6 [KP01].

For the non-preemptive setting, it is open whether a constant-factor approximation exists. It was shown by Cieliebak et al. [CEH$^+$04] that there is no $(2-\varepsilon)$-approximation algorithm unless $\mathcal{P} = \mathcal{NP}$. The best known approximation factor of $\mathcal{O}(\sqrt{\log n / \log \log n})$ is due to Chuzhoy et al. [CGKN04]. For special cases, including that of equal release dates and processing times, constant approximation factors were obtained [CEH$^+$04, YZ09].

2.3 Contribution and Outline

The main contribution of this chapter is on migratory and non-migratory online algorithms, for which we obtain quite different results: On the one hand, we show that there is an $\mathcal{O}(m \log m)$-machines algorithm for migratory online deadline scheduling (Subsections 2.6.1 and 2.6.2); on the other hand, we show that there is no $f(m)$-machines algorithm for non-migratory online deadline scheduling and any function f (Subsection 2.7.1). These results settle long-standing open problems [Pru10, CLT05]. Our results can be interpreted in the following way, using the fact that any feasible migratory schedule can be turned into a feasible non-migratory one at the cost of increasing the number of machines by a constant factor [KP01]: If the offline migratory schedule uses a constant number of machines, then an offline non-migratory schedule and an online migratory schedule can be constructed on a constant number of machines as well; the machine requirement for an online non-migratory schedule is, however, unbounded. In other words, the ability to migrate jobs is essential for an online algorithm. For an illustration of these results, see Figure 2.1.

Towards a migratory $f(m)$-machines algorithm, one may consider the standard lower-bound constructions for EDF and LLF [PSTW02] and note that, for both of the constructions, the following procedure produces a feasible schedule on $\mathcal{O}(m)$ machines: For

constant $\alpha \in (0,1)$, we treat α-loose and α-tight jobs on two separate sets of machines, and we schedules each of them via EDF or LLF. Indeed, our $\mathcal{O}(m \log m)$-machines algorithm treats α-loose and α-tight jobs separately on two different sets of machines (Subsections 2.6.1 and 2.6.2, respectively). Whereas we show that EDF and LLF are indeed $\mathcal{O}(m)$-machine algorithms for α-loose jobs (Subsection 2.6.1), the same two simple algorithms turn not to be a $f(m)$-machines algorithms for α-tight jobs and any function f (Subsection 2.6.2).

To illustrate how our $\mathcal{O}(m \log m)$-machines algorithm for α-tight jobs works, we interpret the laxity of each job as a budget for delaying it: A job's budget remains unchanged whenever the job is processed, and the budget is decreased at unit speed while the job is not being processed. Following this line of thought, EDF does not consider the jobs' budgets at all while LLF spends it too greedily. Towards a more mindful management of the budget, we divide the budget into $\mathcal{O}(m \log m)$ "sub-budgets", where the i-th sub-budget is only used if there are $i-1$ jobs with an earlier release date already running. This is driven by a new lower bound on m for α-tight jobs (Subsection 2.4.1 and below).

Leveraging the same lower bound, we can prove that the same algorithm is an $\mathcal{O}(m)$-machines algorithm for agreeable and laminar instances (Subsection 2.6.3). It is a natural question whether an online partition of the instance into $f(m)$ agreeable and laminar instances is possible – clearly, if $f(m) = \mathcal{O}(1)$, the existence of such a partition procedure would directly imply an $\mathcal{O}(m)$-machines algorithm. We show that such a procedure does not exist for any function f unless it dynamically reassigns jobs to other subinstances.

While a non-trivial lower bound for laminar instances was already implicitly given in [PSTW02], we give the first non-trivial lower bound for agreeable instances in this chapter (Subsection 2.6.3). More specifically, we show that there is no $(6-2\sqrt{6}-\varepsilon)m$-machines algorithm for agreeable instances and any $\varepsilon > 0$. Note that $6-2\sqrt{6} \approx 1.10$.

Our strong lower bound for non-migratory online algorithms (Subsection 2.7.1) shows in particular that every non-migratory algorithm is an $\Omega(\log n)$-machines algorithm, even when $m = 3$. This result is remarkable for at least two different reasons: Firstly, as noted above (also see Figure 2.1), in the offline setting migration can be eliminated at a constant-factor increase in the number of machines [KP01], but in the online setting, using our strong lower bound, this increase is unbounded in m. Secondly, there is a non-migratory $\mathcal{O}(1)$-speed algorithm [CLT05], which may suggest similarity between the migratory and the non-migratory problem. This similarity, however, is not in accordance with the existence of a migratory $\mathcal{O}(m \log m)$-machines algorithm and a $\Omega(\log n)$ lower bound for non-migratory algorithms.

We derive the lower bound using a non-trivial recursive construction that relies on a careful choice of the underlying interval structure of the jobs' time windows. Unlike lower bounds from the literature [PSTW02, LT99], our construction is not based on forcing the algorithm to use up the wrong jobs' laxities and a load argument; we rather force the online algorithm to assign jobs to the wrong machines. Load arguments are only used machine-wise.

As suggested by the non-trivial upper bound on the speed requirement of non-migratory algorithms [CLT05], even α-loose jobs for fixed $\alpha \in (0,1)$ cannot be handled in a trivial way by a non-migratory algorithm. Indeed, EDF and LLF only have a natural formulation if migration is allowed. Still, we are able to obtain an $\mathcal{O}(m)$-machines algorithm (Subsection 2.7.2), employing the non-migratory algorithm [CLT05] that, for any

	migratory		non-migratory		non-preemptive	
	LB	UB	LB	UB	LB	UB
general	$\Omega(m)$ (trivial)	$\mathcal{O}(m \log m)$ (Theorem 2.8)	no $f(m)$ (Theorem 2.12)	–	no $f(m)$ (Saha [Sah13])	–
α-loose	$\Theta(m)$ (Theorem 2.5)		$\Theta(m)$ (Theorem 2.14)		no $f(m)$ (Saha [Sah13])	–
laminar	$\Theta(m)$ (Theorem 2.9)		$\Omega(m)$ (trivial)	$\mathcal{O}(m \log m)$ (Theorem 2.17)	no $f(m)$ (Saha [Sah13])	–
agreeable	$\Theta(m)$ (Theorem 2.9)		$\Theta(m)$ (Theorem 2.18)		$\Theta(m)$ (Theorem 2.18)	

Table 2.1: State-of-the-art machine requirements for general and special cases. Only asymptotical results in terms of m are included.

fixed $\varepsilon > 0$, produces feasible schedules on $\mathcal{O}(1)$ speed-$(1+\varepsilon)$ machines. We use this algorithm as a black box. To do so, we increase the processing times of our jobs by a certain factor (guided by the guaranteed speed-factor) and apply the black-box algorithm. Then we transform back the resulting schedule into a feasible schedule on unit-speed machines for the original instance. The tricky part is to relate the minimum number of machines required by the original instance to the number needed by the modified instance. Our key result is an upper bound on this increase, which might be of independent interest.

It may be surprising that we are also able to obtain non-migratory $f(m)$-machine algorithms for the fairly general special cases of laminar and agreeable instances (Subsections 2.7.3 and 2.7.4, respectively). To get an $\mathcal{O}(\log m)$-machines algorithm for laminar instances, we again treat α-loose and α-tight jobs separately for fixed $\alpha \in (0, 1)$. For α-tight jobs, we use similar ideas as for the migratory $\mathcal{O}(m \log m)$-machines algorithm for general instances, and we also use our new lower bound (see below). When the instance is agreeable, we can, instead of the above algorithm, even use EDF for α-loose jobs, which is well-defined in this case. Moreover, α-tight jobs are run exactly in the middle of their processing intervals; the analysis simply uses a load argument. Overall, we get a non-migratory $\mathcal{O}(m)$-machines algorithm for agreeable instances.

Note that these positive results for special cases also justify the complexity of our lower-bound construction; a direct construction as the one in Saha's lower bound for the non-preemptive problem [Sah13], in which all jobs are α-loose and form a laminar instance, is not possible in our case.

As mentioned above, our migratory algorithms and the non-migratory algorithm for laminar instances are driven by a new lower bound on m (Subsection 2.4.2). This lower bound relates for a given set of time intervals the number of (tight) jobs with intersecting time windows in these intervals to the fraction of the total interval occupied by the laxity of those jobs. Phrased differently, for given m, our new lower bound construction gives an upper bound on the number of jobs with intersecting time intervals, which is how we utilize it for proving our upper-bound result. This bound is in turn based on an exact characterization of feasibility of an instance in the migratory setting via the load that has to be scheduled within in a certain union of intervals (Subsection 2.4.1). The latter

characterization can be viewed as the application of strong LP duality to the LP used by Horn [Hor74].

A summary of our results and those from the literature is given in Table 2.1.

2.4 Feasible Instances

When we analyze our algorithms, we will always, at least implicitly, relate to the feasibility of the input instance. It is, however, difficult to directly relate to the definition of feasibility. The purpose of this sections is to describe properties that are easier to relate to and that follow from, or are even equivalent to, the feasibility of the input instance.

In this section, we focus on the preemptive setting. We start by giving an exact characterization of feasible instances in terms of contributions of jobs to intervals in Subsection 2.4.1, and in Subsection 2.4.2 we present a more elaborate way of proving infeasibility of instances consisting of tight jobs.

2.4.1 A Characterization of Feasibility

A property of feasible instances already used in [PSTW02] is the fact that there is no interval I with endpoints in $\mathbb{Q}$ to which the contribution of the instance is more than $m \cdot |I|$. We would, however, like to have a property that is even equivalent to feasibility, which is not true for this property, as the following example shows.

Example 2.1. *Consider the following instance S. There are four jobs $1, \dots, 4$ with unit processing time and zero laxity, where jobs 1 and 2 are released at time 0, and jobs 3 and 4 at time 2. Additionally released at time 0, there is job 5 with $p_5 = 2$ and $d_5 = 3$.*

It is easy to see that the instance is infeasible for $m = 2$ since jobs $1, \dots, 4$ have to run from their release dates through their deadlines, leaving no space for job 5. The contribution of the instance to every interval I is, however, at most $2|I|$, which can be seen easily by considering three cases:

- *If $|I| < 1$, we have $C(\{1, \dots, 4\}, I) \leq 2|I|$ and $C(5, I) = 0$, that is, $C(S, I) \leq 2|I|$.*
- *If $|I| \in [1, 2]$, we have $C(\{1, \dots, 4\}, I) \leq 2$ and $C(5, I) \leq |I| - 1$. Since $C(S, I)/|I|$ is largest when $|I| = 1$, we have $C(S, I) \leq 2|I|$.*
- *If $|I| > 2$, we have $C(\{1, \dots, 4\}, I) \leq 2|I| - 2$ and $C(5, I) \leq 2$, that is, $C(S, I) \leq 2|I|$.*

Fortunately, it turns out that the above property is equivalent to feasibility if we consider finite unions of intervals instead of just intervals. Indeed, it is easy to check that the contribution of the instance from the above example to $I = [0, 1) \cup [2, 3)$ is $5 > 4 = 2|I|$. More generally and formally, we prove the following theorem.

Theorem 2.1. *Let S be some job set and let $\mathcal{I}$ be the set of all finite unions of intervals with endpoints in $\mathbb{Q}$. Then S is feasibile on m machines if and only if $C(S, I) \leq m \cdot |I|$ for all $I \in \mathcal{I}$.*

In the proof, we will make use of the following construction due to Horn [Hor74], which reduces the problem of deciding feasibility of S on m machines to a maximum single-commodity flow problem: We construct a flow network consisting of a directed

graph $G_{S,m} = (V, A)$, a source-sink pair $s, t \in V$, and a capacity function $c : A \to \mathbb{N}$. We define $V \coloneqq \{s, t\} \cup V_S \cup V_T$, where

$$V_S \coloneqq \{v_j \mid j \in S\} \text{ and } V_T \coloneqq \{v_\vartheta \mid \vartheta \in \mathbb{N}, r_{\min} \leq \vartheta < d_{\max}\}$$

correspond to the job set S and the set of relevant integer time points, respectively. For each job $j \in S$, we add unit-capacity edges $(v_j, v_{r_j}), \ldots (v_j, v_{d_j-1})$. Furthermore, for each job j we add and edge (s, v_j) of capacity p_j, and for each time ϑ we add an edge (v_ϑ, t) of capacity m.

We have the following proposition, which is easy to see; for more details see [Hor74].

Proposition 2.1 (Horn [Hor74])**.** *There exists an s-t flow of value $\sum_{j \in S} p_j$ in $G_{S,m}$ if and only if S is feasible on m machines.*

Using this property together with the max-flow-min-cut theorem [FF56] now essentially shows (the non-trivial direction of) the theorem.

Proof of Theorem 2.1. We first show that, if S is feasible on m machines, $C(S, I) \leq m \cdot |I|$ for all $I \in \mathcal{I}$: Since there is a feasible schedule of S on m machines, which schedules a total volume of at most $m \cdot |I|$ within I, the total volume of S that has to be scheduled within I (that is, its contribution to I) is at most $m \cdot |I|$.

It remains to show that, if S is infeasible, we have $C(S, I^\star)\rceil > m \cdot |I^\star|$ for some $I^\star \in \mathcal{I}$. Since S is infeasible, applying Proposition 2.1 and the max-flow-min-cut theorem yields that the minimum s-t cut $C \subsetneq V$ is of value less than $\sum_{j \in S} p_j$ in $G_{S,m}$. By writing the cut value more explicitly, we get

$$\sum_{j : v_j \notin C} p_j + \sum_{j : v_j \in C} |N^+(v_j) \setminus C| + \sum_{t : v_t \in C} m < \sum_{j \in S} p_j, \tag{2.1}$$

where $N^+(v)$ denotes the set of vertices reachable from vertex v in one step. Equivalently, we have

$$\sum_{j : v_j \in C} p_j - \sum_{j : v_j \in C} |N^+(v_j) \setminus C| < \sum_{t : v_t \in C} m.$$

If we define $I^\star \coloneqq \bigcup_{t : v_t \in V_T \cap C} [t, t+1)$, we thus get

$$\sum_{j : v_j \in C} |I(j) \cap I^\star| - \ell_j = \sum_{j : v_j \in C} p_j - |I(j) \setminus I^\star| < m \cdot |I^\star|.$$

Now by considering the left-hand side of Inequality (2.1) again and recalling that C is a minimum cut, note that $p_j \geq |N^+(v_j) \setminus C|$ for all j with $v_j \in C$, and thus $|I(j) \cap I^\star| - \ell_j \geq 0$ for all these j. Similarly, $|I(j) \cap I^\star| - \ell_j \leq 0$ for all j with $v_j \notin C$. Thus, the latter inequality implies

$$\sum_{j \in S} \max\{|I^\star \cap I(j)|, 0\} = C(S, I^\star) < m \cdot |I^\star|,$$

which shows our claim. □

We note that this proof directly corresponds to an LP-based proof, which uses the natural LP corresponding to the graph construction and strong LP duality instead of the max-flow-min-cut theorem.

2.4.2 A Necessary Condition for Feasibility of Tight-Jobs Instances

In this subsection, we present a necessary criterion for feasibility or, equivalently, a way of proving infeasibility of an instance. This proof works by giving a lower bound on the minimum number of machines needed to schedule the instance which is larger than m. To obtain our new bound, we restrict now explicitly to α-tight jobs for some constant $\alpha \in (0,1)$, that is, we assume $p_j > \alpha(d_j - r_j)$ for any job j. This enables us to relate to the laxity of jobs. The main new ingredient is to take into account the number of intervals covering the time points in a given set of intervals and relate it to the laxity of those jobs. More formally, we use the following definition.

Definition 2.1. *Let G be a set of α-tight jobs and let T be a non-empty finite union of time intervals. For some $\mu \in \mathbb{N}$ and $\beta \in (0,1)$, a pair (G,T) is called* (μ,β)-critical *if*

(i) each $t \in T$ is covered by at least μ distinct jobs in G,

(ii) $|T \cap I(j)| \geq \beta \ell_j$ for any $j \in G$.

In this section, we show the following theorem.

Theorem 2.2. *If there exists a (μ,β)-critical pair, then $m \geq \frac{\mu - 4 - 4\cdot\lceil \log 8/\beta \rceil}{8 + 8/\alpha \cdot \lceil \log 8/\beta \rceil} = \Omega\left(\frac{\mu}{\log 1/\beta}\right)$.*

For given m, this theorem immediately implies the following upper bound on the number of jobs covering the relevant intervals.

Corollary 2.1. *If there exists a (μ,β)-critical pair, then $\mu = \mathcal{O}(m \log 1/\beta)$.*

To show the theorem by contradiction, we consider a (μ,β)-critical pair (G,T), and show how to select jobs from G with a total load contribution to some superset of T that contradicts Theorem 2.1.

Before we prove the Theorem, we need a few auxiliary lemmata. The following one may be folkloric, but we still give a proof.

Lemma 2.1. *Let S be a set of jobs. There exists a subset $S' \subseteq S$ such that each time in $I(S)$ is covered by at least one and at most two different jobs in S'.*

Proof. Given a set of jobs S, we construct S' by a simple greedy procesure. Initially S' is an empty set. Whenever there is a time point in $I(S)$ that is not covered by a job in S', we find the smallest such time point t. Let $j \in S$ be a job covering t and having the largest deadline. Clearly, such a job exists, and we add j to S'. This procedure continues until each time point in $I(S)$ is covered by a job in S'.

By construction, each time point in $I(S)$ is covered by at least one job in S'. It remains to show that each $t \in I(S)$ is covered by at most two jobs in S'. For the sake of contradiction, suppose there are three jobs $j_1, j_2, j_3 \in S'$ (added in this order) covering t, being added because of the uncovered time points $t_1 < t_2 < t_3$. As j_2 is added after j_1, we have $d_{j_1} < t_2$ and therefore, using that t is covered by both j_1 and j_2, we have $t < t_2$. Using this and the fact that j_3 covers t and $t_3 > t_2$, j_3 also covers t_2. Further it holds that $t_3 > d_{j_2}$, so we have $d_{j_3} > d_{j_2}$, which is a contradiction to the fact that j_2 (rather than j_3) is added by the greedy procedure to cover t_2. □

Using this simple lemma, we show the first of two important properties of critical pairs.

Lemma 2.2. *Let (G,T) be a (μ,β)-critical pair. There exist pairwise disjoint subsets $G_1^\star,\dots,G_{\lfloor\mu/4\rfloor}^\star$ of G such that*

(i) $T \subseteq I(G_1^\star) \subseteq \cdots \subseteq I(G_{\lfloor\mu/4\rfloor}^\star)$,

(ii) $\sum_{j\in G_i^\star} \ell_j \le 4|T|/\beta$ *for every* $1 \le i \le \lfloor\mu/4\rfloor$.

Proof. We first select disjoint subsets $G_1,\dots,G_{\lceil\mu/2\rceil}$ of G with a laminar interval structure, i.e., $I(G_1) \subseteq \cdots \subseteq I(G_{\lceil\mu/2\rceil})$, such that each time point in T is covered by at least one and at most two distinct jobs from each G_i: Starting from $i = \lceil\mu/2\rceil$, the iterative procedure selects a job set G_i from $G_i^+ \coloneqq G \setminus (G_{i+1} \cup \cdots \cup G_{\lceil\mu/2\rceil})$ using Lemma 2.1. To also satisfy (ii), we will later further select certain subsets from $G_1,\dots,G_{\lfloor\mu/2\rfloor}$.

Before that, we show that indeed $T \subseteq I(G_1) \subseteq \cdots \subseteq I(G_{\lceil\mu/2\rceil})$. Firstly, we show that $I(G_i) \subseteq I(G_{i+1})$ for every i. This follows from the fact that $I(G_1^+) \subseteq \cdots \subseteq I(G_{\lceil\mu/2\rceil}^+)$ (by definition of G_i^+) and $I(G_i) = I(G_i^+)$ for all i (by Lemma 2.1). Secondly, we show $T \subseteq I(G_1)$, i.e., each time point in T is covered by at least one job in G_1. Recall that every time point in T is covered by at least μ distinct jobs in G and, according to Lemma 2.1, covered by at most two distinct jobs in G_i for every i. Hence, any point in T is covered by at least $\mu - 2(\lceil\mu/2\rceil - 1) \ge 1$ jobs in $G_1^+ = G \setminus (G_2 \cup \cdots \cup G_{\lceil\mu/2\rceil})$, and $T \subseteq I(G_1^+) = I(G_1)$.

Next we apply a counting argument to show an averaging alternative of Condition (ii). We set $G' = G_1 \cup \cdots \cup G_{\lceil\mu/2\rceil}$, and note that each time in T is covered by at most $2\lceil\mu/2\rceil$ distinct jobs in G' because, as shown above, it is covered by at most two distinct jobs in each G_i. Also using $T \subseteq I(G_1) \subseteq \cdots \subseteq I(G_{\lceil\mu/2\rceil})$, we get

$$|T| = \left|\bigcup_{j\in G'} (T \cap I(j))\right| \ge \frac{\sum_{j\in G'} |T \cap I(j)|}{2\lceil\mu/2\rceil} \ge \frac{\sum_{j\in G'} \beta\ell_j}{2\lceil\mu/2\rceil}, \tag{2.2}$$

where the last inequality follows from the definition of a (μ,β)-critical pair.

Now we argue that we can further choose $G_1^\star,\dots,G_{\lfloor\mu/4\rfloor}^\star$ from the family of job sets $G_1,\dots,G_{\lceil\mu/2\rceil}$ such that Condition (ii) is true. Suppose there are no such sets, i.e., we have $\sum_{j\in G_i} \ell_j > 4\cdot|T|/\beta$ for $\lceil\mu/2\rceil - \lfloor\mu/4\rfloor + 1 \ge \lceil\mu/2\rceil/2$ different i. Then the total laxity of G' is

$$\sum_{j\in G'} \ell_j = \sum_{i=1}^{\lceil\mu/2\rceil} \sum_{j\in G_i} \ell_j > \left(\left\lceil\frac{\mu}{2}\right\rceil - \left\lfloor\frac{\mu}{4}\right\rfloor + 1\right) \cdot \frac{4|T|}{\beta} \ge \frac{2\lceil\mu/2\rceil\cdot|T|}{\beta},$$

which is a contradiction to (2.2). □

The following lemma states that, for arbitrary disjoint sets of α-tight jobs with a laminar interval structure, the total size of the covered time intervals is geometrically increasing.

Lemma 2.3. *Let $S_1,\dots,S_{\lceil 2m/\alpha\rceil}$ be pairwise disjoint sets of α-tight jobs such that $I(S_1) \subseteq \cdots \subseteq I(S_{\lceil 2m/\alpha\rceil})$. Then we have $|I(S_{\lceil 2m/\alpha\rceil})| \ge 2|I(S_1)|$.*

Proof. Suppose on the contrary that $|I(S_{\lceil 2m/\alpha \rceil})| < 2|I(S_1)|$. We will show a contradiction based on the total contribution of all the jobs to $I(S_{\lceil 2m/\alpha \rceil})$. By assumption we have $|I(S_i)| \geq |I(S_1)| > |I(S_{\lceil 2m/\alpha \rceil})|/2$ for all $i \in \{1, \ldots, \lceil 2m/\alpha \rceil\}$. Each of these sets S_i consists of α-tight jobs only, i.e., $p_j > \alpha(d_j - r_j)$ for any j, and thus, a workload of at least $\alpha \cdot |I(S_i)| \geq \alpha \cdot |I(S_{\lceil 2m/\alpha \rceil})|/2$ has to be processed within the interval $I(S_{\lceil 2m/\alpha \rceil})$. There are $\lceil 2m/\alpha \rceil$ different such sets S_i, and consequently the total workload that has to be processed within $I(S_{\lceil 2m/\alpha \rceil})$ is

$$C\left(\left(\bigcup_{i=1}^{\lceil 2m/\alpha \rceil} S_i\right),\ I(S_{\lceil 2m/\alpha \rceil})\right) > \left\lceil \frac{2m}{\alpha} \right\rceil \cdot \frac{\alpha \cdot |I(S_{\lceil 2m/\alpha \rceil})|}{2} \geq m \cdot |I(S_{\lceil 2m/\alpha \rceil})|,$$

which is a contradiction to Theorem 2.1. □

We are now ready to prove the main theorem.

Proof of Theorem 2.2. Let (G, T) be a (μ, β)-critical pair. Let $G^\star_1, \ldots, G^\star_{\lfloor \mu/4 \rfloor}$ be subsets of G that satisfy the two conditions of Lemma 2.2, i.e., (i) $T \subseteq I(G^\star_1) \subseteq \cdots \subseteq I(G^\star_{\lfloor \mu/4 \rfloor})$, and (ii) $\sum_{j \in G^\star_i} \ell_j \leq 4|T|/\beta$, for all $i \in \{1, \ldots, \lfloor \mu/4 \rfloor\}$. The proof idea is to bound the contribution of certain subsets $G^\star_q, \ldots, G^\star_{\lfloor \mu/4 \rfloor}$ to the interval $I(G^\star_q)$ and show the bound on m by achieving a contradiction to the load bound in Theorem 2.1.

In the following, we will fix $k \coloneqq \lceil \log 8/\beta \rceil$ and $q \coloneqq \lceil 2m/\alpha \rceil \cdot k$, and we distinguish two cases. If $G^\star_q$ does not exist, i.e., $q > \lfloor \mu/4 \rfloor$, we obtain

$$m \geq \frac{\alpha(\mu - 4 - 4k)}{8k} > \frac{\mu - 4 - 4 \cdot \lceil \log 8/\beta \rceil}{8 + 8/\alpha \cdot \lceil \log 8/\beta \rceil}$$

as claimed. Otherwise, $G^\star_q$ does exist, and it follows from $T \subseteq I(G^\star_1)$ and repeatedly applying Lemma 2.3 that

$$|I(G_q)| = |I(G^\star_{\lceil 2m/\alpha \rceil \cdot k})| \geq 2^k |T|.$$

Now consider any $G^\star_i$, for $i \in \{1, \ldots, \lfloor \mu/4 \rfloor\}$. Using property (ii) of the subsets, we conclude,

$$|I(G^\star_q)| \geq 2^k \cdot |T| \geq 2^k \cdot \frac{\beta}{4} \cdot \sum_{j \in G^\star_i} \ell_j \geq 2 \cdot \sum_{j \in G^\star_i} \ell_j \tag{2.3}$$

where we use $k = \lceil \log 8/\beta \rceil$ in the last step.

For $i \geq q$, the contribution of $G^\star_i$ to $I(G^\star_q)$ can be bounded from below as follows:

$$\begin{aligned} C(G^\star_i, I(G^\star_q)) &= \sum_{j \in G^\star_i} \max\{0, |I(G^\star_q) \cap I(j)| - \ell_j\} \\ &\geq \sum_{j \in G^\star_i} \Big(|I(G^\star_q) \cap I(j)| - \ell_j\Big) \geq \left|I(G^\star_i) \cap \bigcup_{j \in G^\star_i} I(j)\right| - \sum_{j \in G^\star_i} \ell_j \\ &= |I(G^\star_i) \cap I(G^\star_q)| - \sum_{j \in G^\star_i} \ell_j. \end{aligned}$$

For $i \geq q$, Equation (2.3) and $I(G_q^\star) \subseteq I(G_i^\star)$ imply $C(G_i^\star, I(G_q^\star)) \geq |I(G_q^\star)|/2$.

We now show that $m > (\lfloor \mu/4 \rfloor - q)/2$. Suppose this is not true. Then $\lfloor \mu/4 \rfloor \geq q + 2m$, and thus, we have at least $2m+1$ disjoint sets $G_i^\star$ such that $C(G_i^\star, I(G_q^\star)) \geq |I(G_q^\star)|/2$. Hence,

$$C\left(\left(\bigcup_{i=q}^{\mu} G_i^\star\right), I(G_q^\star)\right) \geq (2m+1) \cdot \frac{|I(G_q^\star)|}{2} > m \cdot |I(G_q^\star)|.$$

This is a contradiction to Theorem 2.1. We conclude by noting that indeed

$$m \geq \frac{\mu - 4 - 4 \cdot \lceil \log 8/\beta \rceil}{8 + 8/\alpha \cdot \lceil \log 8/\beta \rceil},$$

using $m > (\lfloor \mu/4 \rfloor - q)/2$ and our choice of $q = \lceil 2m/\alpha \rceil \cdot k = \lceil 2m/\alpha \rceil \cdot \lceil \log 8/\beta \rceil$. □

We will utilize Theorem 2.2 to construct an $\mathcal{O}(m \log m)$-machines algorithm. To show that it is an $\mathcal{O}(m)$-machines algorithm for the special cases (laminar or agreeable instances), we use a slightly weaker definition of a (μ, β)-critical pair. We replace the job-individual Condition (ii) in Definition 2.1 by an averaging condition.

Definition 2.2. *Let G be a set of α-tight jobs and let T be a non-empty finite union of time intervals. For some $\mu \in \mathbb{N}$ and $\beta \in (0,1)$, a pair (G,T) is called weakly (μ,β)-critical if*

(i) each $t \in T$ is covered by at least μ distinct jobs in G,

(ii) $|T| \geq \frac{\beta}{\mu} \cdot \sum_{j \in G} \ell_j$.

It is easy to verify that the proofs above apply also to weakly (μ,β)-critical pairs. (Indeed, the only difference is that we do not need the counting argument to obtain Inequality (2.2) in the proof of Lemma 2.2.) Hence, we have the following theorem.

Theorem 2.3. *If there exists a weakly (μ,β)-critical pair, then $m = \Omega\left(\frac{\mu}{\log 1/\beta}\right)$.*

2.5 Reduction to Semi-Online Scheduling

In this section, we consider both semi-online and fully online deadline scheduling, that is, the variants of our problems where the number of machines the offline scheduler uses is an input to the online algorithm and where it is not, respectively. The main result of this section is that we can reduce fully online deadline scheduling to semi-online deadline scheduling at the cost of increasing the number of used machines by a factor of 4, justifying that we will be concerned with semi-online scheduling for the remainder of this chapter.

Theorem 2.4. *Given a ρm-machines algorithm A for semi-online deadline scheduling, there is a $4\rho m$-machines algorithm A' for fully online deadline scheduling.*

We emphasize that this theorem holds for the preemptive, non-migratory, and the non-preemptive setting. That is, if A does not migrate jobs, then neither will A'; if A does not preempt jobs, then neither will A'.

To prove this theorem, we employ the well-known general idea of *doubling* an unknown parameter (for a survey see [CKM06]). Towards describing the algorithm, denote by $\mathsf{A}(m')$ the semi-online algorithm A that is given m' as number of machines. Further, denote by $m(t)$ the minimum number of machines needed to feasibly schedule all jobs released up to time t. Then our algorithm for the fully online problem is as follows.

Algorithm A': Let $t_0 \coloneqq \min_j r_j$. For $i = 1, 2, \ldots$, if $t_{i-1} \neq \infty$:

- Let $t_i \coloneqq \min\{t \mid m(t) > 2m(t_{i-1})\} \cup \{\infty\}$.
- All jobs released within $\in [t_{i-1}, t_i)$ are run by Algorithm $\mathsf{A}(2m(t_{i-1}))$ on a separate set of machines.

Since the time points $t_0, t_1, \ldots$ as well as $m(t_0), m(t_1), \ldots$ can be computed online and A is assumed to be an algorithm for the semi-online problem, this procedure can be executed online. Notice that indeed A' does not preempt or migrate jobs which would not have been preempted or migrated, respectively, by Algorithm A.

We prove that this algorithm indeed only requires $4\rho m$ machines to produce a feasible schedule.

Proof of Theorem 2.4. For any i such that $t_i \neq \infty$, first note that the set S_i of all jobs released within $[t_0, t_{i+1})$ has a feasible schedule on $m(t_{i+1} - 1) \leq 2m(t_i)$ machines. Thus, as a subset of S_i, also the set S_i' of all jobs released within $r_j \in [t_i, t_{i+1})$ has a feasible schedule on $2m(t_i)$ machines. Consequently, by definition of A, $\mathsf{A}(2m(t_i))$ produces a feasible schedule S_i' on $2\rho m(t_i)$ machines.

It remains to compare the number of machines opened by A' with the minimum number of machines m needed to schedule the whole input instance. Let k be maximal such that t_k is defined. Then the total number of machines opened by A' is

$$\sum_{i=0}^{k} 2\rho m(t_i) \leq \sum_{i=0}^{k} \frac{2\rho}{2^{k-i}} \cdot m(t_k) = 2\rho \sum_{i=0}^{k} \frac{1}{2^i} \cdot m(t_k) < 4\rho m,$$

where we use $2m(t_i) \leq m(t_{i+1})$ for all $i \in \{0, \ldots, k-1\}$ in the first step and $m \geq m(t_k)$ in the last step. This concludes the proof. □

In the remainder of the chapter we will thus be concerned with the semi-online problem.

2.6 Migratory Online Algorithms

The focus of this section is on the migratory setting. The main result is a general $\mathcal{O}(m \log m)$-machines algorithm, which is obtained by treating loose and tight jobs separately by the algorithms presented in Subsections 2.6.1 and 2.6.2, respectively. In Subsection 2.6.3, we analyze agreeable and laminar instances, for which we obtain $\mathcal{O}(m)$-machines algorithms each.

2.6.1 Instances Consisting of Loose Jobs

In the following, let $\alpha \in (0, 1)$ be a constant and consider α-loose jobs. For these jobs, it turns out that EDF only requires $\mathcal{O}(m)$ machines. More specifically, we show the following theorem.

Theorem 2.5. *For $\alpha \in (0,1)$ and instances consisting of α-loose jobs,* EDF *is an $m/(1-\alpha)^2$-machines algorithm.*

Towards the proof of this theorem, recall that EDF on m' machines is a busy algorithm, that is, whenever there are at most m' unfinished jobs available, EDF schedules all of them. Similar to the analysis of EDF in the context of speed augmentation given in [PSTW02], we obtain the following helpful lemma for all busy algorithms. Here, we denote by $W^{\mathsf{A}}(t)$ and $W^{\mathsf{OPT}}(t)$ the total workload (released or unreleased) that is remaining for algorithm A and an arbitrary optimum OPT.

Lemma 2.4. *Consider $\alpha \in (0,1)$, instances consisting of α-loose jobs, and a busy online algorithm* A *using $m/(1-\alpha)^2$ machines making decisions only at integer time points. For all $t \leq d_{\max}$, we have*

$$W^{\mathsf{A}}(t) \leq W^{\mathrm{OPT}}(t) + \frac{\alpha}{1-\alpha} \cdot m \cdot (d_{\max} - t)\,.$$

Proof. We prove by induction on integer time points. The base is clear since $W_A(0) = W_{\mathrm{OPT}}(0)$. Now we assume the statement holds for all $t' < t$ and show it for t. We consider three cases. Note that at least one case occurs, and possibly more than one occurs.

Case 1: A processes at least $m/(1-\alpha)$ jobs at time t. According to the induction hypothesis, it holds that

$$W^{\mathsf{A}}(t-1) \leq W^{\mathrm{OPT}}(t-1) + \frac{\alpha}{1-\alpha} \cdot m \cdot (d_{\max} - t + 1). \tag{2.4}$$

Since the optimum can at most finish a workload of m within the time interval $[t, t+1)$, we get $W^{\mathrm{OPT}}(t) \geq W^{\mathrm{OPT}}(t-1) - m$. For Algorithm A, it holds that $W^{\mathsf{A}}(t) \leq W^{\mathsf{A}}(t-1) - m/(1-\alpha)$. Inserting the two latter inequalities into Inequality 2.4 proves the claim.

Case 2: A processes less than $m/(1-\alpha)$ jobs at time t, and we have $p_j(t) \leq \alpha(d_j - t)$ for all unfinished jobs. Again by the facts that A is busy and that there is an empty machine at time t, there are less than $m/(1-\alpha)$ unfinished jobs. Then the total remaining processing time of unfinished jobs is bounded by $\alpha \cdot (d_{\max} - t) \cdot m/(1-\alpha)$. Plugging in the total processing time of unreleased jobs, which is bounded by $W^{\mathrm{OPT}}(t)$, we again get the claim.

Case 3: There exists an unfinished job j with $p_j(t) > \alpha(d_j - t)$. Using that j is α-loose, i.e., $p_j \leq \alpha(d_j - r_j)$, we get that j has not been processed for at least $(1-\alpha)(t - r_j)$ time units in the interval $[r_j, t)$. As Algorithm A is busy, this means that all machines are occupied at these times, yielding

$$\begin{aligned} W^{\mathsf{A}}(t) &\leq W^{\mathsf{A}}(r_j) - (1-\alpha)(t-r_j) \cdot \frac{m}{(1-\alpha)^2} \\ &\leq W^{\mathrm{OPT}}(r_j) + \frac{\alpha}{1-\alpha} \cdot m \cdot (d_{\max} - r_j) - (1-\alpha)(t-r_j) \cdot \frac{m}{(1-\alpha)^2} \\ &\leq W^{\mathrm{OPT}}(r_j) + \frac{\alpha}{1-\alpha} \cdot m \cdot (d_{\max} - t) - m \cdot (t - r_j), \end{aligned}$$

where the second inequality follows by the induction hypothesis for $t = r_j$, and the third one follows by elementary transformations. Lastly, the feasibility of the optimal schedule implies

$$W^{\text{Opt}}(t) \geq W^{\text{Opt}}(r_j) - m \cdot (t - r_j),$$

which in turn implies the claim by plugging it into the former inequality.

This completes the proof. □

Proving the theorem is now simple.

Proof of Theorem 2.5. Suppose that EDF using $m/(1-\alpha)$ machines fails on a feasible instance J. Out of the jobs that EDF fails to schedule, let $j^\star$ be the job with the earliest deadline. Let $J^\star \coloneqq \{j \mid d_j \leq d_{j^\star}\}$ and note that EDF's processing of $J^\star$ is unaffected of the presence of $J \setminus J^\star$. Hence EDF on $J^\star$ still fails to meet the deadline of $j^\star$, allowing us to consider $J^\star$ instead of J from now on. Now we apply Lemma 2.4, showing $W^{\mathsf{A}}(d_{j^\star}) \leq W^{\text{Opt}}(d_{j^\star}) = 0$, meaning that A has finished *all* workload at time $d_{j^\star}$, and in particular it has finished $j^\star$. Hence, A does not miss the deadline of $j^\star$; a contradiction. □

We note that Theorem 2.5 also holds for LLF instead of EDF. The proof can be extended the same way as the proof for the upper bound on the speed requirement of EDF in [PSTW02].

2.6.2 Instances Consisting of Tight Jobs

Knowing that α-loose jobs can be handled on $\mathcal{O}(m)$ machines for any constant α, we can simply treat these jobs on $\mathcal{O}(m)$ separate machines and focus on α-tight jobs, at least if we do not wish to optimize constants and are only interested in the asymptotical guarantee of our algorithm. In this section, we first rule out that EDF and LLF are good algorithms to handle α-tight jobs by showing that they are no $f(m)$-machines algorithm for these instances and any function f. Then we present a new algorithm, which will be shown to be an $\mathcal{O}(m \log m)$-machines algorithm for α-tight jobs. This then yields a $\mathcal{O}(m \log m)$-machines algorithm for the general case.

A Lower Bound for EDF and LLF

We first consider EDF. In the original lower-bound instance [PSTW02], a lot of very loose jobs are used to delay a very tight job. To get a lower-bound instance solely consisting of tight jobs, we replace the very loose jobs with tight jobs that form an geometric structure such that they can be schedule on a single machine.

Theorem 2.6. *For arbitrary $\alpha \in (0,1)$ and α-tight jobs,* EDF *is an $\Omega(n)$-machines algoroithm even if $m = 2$.*

Proof. We describe an instance with not necessarily natural but rational numbers as released dates, deadlines, and processing times, which can later be scaled up to make these parameters natural numbers again. Let $\alpha' > \alpha$ be rational. We construct n jobs, all released at time 0 such that EDF fails to feasibly schedule them on $n-1$ machines.

Among the n jobs, there are $n-1$ jobs jobs forming a geometric structure: The lengths of their intervals are $1/(1-\alpha')$, ..., $1/(1-\alpha')^{n-1}$, and their laxities are 1, $1/(1-\alpha')$, ..., $1/(1-\alpha')^{n-2}$. In addition to these $n-1$ jobs, there is one *critical* job of 0 laxity and the largest deadline d (i.e., $d > 1/(1-\alpha')^{n-1}$). Since the critical job has the largest deadline, EDF decides to postpone it at time 0 and it thus fails to schedule the n jobs on $n-1$ machines. Meanwhile, it is easy to verify these jobs can be feasibly scheduled on 2 machine since, except for the critical job, the other $n-1$ jobs can be feasibly scheduled on 1 machine. □

Using a similar idea, we can also lift up the lower bound for LLF from [PSTW02] to tight jobs.

Theorem 2.7. *For arbitrary $\alpha \in (0,1)$ and α-tight jobs,* LLF *is not an $f(m)$-machines algorithm for any function f even if $m = 2$.*

In this proof, we again give w.l.o.g. rational instead of natural numbers as job parameters. The key is the following lemma, which essentially blocks one machine for a certain period.

Lemma 2.5. *Let $c \in \mathbb{N}$, $\varepsilon \in (0,1)$, and $\alpha \in (0,1)$. Then there is an instance $J_{c,\varepsilon}$ consisting of α-tight jobs and a time $t^\star$ with the following properties:*

(i) There is a feasible schedule $S^\star$ of $J_{c,\varepsilon}$ on 2 machines such that $p_{j'}(t) = 0$ for all $j' \in J_{c,\varepsilon}$.

(ii) At time $t^\star$ in LLF *given c machines, there is exactly one unfinished job $j \in J_{c,\varepsilon}$ with $d_j > t^\star$, and*

$$\frac{\ell_j(t^\star)}{d_j - t^\star} \leq \varepsilon. \tag{2.5}$$

(iii) Further, no job in $J_{c,\varepsilon}$ ever gets zero laxity in LLF.

Proof. Assume w.l.o.g. that ε is rational, and let $\alpha' > \alpha$ be rational. The idea is the following: In $S^\star$, j runs uninterruptedly from 0 through $p_j =: t^\star$ on the first of the two machines (p_j and d_j will be fixed later). In LLF, however, j is delayed by *waves* of jobs, which are run on the other machine in $S^\star$. A wave released at time t of *size* β is defined as follows: There are c jobs released at time t with interval lengths $\beta \cdot (1-\alpha')^{c-1}, \beta \cdot (1-\alpha')^{c-2}, \ldots, \beta$ and laxities $\beta \cdot (1-\alpha')^c, \ldots, \beta \cdot (1-\alpha')$. Note that, indeed, if $\ell_j(t) \geq \beta$ for all $t \leq t^\star$, a wave decreases the laxity of j by $\delta(\beta) \coloneqq \beta \cdot (1-\alpha')^{c-1} - \beta \cdot (1-\alpha')^c > 0$ in LLF. It is also easy to verify that a wave can actually be scheduled on a single machine via EDF.

We now set the parameters of j and the release times of the waves to get Properties (i) and (ii). We set $\ell_j \coloneqq 1$ and wish to have $\ell_j(p_j) \leq \varepsilon$ so as to fulfill Inequality (2.5). To do so, we release $\lceil (1-\varepsilon)/\delta(\varepsilon) \rceil$ waves of size ε. Here, we release the next wave whenever the previous wave is finished in $S^\star$ (on one machine via EDF) and in LLF. If we let p_j be the time when the last wave is finished in $S^\star$ and LLF, Property (i) follows. Since Inequality (2.5) is fulfilled and all waves are finished by LLF at time $t^\star$, also Property (ii) is fulfilled. Note that, indeed, no job in $J_{c,\varepsilon}$ ever gets zero laxity, so Property (iii) is fulfilled as well. □

To prove the theorem, we will now apply the lemma recursively and thereby block an arbitrary number of machines.

Proof of Theorem 2.7. Note that the claim actually follows from a stronger version of Lemma 2.5: Let $c \in \mathbb{N}$, $\varepsilon \in (0,1)$, $\alpha \in (0,1)$, and $k \in \mathbb{N}$ with $k \leq n$. Then there is an instance $J^k_{c,\varepsilon}$ consisting of α-tight jobs and a *critical time* $t^\star$ with the following properties:

(i′) There is a feasible schedule $S^\star$ of $J^k_{c,\varepsilon}$ on 2 machines such that $p_j(t) = 0$ for all $j \in J^k_{c,\varepsilon}$.

(ii′) At time $t^\star$ in LLF given c machines, there are k different unfinished *critical* jobs $j \in J^k_{c,\varepsilon}$ with identical deadlines $d_j > t^\star$ and

$$\frac{\ell_j(t^\star)}{d_j - t^\star} \leq \varepsilon. \tag{2.6}$$

(iii′) Further, no job in $J^k_{c,\varepsilon}$ ever gets zero laxity in the LLF schedule.

We prove the claim by induction on k for arbitrary c, ε, and α. The base case is given by Lemma 2.5. To show that $J^k_{c,\varepsilon}$ as above exists, we first release $J_{c,\varepsilon'}$ for some ε' yet to determine, resulting in a critical moment $t^\star$ and a critical job j. Then, at $t^\star$, we release a scaled-down version of $J^{k-1}_{c-1,\varepsilon}$ such that the deadlines of its critical jobs are exactly d_j. We set ε' such that j's laxity in LLF is always smaller than that of each job in $J^{k-1}_{c-1,\varepsilon}$, which is possible by Property (iii) of $J^{k-1}_{c-1,\varepsilon}$. This means that, in the LLF schedule, j gets a separate machine from time $t^\star$ on, and the claim follows by this fact and the induction hypothesis for $J^{k-1}_{c-1,\varepsilon}$. □

Hence, we cannot use the standard algorithms EDF and LLF to handle tight jobs, and we need a new idea.

A New Algorithm

We open m' machines and will choose m' later appropriately as a function of m.

The idea for our algorithm is the following. We view the laxity of a job as the budget for delaying it. Whenever a job is delayed for some time, then we pay this delay from its budget. If the budget is empty, we must process the job. Greedily decreasing the budget, as LLF does, fails. Instead, we aim at balancing the decrease of the budget by taking the number of currently processing jobs into account. To that end, we partition the total budget of each job into $m' + 1$ equal-size smaller budgets, numbered from 1 to $m' + 1$. That is, a job j has $m' + 1$ budgets, each of size $\ell_j/(m' + 1)$. The i-th budget of a job is reserved for time points when $i - 1$ other jobs (with larger index) are being processed, which means that their corresponding 1-st, 2-nd, $\cdots$, $(i - 1)$-st budgets have become 0. Once $i - 1$ such jobs are already running and the currently considered job has an empty i-th budget, then we process this job even if its other budgets are not empty.

We now describe our algorithm in detail. Figure 2.2 gives an illustration. At a time t, we consider all jobs with a time window covering t and call them *relevant jobs at* t. We must decide which jobs to process and which jobs to delay/preempt. We call the jobs that are processed at time t *active jobs at time* t and denote them by $a_1(t), a_2(t), \ldots$.

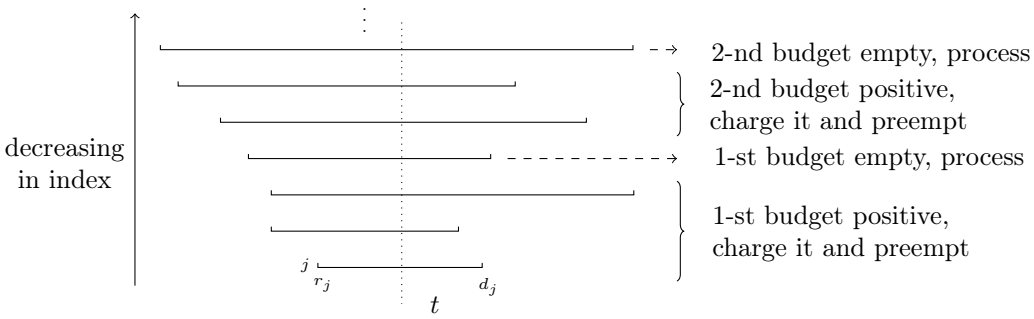

Figure 2.2: Illustration of our algorithm. At time t, we consider all relevant jobs in reverse order of their indices. After having found i active jobs, we check the $(i+1)$-th budget of the current job. If this budget is not empty, we charge it and preempt/delay the job; otherwise the job becomes active.

At any computation time t, which we will specify later, we consider all relevant jobs at t in reverse order of their indices, i.e., in decreasing order of release dates. One after the other, we inspect the 1-st budget of each of these jobs. As long as it is positive, we do not process the corresponding job and charge its 1-st budget by the time duration until the next computation time. Once we find a job whose 1-st budget is 0, this job becomes the first active job $a_1(t)$, and each of its budgets remains unchanged until the next computation time. We continue considering jobs in the reverse order of indices, i.e., we consider jobs with index smaller than $a_1(t)$. Now, we inspect the 2-nd budget of jobs (instead of the 1-st budget; because we have one active job). As long as the 2-nd budgets are non-empty we delay the jobs and charge their 2-nd budgets; once we find a job with an empty 2-nd budget, this job becomes the second active job $a_2(t)$. Then we continue with verifying the 3-rd budgets, etc.

The computation times for our algorithm are release dates, deadlines, and time points at which the remaining budget of some job becomes 0. Since the number of budgets per job is $m'+1$, our algorithm has a polynomial running time if the chosen number of machines m' is polynomial.

If we ever find an $(m'+1)$-th active job, we say that our algorithm *fails*. We will, however, show that we can choose m' such that the algorithm never fails.

We prove the following theorem.

Theorem 2.8. *Our algorithm is an $\mathcal{O}(m \log m)$-machines algorithm.*

By definition of our algorithm, a job's total budget never becomes negative, i.e., it is preempted no longer than its laxity. So we only have to show that our algorithm never finds too many active jobs, i.e., it never finds an $(m'+1)$-th active job. To prove this, we will assume the contrary and will construct a critical pair, from whose existence the theorem then follows by Corollary 2.1.

Lemma 2.6. *If our algorithm fails, then there exists a $(\mu, 1/\mu)$-critical pair where $\mu = m'+1$.*

Proof. As our algorithm fails, there is some time $t^\star$ at which an $(m'+1)$-th active job $j^\star$ is found. We construct a $(\mu, 1/\mu)$-critical pair (F, T) which, intuitively speaking, is a

Algorithm 2.1: Our Algorithm

Input: Set S of α-tight jobs, number of machines m'.
Output: A feasible schedule for S on m' machines **or** *fail*

for all computation times t **do**
  $(j_1, \ldots, j_k) \leftarrow$ list of relevant jobs with $I(j) \ni t$ in order of indices
  $c \leftarrow 1$ `//currently considered budget (depends on number of active jobs)`
  for $i = 1, ..., k$ **do**
    if the c-th budget of j_i is empty **then**
      if $c = m + 1$ **then return** *fail* **end**
      $a_c(t) \leftarrow j_i$
      $c \leftarrow c + 1$
    else
      delay j_i and charge the c-th budget of j_i until the next computation time
    end
  end
end
return the resulting schedule

minimal subset of jobs still causing the failure. More specifically, we have $F = F_1 \cup \cdots \cup F_\mu$ as well as $T = T_1 \cup \cdots \cup T_\mu$ and, from $i = \mu$ down to $i = 1$, we define F_i as well as T_i as follows. We set $F_\mu \coloneqq \{j^\star\}$ and $T_\mu \coloneqq \{t \mid \text{the } \mu\text{-th budget of } j^\star \text{ is charged at } t\}$. Moreover, for all $i = \mu - 1, \ldots, 1$, we define

$$F_i \coloneqq \{j \mid j = a_i(t) \text{ for some } t \in T_{i+1} \cup \cdots \cup T_\mu\}$$
$$\text{and } T_i \coloneqq \{t \mid \text{the } i\text{-th budget of some } j \in F_i \text{ is charged at } t\}.$$

We show that (F, T) is indeed a $(\mu, 1/\mu)$-critical pair. We first show that Condition (i) in Definition 2.1 is satisfied, i.e., each t in T is covered by μ different jobs in F.

By definition of T, for any $t \in T$ there is an i such that $t \in T_i$. Using the definition of T_i, there is some job $j_i \in F_i$ such that its i-th budget is charged at t. Notice that the i-th budget of j_i is charged at time t because there are $i - 1$ active jobs $a_1(t), \ldots, a_{i-1}(t)$, all with larger index than j_i and time intervals covering t. Thus, there are at least i jobs j with $j \geq j_i$ that cover t.

We claim that there are at least $\mu - i$ different jobs j with $j < j_i$ that cover t. Assume this is true, then with the claim above, each t in T is indeed covered by μ different jobs in F.

We now prove the claim by (downward) induction on i. Clearly it holds for $i = \mu$. Assume the claim is true for all $h = i + 1, \ldots, \mu$. We now show it for i: According to the definition of F_i, j_i is the i-th active job at some $t' \in T_k$ for $k > i$, where $t < t'$ because the remaining i-th budget of j_i at t is still positive, whereas it becomes 0 at t'. By the definition of T_k, there also exists a job $j_k \in F_k$ whose k-th budget is charged at time t'. We apply the induction hypothesis for k to obtain that there are $\mu - k$ different jobs j with $j < j_k < j_i$ covering t'. As $j < j_i$ implies $r_j \leq r_{j_i}$ and we have $t < t'$, all these $\mu - k$ jobs also cover t. Similarly j_k also satisfies $j_k < j_i$ and covers t', so it

covers t, too. To finish the proof of the claim, we show that there are $k-i-1$ different jobs j with $j_k < j < j_i$ that cover t. As the k-th budget of j_k is charged at time t', there must exist active jobs $a_{i+1}(t'), \ldots, a_{k-1}(t')$, each of which covers time t', and we have $j_k < a_h(t') < j_i$ for all $h \in \{i+1, \ldots, k-1\}$. Again using $r_{a_h(t')} \leq r_{j_i} \leq t \leq t'$, all of these $k-i-1$ jobs cover time t. Hence in total there are $\mu - i$ different jobs j with $j < j_i$ that cover t. We conclude that each t in T is indeed covered by μ different jobs in F.

Finally, we show that also the second property of a $(\mu, 1/\mu)$-critical pair is fulfilled by (F, T). Indeed, $|T \cap I(j)| > \ell_j/\mu$ holds for each $j \in F$: By definition, there is an i such that $j \in F_i$, and thus j is an i-th active job at some time t. As the i-th budget (which initially amounted to ℓ_j/μ) is exhausted at time t, it follows by definition of T_i that $|T_i \cap I(j)| \geq \ell_j/\mu$, and the lemma follows. □

We are now ready to prove the main theorem of this section.

Proof of Theorem 2.8. We show that our algorithm never fails, i.e., it never finds an $(m'+1)$-st active job, when using $m' = \mathcal{O}(m \log m)$ machines. This is sufficient for proving the theorem, as the algorithm opens m' machines and processes at any time the active jobs. All other jobs have a positive corresponding budget and get preempted/delayed. No job is preempted/delayed for more time than its total laxity (budget).

We assume that the algorithm fails, which implies the existence of a $(\mu, 1/\mu)$-critical pair with $\mu = m' + 1$ by Lemma 2.6. Now, Corollary 2.1 implies $\mu \leq cm \log \mu$ for some constant c. Thus, there exists a constant c' such that $m' \leq c' m \log m'$, i.e., the algorithm fails only if $m' \leq c' m \log m'$. For m' sufficiently large, this inequality is not true. Thus, for $m' = \mathcal{O}(m \log m)$ the algorithm does not fail. □

We remark that careful calculations show that for $m = 2$, our algorithm opens $m' = 236$ machines for tight jobs (taking $\alpha = 4/5$). Including loose jobs (Theorem 2.5), our algorithm requires 261 machines in total. We emphasize that we did not optimize the parameters that we choose in the proofs.

We also remark that we do not have an $\omega(m)$ lower bound for the machine requirement of our algorithm. Indeed, showing the existence of a $(\mu, 1)$-critical pair instead of a $(\mu, 1/\mu)$-critical one (as in Lemma 2.6) in case of failure would suffice to get an $\mathcal{O}(m)$-machines algorithm using the same line of argument in the proof of Theorem 2.8. For laminar and agreeable instances, we are able to show such a stronger version of Lemma 2.6 (with the slight difference that we are considering *weakly* critical pairs).

2.6.3 Agreeable and Laminar Instances

In this subsection we give $\mathcal{O}(m)$-machines algorithms for two important special cases, namely, laminar and agreeable instances. Recall that, in laminar instances, any two jobs j and j' with $I(j) \cap I(j') \neq \emptyset$ satisfy $I(j) \subseteq I(j')$ or $I(j') \subseteq I(j)$. In agreeable instances, $r_j < r_{j'}$ implies $d_j \leq d_{j'}$ for any two jobs j and j'.

Theorem 2.9. *For laminar and agreeable instances, there is an $\mathcal{O}(m)$-machines algorithm.*

Recall that, even for general instances, we can handle α-loose jobs for a constant $\alpha \in (0,1)$ by EDF on $\mathcal{O}(m)$ machines (Subsection 2.6.1). So we can restrict to α-tight jobs again. Interestingly, our new algorithm from Section 2.6.2 turns out to be an $\mathcal{O}(m)$-competitive algorithm for both laminar and agreeable instances consisting of α-tight jobs. Note that, in Section 2.7, we give a simpler $\mathcal{O}(m)$-machines algorithm for laminar instances, which is even non-preemptive.

As in the general case, our proof strategy is to construct a (here: weakly) critical pair whenever our algorithm using $m' = \mu - 1$ machines finds an μ-th active job $j^\star$. To prove a constant competitive ratio, however, we use a slightly different construction in which we drop active jobs with intersecting intervals, and obtain a weakly (μ, β)-critical pair (H, T) where $\beta = 1$. This directly implies[1] the theorem by Corollary 2.1. In fact, the construction is identical for both special cases. We construct job set $H \coloneqq H_1 \cup \cdots \cup H_\mu$ and time points $T \coloneqq T_1 \cup \cdots \cup T_\mu$ by (downward) inductively defining $H_\mu \coloneqq \{j^\star\}$, $T_\mu \coloneqq \{t \mid \text{the } \mu\text{-th budget of } j^\star \text{ is charged at } t\}$,

$$\begin{aligned} F_i &\coloneqq \{j \mid j = a_i(t) \text{ for some } t \in T_{i+1} \cup \cdots \cup T_\mu\}, \\ H_i &\coloneqq \{j \in F_i \mid \text{there does not exist } j' \in F_i \\ &\qquad\qquad \text{such that } I(j) \cap I(j') \neq \emptyset \text{ and } j' < j\}, \\ \text{and } T_i &\coloneqq \{t \mid \text{the } i\text{-th budgetof some } j \in H_i \text{ is charged at } t\}, \end{aligned}$$

for all $i = \mu - 1, \ldots, 1$. Note that the only difference from the construction for the general case is that for each set F_i we additionally maintain a set $H_i \subseteq F_i$ in which we keep for any two intersecting intervals in F_i only one. We call (H, T) the *failure pair*.

To make more concise statements about the structure of the constructed pair, we introduce the notation $S_1 \prec S_2$ (or equivalently, $S_2 \succ S_1$) for two sets of jobs S_1 and S_2. Specifically, this means that for every job $j_2 \in S_2$ there exists a job $j_1 \in S_1$ such that $I(j_1) \cap I(j_2) \neq \emptyset$ and $j_1 < j_2$. Note that $\prec$ is not an order as it does not necessarily obey transitivity.

The following structural lemma is true for both special cases and will be proven later in this subsection.

Lemma 2.7. *Consider a laminar or agreeable instance, and let (H, T) be a failure pair. Then the following statements are true:*

(i) For all $H_i, H_{i'}$ with $i < i'$, we have $H_i \succ H_{i'}$.

(ii) For all H_i, we have $T \subseteq I(H_i)$.

With Lemma 2.7, we can prove Theorem 2.9 without using any further structural information.

Proof of Theorem 2.9. We show that our algorithm never finds a μ-th active job if $\mu - 1 = m' = cm$ for a sufficiently large constant c. To this end, assume the contrary and let (H, T) be the corresponding failure pair, which we claim to be weakly $(\mu, 1)$-critical. Given this claim, the theorem directly follows from Theorem 2.3.

[1] We are dropping constants by writing $m = \Omega(\mu / \log(1/\beta))$ in Theorem 2.3. The logarithm is actually taken over $8/\beta$ instead of $1/\beta$ (see Equation (2.3)), and thus, $\beta = 1$ does not cause a problem in computation.

The first property of a weakly $(\mu, 1)$-critical pair, that is, that each $t \in T$ is covered by μ different jobs in H, is easy to see: By Lemma 2.7 (ii), there is a job in each H_i that covers t. Also, all these jobs are distinct: If there exists a job $j \in H_i \cap H_{i'}$ where $i < i'$, Lemma 2.7 (i) implies the existence of $j' \in H_{i'}$ with $j' < j$ and $I(j) \cap I(j') \neq \emptyset$. This would be a contradiction to the construction of (H, T) as j would not be taken over from $F_{i'}$ to $H_{i'}$ because $j' \in H_{i'}$, $I(j) \cap I(j') \neq \emptyset$, and $j' < j$.

As an intermediate step, we claim that $T_1, \ldots, T_\mu$ are pairwise disjoint. To see this, suppose there exists some $t \in T_i \cap T_{i'}$ where $i < i'$. By definition of T_i, there exists a job $j \in H_i$ such that the i-th budget of j is charged at t. Similarly, there also exists some job whose i'-th budget is charged at t, implying that at time t there exists an i-th active job $a_i(t)$ as $i < i'$. We distinguish three cases, each of them yielding a contradiction:

Case 1: We have $a_i(t) < j$. Note that $a_i(t) \in F_i$ by definition of F_i. As $t \in I(j) \cap I(a_i(t)) \neq \emptyset$, we get a contradiction as, by definition of H_i, it does not include j.

Case 2: We have $a_i(t) = j$. Then the i-th budget of j is already exhausted at t, which is a contradiction to the fact that it is charged at t.

Case 3: We have $a_i(t) > j$. Recall that the algorithm considers jobs one by one in decreasing order of indices. After it found an i-th active job $a_i(t)$ at t, it will never check the i-th budget of jobs of lower indices. Hence the i-th budget of job $j < a_i(t)$ cannot be charged at t.

It remains to show the second property of a weakly $(\mu, 1)$-critical pair, i.e., we have $|T| \geq \sum_{j \in F} \ell_j / \mu$. For each H_i, every $j \in H_i$ is an i-th active job at some time t and thus its i-th budget is exhausted at t. Consequently, there are times at which this budget is charged, implying $|T_i \cap I(j)| \geq \ell_j / \mu$. Using that H_i does not contain distinct jobs j and j' with $I(j) \cap I(j') \neq \emptyset$ (by definition), we obtain

$$|T_i| = \sum_{j \in H_i} |T_i \cap I(j)| \geq \sum_{j \in H_i} \frac{\ell_j}{\mu}.$$

As $T_1, \ldots, T_\mu$ are pairwise disjoint, by summing up these inequalities for all H_i, we get:

$$|T| = \sum_{i=1}^{\mu} |T_i| \geq \sum_{i=1}^{\mu} \sum_{j \in H_i} \frac{\ell_j}{\mu} = \sum_{j \in H} \frac{\ell_j}{\mu},$$

which concludes the proof. □

We will prove Lemma 2.7 separately for the laminar instances as well as agreeable instances. Before we separate the analysis into the two special cases, we state another lemma that is also independent of any structural information and will be useful in both special cases.

Lemma 2.8. *Let (H, T) be a failure pair. Then we have $F_i \succ F_{i+1}$ for all $i \in \{1, \ldots, \mu - 1\}$.*

Proof. Consider an arbitrary $i \in \{1, \ldots, \mu - 1\}$. By the definition of F_i, for any $j \in F_i$ we have $j = a_i(t)$ at some time $t \in T_{i'}$ where $i' > i$. We distinguish two cases and show that in each case there exists a job in F_{i+1} with the desired properies:

Case 1: We have $i' = i + 1$. By definition of T_{i+1}, there is some $j' \in H_{i+1} \subseteq F_{i+1}$ whose $(i+1)$-th budget is charged at time t, implying that $t \in I(j) \cap I(j') \neq \emptyset$. Since the algorithm goes through jobs in decreasing order of indices, it holds that $j' < j$.

Case 2: We have $i' > i + 1$. Using the definition of $T_{i'}$, there is a job $j'' \in H_{i'} \subseteq F_{i'}$ whose i'-th budget is charged at t, implying the existence of an $(i + 1)$-st active job j' at time t. By definition of F_{i+1}, we have $j' \in F_{i+1}$. Notice that $t \in I(j) \cap I(j') \neq \emptyset$. As the algorithm goes through jobs in decreasing order of indices it also follows that $j' < j$.

This completes the proof. □

Before we show Lemma 2.7 for the two special cases, we remark that careful calculations show that, when restricting to only α-tight jobs, we have a $(\lceil 8/\alpha \rceil + 4)m$-machines algorithm for laminar instances, and a $(\lceil 16/\alpha \rceil + 8)m$-machines algorithm for agreeable instances. Note that the factor of two between the ratios is due to that fact that for laminar instances the H_i's we derive already satisfies a laminar structure, i.e., $I(H_1) \subseteq I(H_2) \subseteq \cdots \subseteq I(H_\mu)$, while for agreeable instances we still need to apply Lemma 2.1 to get such a structure. When additionally handling loose jobs by EDF (Theorem 2.5), we derive a 24-competitive algorithm for laminar instances and a 44-competitive algorithm for agreeable instances (by taking $\alpha = 1/2$).

Laminar Instances

Before we prove Lemma 2.7 for the laminar case, we state a simple observation and show three auxiliary lemmas for laminar instances. The following observation directly follows from the way that we index jobs and the laminarity of the instance.

Observation 2.1. *For laminar instances and two jobs j, j', $I(j) \cap I(j') \neq \emptyset$ and $j > j'$ imply $I(j) \subseteq I(j')$.*

Recall the definition of H_i. Observation 2.1 actually implies that the set H_i is constructed by selecting the "maximal" jobs from F_i, i.e., the jobs in F_i which are not included by any other job. The following lemma is also straightforward from the observation.

Lemma 2.9. *For laminar instances, $S_1 \succ S_2$ implies that $I(S_1) \subseteq I(S_2)$.*

Proof. Consider an arbitrary job $j \in S_1$. Due to $S_1 \succ S_2$ there exists $j' \in S_2$ with $j' < j$ and $I(j) \cap I(j') \neq \emptyset$. According to Observation 2.1 we have $I(j) \subseteq I(j')$, hence $I(S_1) \subseteq I(S_2)$. □

Using Observation 2.1, we can also show that the $\prec$-relation on sets of jobs is transitive under laminarity.

Lemma 2.10. *For laminar instances, the relation $\prec$ on sets of jobs is transitive.*

Proof. Let S_1, S_2, and S_3 such that $S_1 \succ S_2$ and $S_2 \succ S_3$. We show that also $S_1 \succ S_3$ holds: For any job $j_1 \in S_1$, there exists a job $j_2 \in S_2$ with $j_1 > j_2$ and $I(j_1) \cap I(j_2) \neq \emptyset$. Further, there is a job $j_3 \in S_3$ with $j_2 > j_3$ and $I(j_2) \cap I(j_3) \neq \emptyset$. Obviously we have $j_1 > j_3$. Further, $I(j_1) \cap I(j_3) \neq \emptyset$ since according to Observation 2.1, we have $I(j_1) \subseteq I(j_2) \subseteq I(j_3)$. Hence, $S_1 \succ S_3$. □

The third auxiliary lemma allows us to induce the relation among H_is from the relation among F_is, as is shown by Lemma 2.8.

Lemma 2.11. *Consider a laminar instance and a failure pair (H, T). Then $F_i \succ F_{i'}$ implies $H_i \succ H_{i'}$.*

Proof. Consider some job $j \in H_i$. By definition of H_i, we also have $j \in F_i$. Due to $F_i \succ F_{i'}$ there is a $j' \in F_{i'}$ such that $j > j'$ and $I(j) \cap I(j') \neq \emptyset$. According to the definition of $H_{i'}$, there is a job $j'' \in H_{i'}$ with $j' \geq j''$ and $I(j') \cap I(j'') \neq \emptyset$ (indeed, job j'' could be j'). Consequently, $j > j' \geq j''$. By Observation 2.1, $I(j) \subseteq I(j') \subseteq I(j'')$ holds, implying that $I(j) \cap I(j'') \neq \emptyset$. Thus, $H_i \succ H_{i'}$. □

Now we are ready to prove Lemma 2.7.

Proof of Lemma 2.7 for laminar instances. We first show (i), i.e., for all $H_i, H_{i'}$ with $i < i'$ we have $H_i \succ H_{i'}$. By Lemma 2.10, it suffices to show $H_i \succ H_{i+1}$ for all $i = 1, \ldots, \mu - 1$. Using Lemma 2.11, we can even restrict to showing $F_i \succ F_{i+1}$ for all $i = 1, \ldots, \mu - 1$. This is exactly the statement of Lemma 2.8.

It remains to show (ii), i.e., $T \subseteq I(H_i)$ for all H_i. Towards this, we first claim that, $I(H_i) = I(F_i)$. It is easy to see that $I(H_i) \subseteq I(F_i)$ as $H_i \subseteq F_i$. Meanwhile, $I(H_i) \supseteq I(F_i)$: Consider an arbitrary job $j \in F_i$. By definition of H_i either $j \in H_i$, or there exists some $j' \in H_i$ such that $I(j) \cap I(j') \neq \emptyset$ and $j > j'$, implying that $I(j) \subseteq I(j')$ by Observation 2.1. Notice that by definition $T \subseteq I(F_1)$. Hence, we have $T \subseteq I(F_1) = I(H_1) \subseteq I(H_i)$, where the last relation follows from (i) and Lemma 2.9. □

Agreeable Instances

We prove Lemma 2.7 for agreeable instances. Recall that

$$H_i \coloneqq \{j \in F_i \mid \text{there does not exist } j' \in F_i \text{ such that } I(j) \cap I(j') \neq \emptyset \text{ and } j' < j\}.$$

For agreeable instances, it is easy to see that the job in F_i with the smallest index (and hence the smallest release date) is always contained in H_i. In the following we will show that, indeed, this is the only job contained in H_i.

Proof of Lemma 2.7 for agreeable instances. We first prove (i), i.e., for all H_i and $H_{i'}$ with $i < i'$ we have $H_i \succ H_{i'}$. To do so, we make the (stronger) claim that, for every $i \in \{1, \ldots, \mu\}$, the following is true: For $k \geq i$ it holds that $H_k = \{j_k\}$, where j_k is the job of smallest index in F_k. Furthermore, $H_k \succ H_{k'}$ holds for any k, k' with $i \leq k < k'$. We prove this claim by (downward) induction on i. For $i = \mu$, this claim is clear from the construction of (H, T).

For $i < \mu$, we first prove that H_i consists of the single job j_i. To this end, suppose there are two jobs $j, j' \in H_i$ where $j < j'$. By the construction of H_i, it holds that $I(j) \cap$

$I(j') = \emptyset$ and hence $d_j \leq r_{j'}$. Since we have $F_i \succ F_{i+1}$ according to Lemma 2.8, there exists a job $j'' \in F_{i+1}$ with $I(j) \cap I(j'') \neq \emptyset$ and $j'' < j$. According to the induction hypothesis $j_{i+1} \leq j'' < j$, implying $d_{j_{i+1}} \leq d_j$. Hence,

$$j_\mu < \cdots < j_{i+1} < j \text{ and } d_{j_\mu} \leq \cdots \leq d_{j_{i+1}} \leq d_j. \tag{2.7}$$

It is easy to see that no time $t \geq d_j$ is covered by any job in $H_{i+1}, \ldots, H_\mu$. Hence, for $t \geq d_j$ we have $t \notin T_{i+1} \cup \cdots \cup T_\mu$. As $d_j \leq r_{j'}$, it follows that $j' \notin H_i$ by definition of H_i, which is a contradiction. Hence, H_i consists of only a single job. Recall that, by definition of H_i, the job j_i of smallest index in F_i is contained in H_i, and thus $H_i = \{j_i\}$.

Next we prove that $H_k \succ H_{k'}$ holds for any k, k' with $i \leq k < k'$. As we have shown (2.7) already, it remains to show that $I(j_i) \cap I(j_h) \neq \emptyset$ for any $h > i$. Suppose this is not true for some h, i.e., $d_{j_h} \leq r_{j_i}$. As j_h is an h-th active job at some time t, there is also an i-th active job at the same time and $a_i(t) \in F_i$ by definition of F_i. Note that, as $I(a_i(t)) \cap I(j_h) \neq \emptyset$, we have $r_{a_i(t)} < d_{j_h} \leq r_{j_i}$. Thus, $a_i(t) < j_i$ holds, contradicting the fact that j_i is the job of smallest index in F_i.

We proceed to proving (ii): For all H_i, we have $T \subseteq I(H_i)$. As (i) implies that $r_1 \geq \cdots \geq r_\mu$ and $d_1 \geq \cdots \geq d_\mu$, it suffices to show that $T \subseteq I(H_1) \cap I(H_\mu) = I(j_1) \cap I(j_\mu)$.

We first show that for all $t \in T$, it holds that $t \in I(j_1)$. By definition of T, the i-th budget of job j_i is charged at t for some $i \in \{1, \ldots, \mu\}$. If $i = 1$, clearly $t \in I(j_1)$. Otherwise, there is a 1-st active job j at time t, and by definition we have $j \in F_1$. If $j = j_1$, again we are done. Otherwise, we have $j_1 < j$, and thus $r_{j_1} \leq r_j$. Now notice that $t \in I(j) \cap I(j_h)$, and thus $r_j \leq t < d_{j_h}$ holds. As $r_{j_1} \leq r_j$ and $d_{j_h} \leq d_{j_1}$ (by (i)), we have $t \in I(j_1)$.

Next, we show that for all $t \in T$, it holds that $t \in I(j_\mu)$. Again by definition of T, at time t the i-th budget of j_i is charged for some $i \in \{1, \ldots, \mu\}$. We prove that for $t \in T_i$, $t \in I(j_\mu)$ by (downward) induction on i. The case $i = \mu$ is obvious. For $i < \mu$ we argue in the following way: As $j_i \in F_i$, it is the i-th active job at some time $t' \in T_{i+1} \cup \cdots \cup T_\mu$. Since its i-th budget is exhausted at time t', while at time t it is still positive, we have $t < t'$. According to the induction hypothesis, $t' \in I(j_\mu)$ holds, and consequently $t < t' \leq d_{j_\mu}$. Further, it follows by (i) that $r_{j_\mu} \leq r_{j_i} \leq t$. Thus, we have $t \in I(j_\mu)$. □

We complement the upper-bound result with the first non-trivial lower bound for agreeable instances. We can even restrict to instances where all jobs have the same processing time.

Theorem 2.10. *There is no $(6 - 2\sqrt{6} - \varepsilon)m$-machines for agreeable instances where all jobs have identical processing times, where $6 - 2\sqrt{6} \approx 1.10$.*

The lower-bound instances have a structure similar to existing ones for general instances [PSTW02, LT99]. W.l.o.g. we again give an instance whose job parameters are not necessarily natural but rational numbers. We say that an algorithm is *behind* by $w \geq 0$ at some time t if the following three properties hold:

(i) An optimal schedule could have finished all jobs released so far at time t.
(ii) The work unfinished by the algorithm is at least w.
(iii) We have $d_j \leq t + 1$ for all jobs released so far.

To prove the theorem, we will iteratively apply the following lemma.

Lemma 2.12. *Consider an online algorithm* A *that uses at most* $(6-2\sqrt{6}-\varepsilon)m$ *machines for some* $\epsilon > 0$*. There is a* $\delta > 0$ *with the following property.*

Let t *be a time at which* A *is behind by* w*. Then jobs can be released (in an agreeable way) such that there is a time* $t' > t$ *at which* A *is behind by* $w + \delta$*.*

Proof. Define $\beta := (6 - 2\sqrt{6} - \epsilon) - 1$. We choose some $\alpha \in [0, 1] \cap \mathbb{Q}$ that is yet to be optimized, and we assume that $\alpha m \in \mathbb{N}$. At time t, we release m jobs j with $p_j = 1$ and $d_j = t + 1 + \alpha$ (type-1 jobs) and αm jobs j with $p_j = 1$ and $d_j = t + 2$ (type-2 jobs). We will now show that there is a $\delta > 0$ such that A is behind by $w+\delta$ at time $t' = t+1+\alpha$. Observe that it suffices to give a lower bound of $w + \delta$ on the work that is left of type-1 jobs at time $t + 1 + \alpha$.

First note that $(1-\alpha)m$ jobs j with $p_j = 1$ and $d_j = t+2$ could be released at time $t+1$ without violating feasibility of the instance. This leaves at most $(\alpha + \beta)m$ machines for processing the type-1 jobs within $[t + 1, t + 1 + \alpha)$, i.e., a total capacity of $(\alpha^2 + \alpha\beta)m$. Thus, the type-2 jobs can receive $(\alpha^2 + \alpha\beta)m + \beta m - w$ of processing within $[t, t + 1)$, amounting to $(\alpha^2 + \alpha\beta)m + \beta m - w + \alpha^2 m$ within $[t, t + 1 + \alpha)$. Consequently, at time $t' = t + 1 + \alpha$, the work left of type-1 jobs is at least

$$\alpha m - ((\alpha^2 + \alpha\beta)m + \beta m - w + \alpha^2 m) = w + \delta.$$

First note that δ is independent of w. Now the requirement $\delta > 0$ is equivalent to

$$\beta < \frac{\alpha - 2\alpha^2}{1 + \alpha}.$$

Maximizing the right-hand side of this inequality over the real numbers yields a maximum of $5 - 2\sqrt{6}$ for $\alpha = (\sqrt{6} - 2)/2 \approx 0.25$. By continuity of the right-hand side and definition of β, we can choose a rational α that makes the inequality true. This proves the lemma. □

We are ready to prove the theorem.

Proof of Theorem 2.10. Suppose there exists some $(6 - 2\sqrt{6} - \varepsilon)m$-machines online algorithm A. Clearly, for the empty instance and at time 0, A is behind by 0. Now applying Lemma 2.12 sufficiently often forces A to be behind by w at some time t where w exceeds the available capacity of $6 - 2\sqrt{6} - \varepsilon$ in $[t, t + 1)$. □

Partitioning into Agreeable and Laminar Instances

Having found $\mathcal{O}(m)$-machines algorithms for both agreeable and laminar instances, it is a natural question whether we can online partition instances into agreeable and laminar subinstances like we are doing for loose and tight jobs in our $\mathcal{O}(m \log m)$-machines algorithm for general instances (Theorem 2.8). Indeed, if we could also partition the instance online into $\mathcal{O}(1)$ subsets that are each either agreeable or laminar, this would directly yield an $\mathcal{O}(m)$-machines algorithm for general instances. Unfortunately, even obtaining $f(m)$ such subsets is not possible if we knew all the jobs in advance, for any function f. Here we use a similar geometric interval structure as in the lower bounds in Subsection 2.6.2.

Theorem 2.11. *Let $m = 1$. For every $k \in \mathbb{N}$ and $\alpha \in (0, 1)$, there is a feasible instance J with k^2 α-tight jobs and the following property. Every partition of J into sub-instances, each of which is either agreeable or laminar, has size at least k.*

Proof. Again, we give an instance that w.l.o.g. has rational numbers as job parameters, and define $\alpha' > \alpha$ be rational. We release k *levels*, each consisting of k jobs. When we say we *release a level within* $[t, t']$, we mean the following: We release jobs at $t, t + \varepsilon \ldots, t + (k-1) \cdot \varepsilon$ for some sufficiently small $\varepsilon > 0$. The respective interval lengths are $(1-\alpha')^{k-1}(t'-t), (t'-t)(1-\alpha')^{k-2}, \ldots, (t'-t)$, and the laxities are $(1-\alpha')^k(t'-t), (t'-t)(1-\alpha')^{k-1}, \ldots, (1-\alpha')(t'-t)$. It is easy to see that a level can be scheduled on a single machine such that the machine is idle during some interval $I \subseteq I(j_k) \setminus I(j_{k-1})$. We call I the level's *idle interval*.

The first level is released within $[0, 1]$. The i-th level is released within the $(i-1)$-st level's idle interval. Note that the whole instance can indeed be scheduled on a single machine. Also note that no two jobs within a level can be contained in the same laminar sub-instance and that no two jobs out of different levels can be contained in the same agreeable sub-instance.

Consider some partition of J into $J_1, \ldots, J_{k-1}$. We apply the pigeon-hole principle to select a sub-instance that is neither laminar nor agreeable: By the pigeon-hole principle, in every level i, there are two jobs assigned to the same sub-instance $J_{\sigma(i)}$. Again by the pigeon-hole principle, there is a sub-instance $J_{i^\star}$ such that $J_{\sigma(i)} = J_{\sigma(i')} = J_{i^\star}$ for $i \neq i'$. Now the fact that it contains two jobs from the same level implies that $J_{i^\star}$ cannot be laminar, while the fact that it contains two jobs from different levels implies that $J_{i^\star}$ cannot be agreeable. □

We note that this result may be quite fragile against slightly modifying the definitions of agreeable and laminar instances: Indeed, a level of our lower-bound construction would not be agreeable any more if we slightly shifted the left borders of their intervals. Hence, it is still plausible that alternative definitions for agreeable and laminar instances exist such that there are $\mathcal{O}(m)$-machines algorithms for them and a partitioning in the above sense is possible.

2.7 Non-Migratory Online Algorithms

We consider non-migratory algorithms in this section. We start by giving our strong lower bound for general instances in 2.7.1, and complement it with upper bounds for loose jobs (Subsection 2.7.2), laminar instances (Subsection 2.7.3) and agreeable instances (Subsection 2.7.4).

2.7.1 General Instances

We prove the following theorem.

Theorem 2.12. *Let $m \geq 13$. For every (deterministic) non-migratory online algorithm, there is an instance on n jobs such that the algorithm requires $\Omega(\log n)$ machines.*

We inductively define an adversarial strategy that forces any non-migratory online algorithm to use an unbounded number of machines while the resulting instance has a migratory schedule on three machines offline. The idea is quite different from standard lower-bound instances [PSTW02, LT99], where the online algorithm is forced to open a new machine because it has simply too much (aggregate) workload left. Here, we instead force the online algorithm to distribute jobs over the machines in such a bad way that assigning a new job to any of the machines would cause too much workload on this machine, forcing the algorithm to open a new machine.

Before giving the technical arguments, we introduce some notations. The latest time when a job j has to be assigned to a machine and the earliest time when it can be finished are denoted by $a_j = r_j + \ell_j$ and $f_j = d_j - \ell_j$, respectively. The following key lemma directly implies a weaker version of our theorem, where we allow migration offline.

Lemma 2.13. *For every non-migratory online algorithm* A *that schedules feasibly and* $k \in \mathbb{N}$*, there is an instance* I_k *with* $\mathcal{O}(2^k)$ *jobs and a critical time* t_0 *such that:*

(i) In the schedule computed by A*, there are at time* t_0 *unfinished critical jobs* $j_1, \dots, j_k \in I_k$ *assigned to* k *different machines.*

(ii) There is a feasible migratory schedule of I_k *on three machines with the following properties. Two machines are idle within* $[t_0, t_0 + \varepsilon)$ *for some* $\varepsilon > 0$*. The other one is continuously idle from* t_0 *on.*

Proof. Clearly, I_1 exists. We define the I_2 depending on constants $\alpha \in (1/2, 1)$ and $\beta \in (0, 1/2)$, which will later be chosen appropriately. For the sake of intuition, α and β can be thought of constants very close to 1 and 0, respectively. At time 0, we release job j_1 with $p_{j_1} = \alpha$ and $d_{j_1} = 1$. From a_{j_1} on, we start additionally releasing *short* jobs: For the i-th short job j, we set $r_j = a_{j_1} + (i-1) \cdot \beta$, $p_j = \alpha\beta$, and $d_j = a_{j_1} + i\beta$. We ensure that our choice of α and β fulfills

$$\left\lfloor \frac{f_{j_1} - a_{j_1}}{\beta} \right\rfloor \cdot \alpha\beta > \ell_{j_1}, \tag{2.8}$$

implying that the total processing time of short jobs that have to be scheduled within $[a_j, f_j) \subsetneq I(j_1)$ will eventually add up to more than the laxity of of j_1. This ensures that A has to schedule some short job on a different machine than j_1. We define j_2 to be the first such job, let $t_0 := a_{j_2}$, and stop releasing jobs then.

Given Equation (2.8), we check (i) and (ii) for critical jobs j_1, j_2 and critical time t_0: By definition of j_1 and j_2, they are assigned to different machines by A. Using $t_0 = a_{j_2} < f_{j_1}$ (implied by Equation (2.8)) and $t_0 = a_{j_2} < f_{j_2}$ (implied by $\alpha > 1/2$), both jobs are not finished at time t_0, that is, we get (i). On the other hand, an offline schedule could simply run job j_1 on one machine and all short jobs on another machine. Since for all jobs j holds $p_j < d_j - r_j$, they can be preempted within some interval $[t_0, t_0 + \varepsilon)$. This shows (ii).

We check that Equation (2.8) can indeed be made true: Using the definition of a_j, f_j, and ℓ_j, we obtain that Equation (2.8) is equivalent to

$$\left\lfloor \frac{2\alpha - 1}{\beta} \right\rfloor \cdot \alpha\beta > 1 - \alpha,$$

which, for instance, is true for $\alpha = 3/4$ and $\beta = 1/4$.

For $k > 2$, we define I_k inductively. Suppose the lemma is true for $k-1$. We start by applying the lemma once, creating $k-1$ critical jobs $j_1, \dots, j_{k-1}$ that are not finished by A and assigned to $k-1$ different machines at the critical time t_0. Also, there is a feasible schedule of I_k on three machines such that machine 1 and 2 are idle within some $[t_0, t_0+\varepsilon)$ with $\varepsilon > 0$ while machine 3 is continuously idle from t_0 on. Recall that $p_j(t)$ is defined to be the remaining processing time of a job j at time t. We define

$$\varepsilon' \coloneqq \min\left\{\varepsilon, p_{j_1}(t_0), \dots, p_{j_{k-1}}(t_0)\right\} > 0 \tag{2.9}$$

to be the largest ε' such that within $[t_0, t_0 + \varepsilon')$ no job j_i for $i \in \{1, \dots, k-1\}$ can get finished by A and machine 1 and 2 can still be idle in a feasible schedule. We apply the lemma with $k-1$ another time in a scaled-down way such that the latest deadline (i.e., that of the job released first) occurring in this sub-instance I'_k is $t_0 + \varepsilon'/2$. We call the corresponding critical jobs and time $j'_1, \dots, j'_{k-1}$ and t'_0, respectively, and distinguish two cases:

Case 1: The jobs $j_1, \dots, j_{k-1}$ and $j'_1, \dots, j'_{k-1}$ are scheduled by A on different sets of machines. Byt using the induction hypothesis, there is a job j'_i such that $j_1, \dots, j_{k-1}, j'_i$ are scheduled on k different machines. By our choice of ε' and the induction hypothesis for I'_k, these jobs are not finished by the critical time t'_0, implying (i). We also show (ii): Again by our choice of ε', only I'_k has to be scheduled within $[t_0, t_0 + \varepsilon'/2)$. By the induction hypothesis regarding I'_k, there is a schedule of this sub-instance with the following property: Two machines are idle within $[t'_0, t'_0 + \varepsilon'')$ for some $\varepsilon'' > 0$ and the other machine is continuously idle from t'_0 on. Since we can again choose to leave machine 3 continuously idle from t'_0 on, we get (ii).

Case 2: The jobs $j_1, \dots, j_{k-1}$ and $j'_1, \dots, j'_{k-1}$ are scheduled by A on the same set of machines. Then we additionally release exactly one job $j^\star$ at time t'_0 with deadline $t_0 + \varepsilon'$. We choose

$$p_{j^\star} \in \Big(t_0 + \varepsilon' - t'_0 - \min_{i=1,\dots,k-1} p_{j'_i}(t'_0), \\ t_0 + \varepsilon' - t'_0\Big),$$

ensuring that $j^\star$ cannot be scheduled on the same machine as any j'_i for $i \in \{1, \dots, k-1\}$ (lower bound on $p_{j^\star}$) and that it has strictly positive initial laxity (upper bound on $p_{j^\star}$). We further make sure that

$$p_{j^\star} > t_0 + \frac{\varepsilon'}{2} - t'_0,$$

meaning that $j^\star$ cannot be finished at time $t_0 + \varepsilon'/2$ (note that $p_{j^\star}$ still exists since $t_0 + \varepsilon'/2 - t'_0 < t_0 + \varepsilon' - t'_0$). We release no further jobs and claim that (i) and (ii) hold with critical jobs $j_1, \dots, j_{k-1}, j^\star$ and critical time $t''_0 \coloneqq t_0 + \varepsilon'/2$.

We first show (i): Consider the schedule computed by A. By the induction hypotheses for I_{k-1} and our choice of ε', at time t''_0 the jobs $j_1, \dots, j_{k-1}$

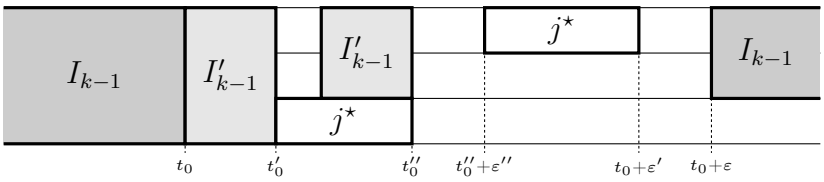

Figure 2.3: The optimal schedule constructed in Case 2 of the proof of Lemma 2.13. Recall $t''_0 = t_0 + \varepsilon'/2$.

are not finished and scheduled on $k-1$ different machines. Using the assumption that $j'_1, \ldots, j'_{k-1}$ are scheduled on exactly the same set of machines as $j_1, \ldots, j_{k-1}$ and the fact that $j^\star$ cannot be scheduled on the same machine as any j'_i for $i \in \{1, \ldots, k-1\}$, the jobs $j_1, \ldots, j_{k-1}, j^\star$ are scheduled on k different machines. By construction of $j^\star$, it is not finished by time t''_0 either.

To create a schedule obeying the condition in (ii), we schedule the whole instance except for $j^\star$ according to Part (ii) of the induction hypothesis (for both I_{k-1} and I'_{k-1}). Using that the latest deadline in I'_{k-1} is $t_0 + \varepsilon'/2$, in this schedule, machine 3 is continuously idle from t'_0 on and machine 1 (and 2) within $[t_0 + \varepsilon'/2, t_0 + \varepsilon')$. We schedule $j^\star$ within $[t'_0, t_0 + \varepsilon'/2)$ on machine 3 and afterwards on machine 1, starting it on machine 1 as late as possible. Since $j^\star$ has positive initial laxity, this again leaves an idle period $[t''_0, t''_0 + \varepsilon'')$ for $\varepsilon'' > 0$ on machines 1 and 2. Also, machine 3 is continuously idle from time t''_0 on, meeting the requirements of (ii). We illustrate this schedule in Figure 2.3.

We finally note that we have indeed $\mathcal{O}(2^k)$ jobs in the resulting instance: I_2 has a constant number of jobs, and for I_k we release I_{k-1} twice plus at most one additional job, showing that I_k has $\mathcal{O}(2^k)$ jobs. □

To relate this result to a non-migratory offline solution, we apply the following theorem.

Theorem 2.13 (Kalyanasundaram and Pruhs [KP01])**.** *Any feasible migratory schedule on m machines can be turned (offline) into a feasible non-migratory schedule on $6m-5$ machines.*

Indeed, the theorem now follows easily.

Proof of Theorem 2.12. Consider some non-migratory online algorithm A. Using Lemma 2.13, there is an instance consisting of n jobs and having a migratory schedule on 3 machines such that A requires $\Omega(\log n)$ jobs to schedule it. By Lemma 2.13, this instance has a non-migratory schedule on $6 \cdot 3 - 5 = 13$ machines, implying the theorem. □

2.7.2 Instances Consisting of Loose Jobs

In this section, we consider instances consisting of loose jobs, i.e., jobs j with $p_j \leq \alpha \cdot (d_j - r_j)$ for arbitrary constant $\alpha \in (0, 1)$. The main theorem of this section is the following.

Theorem 2.14. *For any fixed $\alpha \in (0,1)$, there is a non-migratory $\mathcal{O}(m)$-machines algorithm for instances consisting only of α-loose jobs.*

The key to proving the result is a reduction to a speed-augmentation problem on m machines.

Theorem 2.15. *Suppose there is a non-migratory online algorithm* A *that producues feasible schedules on $f(m)$ speed-s machines for general instances. Then, for every fixed $\alpha \in (0, 1/s)$, there is a non-migratory online $f(\mathcal{O}(m))$-machines (unit-speed) algorithm.*

The proof idea is as follows. To schedule α-loose jobs (for $\alpha < 1/s$) without additional speed, we increase the processing time of each job from the input instance by a factor of s. We denote the set of jobs with modified processing times by J^s. Note that the resulting instance is feasible in the sense that $p_j \leq d_j - r_j$ for all jobs j. We apply the speed-s algorithm to this instance. Whenever a job is scheduled in the resulting schedule, our algorithm schedules the corresponding job from the original instance on the respective machine.

To analyze our algorithm, we have to relate the minimum numbers of machines needed for scheduling the original instance J and for scheduling the instance with increased processing times J^s, respectively. To do so, we introduce two auxiliary job types. For some $\gamma \in (0,1)$ and every job j, we define two jobs $j^{\triangleleft}_\gamma, j^{\triangleright}_\gamma$ with

$$\begin{aligned} I(j^{\triangleleft}_\gamma) &\coloneqq [r_j, d_j - \gamma\ell_j),\ p_{j^{\triangleleft}_\gamma} \coloneqq p_j \\ \text{and } I(j^{\triangleright}_\gamma) &\coloneqq [r_j + \gamma\ell_j, d_j),\ p_{j^{\triangleright}_\gamma} \coloneqq p_j, \end{aligned}$$

that is, we remove a γ-fraction of the laxity from either side of j's feasible time window. We further define $J^{\triangleleft}_\gamma \coloneqq \{j^{\triangleleft}_\gamma \mid j \in J\}$ and $J^{\triangleright}_\gamma \coloneqq \{j^{\triangleright}_\gamma \mid j \in J\}$. The following lemma relates the number of machines needed for scheduling J to that needed for scheduling $J^{\triangleleft}_\gamma$ and $J^{\triangleright}_\gamma$, and it may be of independent interest. To get a cleaner statement, we first consider the migratory setting. Recall that, by $m(S)$, we denote the minimum number of machines needed to schedule an instance S.

Lemma 2.14. *Consider the migratory setting. For every instance J and $\gamma \in (0,1)$, we have*

$$\begin{aligned} m(J^{\triangleleft}_\gamma) &\leq \frac{1}{1-\gamma} \cdot m(J) + 1 \\ \textit{and}\ \ m(J^{\triangleright}_\gamma) &\leq \frac{1}{1-\gamma} \cdot m(J) + 1. \end{aligned} \tag{2.10}$$

Proof. We show the statement for $J^{\triangleright}_\gamma$, and the proof for $J^{\triangleleft}_\gamma$ is symmetric. According to Theorem 2.1, there exists a finite union of intervals I such that $m(J^{\triangleright}_\gamma) = \lceil C(J^{\triangleright}_\gamma, I)/|I| \rceil$. Without loss of generality, we can assume $I = [g_1, h_1) \cup \cdots \cup [g_k, h_k)$ with $h_i < g_{i+1}$ for all $i = 1, \ldots, k-1$. To derive the relationship between $m(J^{\triangleright}_\gamma)$ and $m(J)$, we expand I to a superset (and also finite union of intervals) $\operatorname{ex}(I)$ such that $|\operatorname{ex}(I)| = |I|/(1-\gamma)$. We will show for each job $j \in J$ with $C(j^{\triangleright}_\gamma, I) > 0$ that $C(j, \operatorname{ex}(I)) \geq C(j^{\triangleright}_\gamma, I)$. This will

imply

$$\begin{aligned} m(J_\gamma^\triangleright) - 1 &\leq \frac{C(J_\gamma^\triangleright, I)}{|I|} = \frac{\sum_{j \in J} C(j_\gamma^\triangleright, I)}{|I|} \\ &\leq \frac{\sum_{j \in J} C(j, \operatorname{ex}(I))}{(1-\gamma) \cdot |\operatorname{ex}(I)|} = \frac{1}{1-\gamma} \cdot \frac{C(J, \operatorname{ex}(I))}{|\operatorname{ex}(I)|} \\ &\leq \frac{1}{1-\gamma} \cdot m(J), \end{aligned}$$

where we use the other direction of Theorem 2.1 in the last step. Hence, Inequality (2.10) will follow.

The expanding works as follows. Given I as above, we expand each of the intervals $[g_i, h_i)$ to $[g_i', h_i)$ with $g_i' \leq g_i$ and obtain $\operatorname{ex}(I) \coloneqq [g_1', h_1) \cup \dots \cup [g_k', h_k)$. We start at the rightmost interval $[g_k', h_k)$ and try to set $g_k' \coloneqq h_k - (h_k - g_k)/(1-\gamma)$. If this would, however, produce an overlap between $[g_k', h_k)$ and $[g_{k-1}, h_{k-1})$, we set $g_k' \coloneqq h_{k-1}$ as well as $\delta_k \coloneqq h_{k-1} - (h_k - (h_k - g_k)/(1-\gamma))$ and try to additionally expand $[g_{k-1}, h_{k-1})$ by δ_k instead. After that, we continue this procedure until $i = 1$. More formally, we let $h_0 = -\infty$ as well as $\delta_{k+1} = 0$, and for all $i = k, \dots, 1$ we set

$$g_i' \coloneqq \max\left\{ h_{i-1}, h_i - \left(\frac{h_i - g_i}{1-\gamma} + \delta_{i-1} \right) \right\}$$

$$\text{and } \delta_i \coloneqq \max\left\{ 0, h_{i-1} - \left(h_i - \left(\frac{h_i - g_i}{1-\gamma} + \delta_{i-1} \right) \right) \right\}.$$

Obviously, indeed $|\operatorname{ex}(I)| = |I|/(1-\gamma)$.

We show $C(j, \operatorname{ex}(I)) \geq C(j_\gamma^\triangleright, I)$ for all $j \in J$ with $C(j_\gamma^\triangleright, I) > 0$. By the definition of contribution, we have

$$\begin{aligned} C(j, \operatorname{ex}(I)) &= \max\{0, |\operatorname{ex}(I) \cap I(j)| - \ell_j\} \\ \text{and} \quad C(j_\gamma^\triangleright, I) &= \max\{0, |I \cap I(j_\gamma^\triangleright)| - \ell_{j_\gamma^\triangleright}\} \\ &= \max\{0, |I \cap I(j_\gamma^\triangleright)| - (1-\gamma) \cdot \ell_j\}, \end{aligned}$$

that is, we can restrict to showing $|\operatorname{ex}(I) \cap I(j)| \geq |I \cap I(j_\gamma^\triangleright)| + \gamma \ell_j$. Next, we define $I' \coloneqq I \cap I(j_\gamma^\triangleright)$. Using the fact that $I' \subseteq I$ implies $\operatorname{ex}(I') \subseteq \operatorname{ex}(I)$, it even suffices to show

$$|\operatorname{ex}(I') \cap I(j)| \geq |I'| + \gamma \ell_j. \tag{2.11}$$

To see this, we distinguish two cases:

Case 1: The leftmost point of $\operatorname{ex}(I')$ is left of r_j. Since the leftmost point of I' was not left of $r_j + \gamma \ell_j$, $[r_j, r_j + \gamma \ell_j)$ is a subset of $\operatorname{ex}(I')$. Consequently, we have $|\operatorname{ex}(I') \cap I(j)| - |I' \cap I(j)| \geq \gamma \ell_j$. Using $|I' \cap I(j)| = |I'|$, the claim follows.

Case 2: The leftmost point of $\operatorname{ex}(I')$ is not left of r_j. By the way our expanding works, we have that $\operatorname{ex}(I')$ takes up a length of $|I' \cap I(j)|/(1-\gamma)$ within $I(j)$. Plugging $|I'| > \ell_{j_\gamma^\triangleright} = (1-\gamma) \cdot \ell_j$, which follows since $C(j_\gamma^\triangleright, I) > 0$, into that, we also obtain $|\operatorname{ex}(I') \cap I(j)| - |I' \cap I(j)| \geq \gamma \ell_j$. Again using $|I' \cap I(j)| = |I'|$, the claim follows.

This completes the proof. □

We apply this lemma to get a similar statement about jobs with an in increased processing time, J^s, which might be of independent interest. Formally, for $j \in J$, let j^s denote a copy of job j with a processing time increased by a factor s, and $J^s \coloneqq \{j^s \mid j \in J\}$. The following lemma holds for both the migratory and non-migratory setting.

Lemma 2.15. *Consider the migratory or the non-migratory setting and let $s \geq 1$. For every instance J consisting only of α-loose jobs for some fixed $\alpha \in (0, 1/s)$, we have $m(J^s) = \mathcal{O}(m(J))$.*

Proof. First consider the migratory setting. We investigate the number of machines required to schedule J^s: To do so, we define

$$\delta \coloneqq \frac{1 - \alpha s}{\lceil s \rceil} \cdot (d_j - r_j).$$

Note that $\ell_{j^s} \geq (1 - \alpha s) \cdot (d_j - r_j) > 0$ and thus

$$0 < \delta \leq \frac{\ell_{j^s}}{\lceil s \rceil}. \tag{2.12}$$

Further, for every job $j \in J$, we define $\lceil s \rceil$ different jobs $j_1, \ldots, j_{\lceil s \rceil}$ with

$$\begin{aligned} I(j_i) &\coloneqq [r_j + (i-1) \cdot (p_j + \delta),\ r_j + i \cdot (p_j + \delta)), \\ p_{j_i} &\coloneqq p_j \\ &\quad \text{for all } i < \lceil s \rceil \\ \text{and } I(j_{\lceil s \rceil}) &\coloneqq [r_j + (\lceil s \rceil - 1) \cdot (p_j + \delta),\ r_j + s \cdot p_j + \lceil s \rceil \delta)), \\ p_{j_{\lceil s \rceil}} &\coloneqq (s - \lceil s \rceil + 1) \cdot p_j. \end{aligned}$$

Furthermore let $J_i \coloneqq \{j_i \mid j \in J\}$ for all i. Now note that, for showing $m(J^s) = \mathcal{O}(m(J))$, it suffices to show that $m(J_i) = \mathcal{O}(m(J))$ for all i. This is because we have $I(j_i) \subseteq I(j^s)$ for all i, $I(j_i) \cap I(j_{i'}) \neq \emptyset$ for all i, i', and $\sum_i p_{j_i} = p_{j^s}$, which can be easily checked. Thus any feasible schedule of $J_1, \ldots, J_{\lceil s \rceil}$ on $\mathcal{O}(m(J))$ machines each can be transformed into a feasible schedule of J^s on $\lceil s \rceil \cdot \mathcal{O}(m(J)) = \mathcal{O}(m(J))$ by scheduling each j^s whenever any j_i is scheduled in the respective schedule.

We show that indeed $m(J_i) = \mathcal{O}(m(J))$ for all i. We first show this for $i < \lceil s \rceil$: Note that, using Inequality 2.12, we have $\ell_{j_i} = \delta = \beta \ell_j > 0$ for all $j \in J$ and constant $\beta > 0$. Thus, by applying Lemma 2.14 up to two times to shorten the time window from both sides, $m(J_i) = \mathcal{O}(m(J))$. For $i = \lceil s \rceil$, first note that the above reasoning also works for $J'_{\lceil s \rceil} \coloneqq \{j'_{\lceil s \rceil} \mid j \in J\}$ with

$$\begin{aligned} I(j'_{\lceil s \rceil}) &\coloneqq [r_j + (s-1) \cdot p_j + (\lceil s \rceil - 1) \cdot \delta,\ r_j + s \cdot p_j + \lceil s \rceil \delta)), \\ p_{j'_{\lceil s \rceil}} &\coloneqq p_j, \end{aligned}$$

that is, $m(J'_{\lceil s \rceil}) = \mathcal{O}(m(J))$. Now clearly any feasible schedule of $J'_{\lceil s \rceil}$ can be transformed into one of $J_{\lceil s \rceil}$ without increasing the number of machines by scheduling each $j_{\lceil s \rceil}$ whenever $j'_{\lceil s \rceil}$ is scheduled unless $j_{\lceil s \rceil}$ is finished, implying $m(J_{\lceil s \rceil}) \leq m(J'_{\lceil s \rceil}) = \mathcal{O}(m(J))$.

To get the result for the non-migratory setting, we apply Theorem 2.13, increasing the constant hidden by the $\mathcal{O}$ notation. □

We now apply the preceding lemma in the proof of Theorem 2.15.

Proof of Theorem 2.15. To schedule J only having α-loose jobs with constant $\alpha < 1/s$, we first transform J into J^s. Using Lemma 2.15, by applying A to J^s, we obtain a non-migratory schedule of J^s on $f(m(J^s)) = f(\mathcal{O}(m(J)))$ speed-s processors. We transform this schedule into a non-migratory schedule of J on $f(\mathcal{O}(m(J)))$ speed-1 machines by replacing each j^s by j for all j, which does not violate feasibility by the definition of j^s. □

We are now ready to utilize an algorithm due to Chan, Lam, and To [CLT05] for which they prove an upper bound on the required number of machines that scales with the given speed – even for speeds arbitrarily close to 1.

Theorem 2.16 (Chan, Lam, and To [CLT05])**.** *For every $\varepsilon > 0$, there is a non-migratory online algorithm that produces feasible schedules on $\lceil (1+1/\varepsilon)^2 \rceil$ speed-$(1+\varepsilon)^2$ machines.*

Indeed, it suffices to plug Theorem 2.16 as a black box into Theorem 2.15 to obtain Theorem 2.14.

2.7.3 Laminar Instances

In this section, we consider laminar instances. Recall that this means that, for any two jobs j, j' with $I(j) \cap I(j') \neq \emptyset$, it holds that $I(j) \subseteq I(j')$ or $I(j') \supseteq I(j)$. We prove the following result.

Theorem 2.17. *There exists a non-migratory $\mathcal{O}(m \log m)$-machines algorithm for laminar instances.*

It suffices to consider α-tight jobs for some fixed $\alpha \in (0, 1)$. The remaining α-loose jobs are scheduled on a separate set of $\mathcal{O}(m)$ machines as described in the previous subsection (Theorem 2.14).

Job Assignment. We open m' machines and will later see that we can choose $m' = \mathcal{O}(m \log m)$. At every release date, the arriving jobs are immediately assigned to their machines in order of job indices (i.e., in decreasing order of deadlines). Each job is assigned to exactly one machine, that is, we obtain a non-migratory schedule. Consider some job j. If there is a machine that has no job j' with $I(j) \cap I(j') \neq \emptyset$ assigned to it, we assign j to any such machine. Otherwise we execute the following procedure.

Consider any machine and the jobs j' assigned to it with $I(j) \cap I(j') \neq \emptyset$. Based on the laminarity of the instance and the order in which we assign jobs, all these j' dominate j and are ordered linearly by $\prec$. Consequently, there exists a unique $\prec$-minimal job among them, which is said to be currently *responsible* on the considered machine. Consider the set of currently responsible jobs of *all* machines. Again, by laminarity of the instance, these jobs form a chain $c_1(j) \prec \cdots \prec c_{m'}(j)$. We call $c_i(j)$ the i-th *candidate* of j.

We now want to select a candidate $c_i(j)$ and assign j to the same machine as $c_i(j)$. Consider the laxity of a candidate, which we view (as in Subsection 2.6.2) *budget* for delaying the jobs. Since $I(j) \subseteq I(c_i(j))$, it is a necessary criterion that the budget of $c_i(j)$ suffices to schedule both j and the jobs j' with $I(j') \subseteq I(c_i(j))$ that have been assigned to the same machine earlier. Intuitively, we would also like to minimize the candidate that

we pick w.r.t. $\prec$ so as to save the budget of jobs with larger time windows. However, it fails to greedily assign jobs to the machine of their $\prec$-minimal candidate that fulfills the above necessary criterion. This can be shown using the instance given in Theorem 2.7.

Instead we use a more sophisticated balancing scheme similar to that used for the algorithm in Section 2.6.2: We partition the budget of each candidate into m' equally-sized (sub-)budgets. For every i, we only consider the i-th budget of $c_i(j)$ when assigning j. When picking $c_i(j)$ and assigning j to the same machine, we charge not just p_j but $|I(j)| \geq p_j$ to the i-th budget of $c_i(j)$. This policy ensures that, as long as no budget becomes negative, there is a feasible schedule under the current assignment (Lemma 2.16).

We make this more formal: We call a job j' that has been assigned to the machine of its h-th candidate $c_h(j')$ the h-th *user* of $c_h(j')$. We denote the set of all h-th users of a (candidate) job j' by $U_h(j')$, for all h. Note that at any time the h-th budget of j' has been charged exactly $|I(j'')|$ for all its users j''. To assign j, we select the smallest i such that the i-th budget of $c_i(j)$ can still pay for $|I(j)|$, that is,

$$\frac{\ell_{c_i(j)}}{m'} - \sum_{j' \in U_i(c_i(j))} |I(j')| \geq |I(j)|. \tag{2.13}$$

If we find such an i, we assign j to $c_i(j)$ (and thereby add j to $U_i(c_i(j))$ and charge $|I(j)|$ to the i-th budget of $c_i(j)$). If we do not find such an i, the assignment of j *fails*. In the analysis, we will show that the latter case will never happen if m' is chosen large enough.

Scheduling. At any time and on each machine, we process an arbitrary unfinished job assigned to it with minimum deadline. It will later turn out (Lemma 2.16) that there is always a *unique* such job.

We first show that our algorithm obtains a feasible schedule if no job assignment has failed. Then we give a proof of the fact that the job assignment never fails on instances that admit a feasible schedule. As a byproduct, we get a simplification of our scheduling rule.

Lemma 2.16. *Suppose the job assignment does not fail for any job. We have the following two properties.*

(i) Our algorithm produces a feasible schedule.

(ii) At any time and on each machine, there are no two unfinished jobs with the same deadline.

Proof. Consider some machine. We first show (ii) by induction on the jobs assigned to this machine in order of their indices. The induction base is clear. So assume the statement holds until some job j gets assigned to the considered machine.

The statement is obviously fulfilled if no other unfinished job on the considered machine has deadline d_j. So suppose $j^\star$ with $d_{j^\star} = d_j$ exists. We note that then the candidate j' of j must (also) have deadline d_j: Otherwise, using the laminarity of the instance, we have $d_{j^\star} < d_{j'}$ and thus $j' \succ j^\star$ and j' cannot be the $\prec$-minimal job whose interval contains r_j. Since j is assigned to the machine of its candidate j', one sub-budget and thus the total budget of j' must be at least $|I(j)|$ right before j is assigned. Consider the state of j' and its users at this time. Using Part (ii) of the induction hypothesis, j'

was so far only preempted at some time t when there was a user j'' of it with $t \in I(j'')$. Thus, it was processed for at least

$$\begin{aligned}&\big(|I(j')| - (d_{j'} - r_j)\big) - \sum_{i=1}^{m'} |I(U_i(j'))| \\ \geq\ &\big(|I(j')| - (d_{j'} - r_j)\big) - \big(\ell_{j'} - |I(j)|\big) \\ =\ &|I(j')| - \ell_{j'} = p_{j'},\end{aligned}$$

using $d_j = d_{j'}$ in the second step. Hence j' is finished at time r_j, a contradiction.

To see (i), consider some job j and note that, by applying (ii), whenever it is preempted at some time t, there must be a user j' of j with $t \in I(j')$. According to Inequality (2.13), j is thus preempted for no longer than

$$\sum_{i=1}^{m'} |I(U_i(j))| \leq \sum_{i=1}^{m'} \sum_{j' \in U_i(j)} |I(j')| \leq m' \cdot \frac{\ell_j}{m'} = \ell_j.$$

Therefore, j can be processed for p_j time units. □

It remains to show that the job assignment never fails for some $m' = \mathcal{O}(m \log m)$. Similar to our arguments in Subsections 2.6.2 and 2.6.3, we will assume that our algorithm fails, and we will select a set of jobs and a set of time points, which will be shown to form a critical pair, allowing us to apply Theorem 2.2. We initialize $G_{m'} \coloneqq \{j^\star\}$. Given G_i, we construct F_i and G_{i-1} in the following way. First, F_i is defined to be the set of all $\prec$-maximal i-th candidates of jobs in G_i. Subsequently, G_{i-1} is constructed by adding all the i-th users of jobs in F_i to G_i. Formally,

$$\begin{aligned}F_i &\coloneqq M_\prec\big(\{c_i(j) \mid j \in G_i\}\big) \\ \text{and } G_{i-1} &\coloneqq G_i \cup \bigcup_{j \in F_i} U_i(j),\end{aligned}$$

where $M_\prec$ is the operator that picks out the $\prec$-maximal elements from a set of jobs: $M_\prec(S) \coloneqq \{j \in S \mid \nexists j' : j \prec j'\}$. After m' such iterations, that is, when $G_0, F_1, \ldots, F_{m'}$ have been computed, we set $F_0 \coloneqq M_\prec(G_0)$.

We define

$$F \coloneqq F_0 \cup F_1 \cup \cdots \cup F_{m'} \text{ and } T \coloneqq \bigcup_{j \in F_0} I(j),$$

where we call (F, T) a *failure pair*. As before, we show that . But first we establish some structural properties of (F, T) that will be useful later. Towards this, we define the following notation. For two job sets S_1 and S_2, we write $S_1 \prec S_2$ if, for each $j_1 \in S_1$, there is a $j_2 \in S_2$ with $j_1 \prec j_2$.

Lemma 2.17. *For any failure pair (F, T), we have the following structural properties:*

(i) We have $F_0 \prec \cdots \prec F_{m'}$.

(ii) The sets $F_0, \ldots, F_{m'}$ are pairwise disjoint.

Proof. To see (i), we define $C_i \coloneqq \{c_i(j) \mid j \in G_i\}$, for every $1 \leq i \leq m'$, and $C_0 \coloneqq G_0$. We first show $C_{i-1} \prec C_i$ for every $1 \leq i \leq m'$. For $i = 1$, this directly follows from the construction. Consider $2 \leq i \leq m'$. Let j be an arbitrary job in C_{i-1}, which is then an $(i-1)$-th candidate of some job in G_{i-1}, say, job j'. According to the construction, we have $j' \in G_i$, or j' is the i-th user of some job in $F_i \subseteq C_i$. In both cases, C_i contains the i-th candidate of job j', which dominates job j. Hence, $C_{i-1} \prec C_i$. Since F_{i-1} and F_i are obtained from C_{i-1} and C_i only by deleting dominated jobs, $F_{i-1} \prec F_i$ follows.

Consider property (ii) and suppose there is some $j \in F_i \cap F_{i'}$ for some $i < i'$. Then, by (i), there is also a job $j' \in F_{i'}$ with $j \prec j'$, contradicting the fact that $F_{i'}$ only contains $\prec$-maximal elements. □

We are ready to show that (F, T) is indeed a critical pair.

Lemma 2.18. *Any failure pair (F, T) is a (μ, β)-critical pair for $\mu = m'$, $\beta = 1/m'$.*

Proof of Lemma 2.18. We first consider Property (i) of a (μ, β)-critical pair. Given Lemma 2.17, it is obvious that every $t \in I(j)$ for $j \in F_0$ is covered by at least m' distinct jobs.

Concerning Property (ii) of a (μ, β)-critical pair, we have

$$T = \bigcup_{j \in F_0} I(j) = \bigcup_{j \in M_{\prec}(G_0)} I(j) = \bigcup_{j \in G_0} I(j)$$

according to the definition of T. Furthermore, G_0 contains all the the i-th users of jobs in F_i, hence $|T \cap I(j)| \geq \ell_j/m'$ for any $j \in F_i$. □

We now use Lemma 2.18 and Corollary 2.1 to get an upper bound on m'.

Proof of Theorem 2.17. We show that, for a sufficiently large constant c and $m' = cm \log m$, our algorithm always produces feasible schedules. According to Lemma 2.16, it suffices to show that the job assignment procedure never fails. Suppose it does fail. Then we derive the witness set (F, T), which is a $(m', 1/m')$-critical pair (G, T) by Lemma 2.18. Using Corollary 2.1 and the fact that every feasible schedule in the non-migratory setting is also a feasible schedule in the migratory setting, we get $m = \Omega\left(\mu/\log 1/\beta\right) = \Omega(m'/\log m')$, which is a contradiction if $m' = cm \log m$ for sufficiently large c, that is, for some $m' = \mathcal{O}(m \log m)$ the algorithm never fails and thus always produces feasible schedules. □

2.7.4 Agreeable Instances

In this section, we consider agreeable instances. Recall that then $r_j < r_{j'}$ implies $d_j \leq d_{j'}$ for any two jobs j, j'. We derive an online algorithm that only uses $\mathcal{O}(m)$ machines. In fact, our algorithm is even non-preemptive, even though we can compare to a preemptive optimum.

Theorem 2.18. *In the non-migratory setting, there is a non-preemptive $\mathcal{O}(m)$-machines algorithm for agreeable instances.*

The idea is to again treat α-loose and -tight jobs separately for fixed $\alpha \in (0,1)$. To obtain a non-migratory (but not necessarily non-preemptive) algorithm for α-loose jobs, note that we could simply use Theorem 2.15. It is, however, easy to observe that EDF (with suitable tie breaking) never migrates jobs and also never preempts them. Then the following is a direct corollary from Theorem 2.5.

Corollary 2.2. *Let $\alpha \in (0,1)$. In the migratory setting,* EDF *is a non-preemptive $(m/(1-\alpha)^2)$-machines algorithm for α-loose jobs.*

Consider α-tight jobs. We use the following simple algorithm MediumFit: Any job j runs exactly in the interval $[r_j + \ell_j/2, d_j - \ell_j/2)$, independently of all other jobs. Note that this algorithm is meaningful in the sense that running j in $[r_j + \ell_j, d_j)$ or $[r_j, d_j - \ell_j)$ does *not* yield a schedule on $\mathcal{O}(m)$ machines.

The analysis MediumFit works via a load argument.

Lemma 2.19. *Let $\alpha \in (0,1)$.* MediumFit *is an online algorithm that produces a non-preemptive solution on $16m/\alpha$ machines for any agreeable instance that consists only of α-tight jobs.*

Proof. Consider an arbitrary time t and all jobs j' that are run by MediumFit at this time, among which we let j be a job with minimum laxity. We estimate their contributions to the interval(s)

$$I = [r_j - 2\ell_j, r_j + 2\ell_j) \cup [d_j - 2\ell_j, d_j + 2\ell_j),$$

which has a total length of at most $8\ell_j$. Distinguish two cases for each j' that is run at t.

Case 1: We have $|I(j')| \geq 2\ell_j$. As $I(j')$ contains r_j or d_j, we have $|I \cap I(j')| \geq 2\ell_j$. Given that $\ell_{j'} \leq \ell_j$, j' contributes at least $2\ell_j - \ell_{j'} \geq \ell_j$ to I.

Case 2: We have $|I(j')| < 2\ell_j$. As a consequence $I(j') \subseteq I$. Observe that $|I(j')| \geq \ell_j/2$ holds because both j and j' are run at time t by MediumFit. As j' is α-tight, its contribution to I is at least $\alpha\ell_j/2$.

Let n_1 and n_2 be the number of jobs corresponding to the above two cases, respectively. Then the contribution of the $n_1 + n_2$ jobs to I is at least

$$(n_1 + \alpha n_2/2) \cdot \ell_j \geq (n_1 + n_2) \cdot \alpha\ell_j/2.$$

Using Theorem 2.1, the total contribution is upper bounded by $m|I| \leq 8m\ell_j$, implying $n_1 + n_2 \leq 16m/\alpha$. □

Corollary 2.2 and Lemma 2.19 imply a non-preemptive solution on $m/(1-\alpha)^2 + 16m/\alpha$ machines, implying Theorem 2.18. We remark that the latter expression attains its minimum at approximately $32.70 \cdot m$ for $\alpha \approx 0.63$.

2.8 Conclusion and Open Problems

Whereas we have made significant progress in this chapter, it remains one of the most important open problems in online scheduling [Pru10] to close the gap for the online machine requirement in the migratory setting: We could prove an upper bound of $\mathcal{O}(\log m)$ on the factor by which the number of machines has to be augmented online, which is the first factor that only depends on m. The best known lower bound on this factor is, however, only a constant. Towards solving this problem, a promising direction would be trying to come up with a tighter analysis of our $\mathcal{O}(m \log m)$-machines algorithm; indeed, it has not been ruled out that it is in fact an $\mathcal{O}(m)$-machines algorithm.

For the non-migratory setting, we have settled the asymptotical behavior of the online machine requirement terms of m for the general case and the considered special cases, except for the laminar one. Closing the gap for this special case, which is again between $\mathcal{O}(m \log m)$ and $\Omega(m)$, is also an interesting open problem for future research.

The focus of this chapter was not on exact guarantees but on asymptotical behavior. Especially once the gap for the migratory setting has been reduced to a constant, it will be worth studying exact bounds. Indeed currently, even though we have an $\mathcal{O}(m \log m)$-machines algorithm, this algorithm requires 261 machines when $m = 2$. Quite remarkably, however, there is no lower bound that shows that 3 machines are not sufficient. In fact, for speed augmentation, which will be the topic of the following chapter, the existence of a speed-2 algorithm is rather obvious, and we focus on reducing this constant.

Chapter 3

Online Deadline Scheduling with Speed Augmentation

Bibliographic information. The lower bound presented in Subsection 3.5.2 has been announced in an erratum [AGM15], and a simpler form of it was sketchted there. The other results presented in this section are unpublished.

This section deals with the variant of the problem from Chapter 2 in which we augment the resources that the online algorithm uses by additional speed instead of additional machines, as motivated in Section 1.1. We formally introduce the problem in Section 3.1. We name related and new results in Sections 3.2 and 3.3, respectively. Then we consider known and new algorithms in Sections 3.4 and 3.5. This leaves some open problems for Section 3.6.

3.1 Preliminaries

We start by defining the speed-augmentation problem that was introduced by Phillips et al. [PSTW02]. However, we will use a slightly more precise formulation of schedules than in both the literature and Chapter 2 as we wish to allow more elegant descriptions of the algorithms in this chapter – in particular, we would like to avoid the need for round-robin schedules so as to, for instance, run a job at unit speed on a speed-s machine. Afterwards, we introduce notions and terms, which will be used throughout the chapter. To make this chapter self contained, we will repeat a few definitions already used in Chapter 2.

Problem Definition. As in Chapter 2, we are given a number of machines m and a set of jobs $J = \{1, \dots, n\}$. Again, each job has a *release date* $r_j \in \mathbb{N}$, a *processing time* p_j, and a *deadline* d_j. Compared to Chapter 2, the following is a more general definition of migratory schedules: For $s \in \mathbb{R}_+$, a *speed-s schedule* S of J is given by a family of functions $(S_j)_{j \in J}$, where, for all $j \in J$, $S_j : [r_j, d_j) \to [0, s]$ is a Lebesgue-integrable function that defines, for all $t \in [r_j, d_j)$, at which speed S runs j at t. Note that, indeed, the schedules from Chapter 2 correspond to those S where $S_j(t) \in \{0, 1\}$ for all j and $t \in [r_j, d_j)$. Allowing fractional values for $S_j(t)$ can be interpreted as allowing

jobs to share machines, which one would need to simulate by a round-robin schedule if only 0 and 1 were allowed as values.

For each speed-s schedule S of J, we must have

$$\sum_{j \in J : t \in [r_j, d_j)} S_j(t) \leq s \cdot m,$$

for all times $t \in \mathbb{R}$, meaning that a *(machine) capacity* of only $s \cdot m$ (that is, only m machines) may be used at each time. Further, for S to be *feasible* it must hold that

$$\int_{r_j}^{d_j} S_j(t) \, \mathrm{d}t \geq p_j$$

for all jobs $j \in J$, that is, each job must receive a processing of p_j (be *finished*) until its deadline. We assume that the input instance is *feasible*, that is, that it has a feasible unit-speed schedule. We note that this definition of feasibility is identical to the one from Chapter 2.

To measure the quality of algorithms in this chapter, we will consider the speed that it uses to produce feasible schedules: We call an online algorithm a *speed-s algorithm* if it is guaranteed to produce feasible speed-s schedules. We also remark that any schedule on cm unit-speed machines can be simulated by a speed-c schedule in a trivial way; a reverse simulation is, however, obviously not possible. We also remark that all the algorithms considered in this sections produce schedules S such that all functions S_j are step functions. Future algorithms may, however, use our more general definition.

Further Notation. For some j and $t \in [r_j, d_j)$, we say that j is *available* at t. At t, the *remaining processing time* of a job j at unit speed (its *processing volume*) is (as in Chapter 2) denoted by $p_j(t)$. For a speed $\sigma \in \mathbb{R}_+$ that is potentially different from the speed that the algorithm uses, the remaining processing time of j on a speed-σ machine is denoted by $p_j^\sigma(t) \coloneqq p_j(t)/\sigma$. Accordingly, we define besides the standard *laxity* $\ell_j(t) \coloneqq d_j - r_j - p_j(t)$ the *σ-laxity* of j at t by $\ell_j^\sigma(t) \coloneqq d_j - r_j - p_j^\sigma(t)$. This value can be interpreted as the maximum duration that j can be delayed after t to guarantee that it meets its deadline when run at speed σ.

Simple Algorithms. In absence of (additional) speed, *Earliest Deadline First* (EDF) and *Least Laxity First* (LLF) have natural formulations, in which the set of scheduled jobs is only changed at integer points in time (Section 2.1). In presence of speed, however, a job may run at a fractional speed, causing it to be finished at a non-integer point in time. To avoid idleness, one then wishes to pick a currently unscheduled job (if it exists) to be scheduled on the otherwise idle machine. This is not a big issue for EDF, where processing does not change the priority order of jobs: At each integer time and at all times when a job's processing is finished, EDF picks m unfinished jobs with minimum deadline and schedules them at full speed on one machine each. For LLF, however, where processing does affect the priority order of jobs, it would seem unnatural to let the finishing times of certain jobs steer at which time points the schedule respects the priority order of jobs. Later in this chapter (Subsection 3.4.1), we will give a natural formulation of LLF that fulfills natural properties at *all* points in time.

3.2 Related Work

In addition to the other approaches to overcoming the impossibility result due to Dertouzos and Mok [DM89] as discussed in Section 2.2, speed augmentation was proposed by Phillips et al. [PSTW02], who showed that there is no speed-s algorithm for any $s < 6/5$. Lam and To [LT99] improved this lower bound later to $(1 + \sqrt{2})/2 \approx 1.207$.

Phillips et al. further showed that both EDF and LLF are speed-$(2 - 1/m)$ algorithms. We note that there are different variants of LLF (Subsection 3.4.1), but the upper-bound proofs works for all (reasonable) variants. Both the proofs for EDF and LLF rely on the same simple load argument. While a simple matching lower bound on the speed requirement was given for EDF, the largest known lower bound for LLF is the golden ratio $\varphi \approx 1.618$, which is due to Anand, Garg, and Megow [AGM11] and only works for a certain variant of LLF. The authors also showed a (matching) lower bound of $2 - 1/m$ on the speed requirement of the algorithm *Earliest Deadline until Zero Laxity*, which gives highest priority to zero-laxity jobs and prioritizes the other jobs by their deadlines.

The first improvement upon the guarantee of EDF and LLF was achieved by Lam and To's algorithm *Full-Reduced* (FR) [LT99]. They showed the speed requirement of FR to be exactly $2 - 2/(m + 1)$, which is smaller than $2 - 1/m$ for each fixed m but asymptotically identical. This algorithm tries to meet the finishing times of the jobs in an estimate of the optimum that can be constructed online and is called *Yardstick* (YS, Subsection 3.5). This estimate (on m unit-speed machines) is obtained by dropping the constraint that a job may not be parallelized on multiple machines and, at all times, considering the jobs in order of their deadlines. When a job has received less work than it could possibly have by being run on a single machine from its released date on (that is, it is *underworked*), it gets all available machines; otherwise it only gets a single machine. The algorithm FR meets the finishing times in YS and, to do so, starts each job in *full mode* and schedules the full-mode jobs via EDF. Whenever the processing a job has received is sufficiently far ahead of that in YS, it goes to *reduced mode* and, from now on, only requires processing when it is run in YS.

As an algorithm that only considers the jobs via their YS solution, FR is *deadline-ordered*, meaning that it only considers the order of deadlines instead of the actual deadline values. There is a lower bound of $e/(e - 1) \approx 1.58$ on the speed guarantee of any deadline-ordered algorithm [LT99]. In fact, Anand, Garg, and Megow [AGM11] defined the algorithm YS-*Stretch* (YSS) that is also deadline-ordered and was claimed to be a speed-$e/(e - 1)$ algorithm. To also meet the finishing times given by YS, the idea was to "round" the YS solution by distributing the parallelized processing volume of jobs over larger time intervals so as to handle them without parallelization on speed-s machines. The claim that YSS is a speed-$e/(e - 1)$ algorithm is, however, wrong [AGM15], as will be shown in Subsection 3.5.2.

Chan, Lam, and To [CLT05] considered non-migratory algorithms (see Chapter 2 for a definition) but compared their performance to a migratory offline solution. The upper and lower bounds they obtained on the speed requirement of such a non-migratory algorithm are $2 - 2/(m + 1)$ and $3 + 2\sqrt{2} \approx 5.828$, respectively.

3.3 Contribution and Outline

In this chapter (Subsection 3.5.2), we show that YSS is not a speed-$e/(e-1)$ algorithm as claimed in [AGM11]. While a simple four-job instance shows that YSS is not a speed-4/3 algorithm for $m = 2$ as claimed [AGM15], we can show a lower bound of 3/2 in this case by a more complicated instance (matching the upper bounds for EDF and LLF). Similarly, we can show a lower bound of 5/3 for $m = 3$, which then shows that YSS is indeed no speed-$e/(e-1) \approx 1.58$ algorithm. The results for $m = 2, 3$ may suggest a lower bound of $2 - 1/m$, which we, however, fail to prove and only conjecture along with an upper bound of $2 - 1/m$.

Considering the lower-bound construction for YSS, its main issue may be that it does not use the entire capacity even if an underworked job does not receive full speed. In fact, we propose another class of algorithms which try to meet the deadline imposed by YS (Subsection 3.5.3). Given some algorithm A, the algorithm YS-A (like YSS) schedules each job exactly as in YS whenever the job is not underworked. Then we use A in a certain way to distribute the remaining volume of jobs over the machines. We show that YS-EDF is not a speed-$(2-\varepsilon)$ algorithm for any $\varepsilon > 0$. While YS-LLF may require an arbitrarily large speed for some variant of LLF, we conjecture that YS-LLF for another variant of LLF is actually a speed-$e/(e-1)$ algorithm.

As it is also needed as subroutine for YS-LLF, we rigorously define LLF within the speed-augmentation model in Subsection 3.4.1. In fact, our algorithm is (in a sense) the unique algorithm that fulfills natural properties. We extend this formulation to arbitrary σ-laxities, where we denote the resulting algorithm by LLF_σ, and relate it to variants used in the literature. We then show a lower bound of at least $2 - 1/m$ for all these variants (Subsection 3.4.2), matching the upper bound (for reasonable variants) given in the literature. Quite surprisingly, this also matches the upper bound for the significantly simpler algorithm EDF. Indeed, we are able to show an upper bound in the defense of LLF (Subsection 3.4.3): We prove that, for instances with a single release date, LLF is a unit-speed algorithm, leading us the the conjecture that, for instances with any fixed number of release dates, LLF is a speed-$(2-\varepsilon)$ algorithm for some $\varepsilon > 0$. In contrast, EDF shows its worst-case behavior already on instances with a single release date.

3.4 Least Laxity First

In this section, we first give our definition of LLF, which is based on natural properties, and then we relate it to definitions used in the literature (Subsection 3.4.1). Afterwards, we give a matching lower bound on the speed requirement of LLF that also matches the speed requirement of EDF (Subsection 3.4.2), but finally we work towards positive results concerning LLF (Subsection 3.4.3).

3.4.1 A Natural Definition

We parameterize our LLF algorithms with a laxity-speed $\sigma \geq 1$ besides the actual speed s that it uses for processing jobs. The algorithm LLF_σ will prioritize jobs by their σ-laxity. We propose the following natural properties which LLF_σ should have:

Algorithm 3.1: Least σ-Laxity First (LLF_σ)

Input: instance J, a speed s, and laxity-speed σ
Output: the LLF_σ schedule of J

for all times t **do**
  $(j_1, \dots, j_k) \leftarrow$ jobs $j \in J$ with $t \in [r_j, d_j)$ and $p_j(t) > 0$, asc. ordered by $\ell_j^\sigma(t)$
  if $k \leq m$ **then**
    $S_j(t) \leftarrow s$, for all $j = j_1, \dots, j_k$
  else
    $M \leftarrow \left\{ j \in \{j_1, \dots j_k\} \mid \ell_j^\sigma(t) = \ell_{j_m}^\sigma(t) \right\}$ `//possibly partial processing`
    $F \leftarrow \{j_1, \dots, j_m\} \setminus M$ `//full processing`
    $S_j(t) \leftarrow s$, for all $j \in F$
    $S_j(t) \leftarrow s \cdot \frac{|M|}{m - |F|}$, for all $j \in M$
  end
end
return S

(i) At all times, LLF_σ uses as much machine capacity as possible (to schedule unfinished jobs).
(ii) At all times t and for all jobs j, j' with $\ell_j^\sigma(t) < \ell_{j'}^\sigma(t)$, LLF_σ only schedules j' at time t if it schedules j at speed s at t.

We describe our algorithm LLF_σ that has all of the above properties. Consider any point in time. LLF_σ considers the currently unfinished available jobs. Let $j_1, \dots, j_k$ be these jobs in increasing order of σ-laxity. If $k \leq m$, each of these jobs is scheduled at speed s. Otherwise, all jobs that have a smaller σ-laxity than j_m, say, $m' < m$ jobs, are run at speed s. All jobs that have the same laxity as j_m equally share the $m - m'$ remaining machines, meaning their laxity is kept equal. We give a formal listing as Algorithm 3.1.

It is easy to verify that LLF_σ indeed has Properties (i) and (ii). We note that this definition is unique, up to, for instance, scheduling slightly differently for a set of time points of measure zero. We will show that every algorithm having Properties (i) and (ii) processes each job in each interval exactly as much as LLF_σ. Formally, we show the following result.

Theorem 3.1. *Consider some instance J, and let S be the schedule of J computed by LLF_σ. Furthermore, let A be another algorithm that has Properties* (i) *and* (ii)*, and let S' be the schedule of J it computes. Then, for every interval $[a, b]$ and job j, it must hold that*

$$\int_a^b S_j(t)\,\mathrm{d}t = \int_a^b S'_j(t)\,\mathrm{d}t. \tag{3.1}$$

The non-trivial part of the proof is to show the statement for a job j that is run neither at speed s nor 0 by LLF_σ. The proof idea is to assume that for an interval $[a, b]$ Equation (3.1) does not hold, and then to find an interval $[a, b'] \subsetneq [a, b]$ such that, at time b', a job with a larger laxity than j is run at a higher speed than j by A. We now give the formal proof.

Proof. Suppose the contrary of the theorem statement, and let a be the infimum of all time points such that there is a job j and an a right endpoint b that do not satisfy Equation (3.1). We can assume ($\star$) w.l.o.g. that, during $(a, b]$, no job reaches its deadline, is released, or finished by LLF_σ or A.

Clearly, if there are at most m unfinished jobs available, Property (i) implies that A needs to run all of them at speed s. Hence, there are more than m unfinished jobs available at a in A. By our choice of a and assumption ($\star$), these are exactly the available unfinished jobs in both A and LLF_σ throughout $[a, b]$. Let $j_1, \ldots, j_k$ with $k > m$ be these jobs in ascending order of their σ-laxities at a in A (and thus in LLF_σ by our choice of a). Further, let $j_q, \ldots, j_{q'}$ with $q \leq m \leq q'$ be all the jobs among them that have the same σ-laxity as j_m.

Due to Property (i), A needs to use the entire machine capacity at a. To be able to use the capacity for jobs $j_m, \ldots, j_k$, however, Property (ii) forces A to give speed s to all $q-1 < m$ jobs $j_1, \ldots, j_{q-1}$. So, since Equation (3.1) odes not hold, A must schedule $j_q, \ldots, j_{q'}$ differently than LLF_σ. That is, there must exist an $\varepsilon > 0$ such that $a + \varepsilon \leq b$ and

$$\Delta_{j_i,j_{i'}}(a+\varepsilon) \coloneqq \int_a^{a+\varepsilon} S'_{j_i}(t)\,\mathrm{d}t - \int_a^{a+\varepsilon} S'_{j_{i'}}(t)\,\mathrm{d}t > 0$$

for some jobs $j_i, j_{i'}$ with $i, i' \in \{q, \ldots, q'\}$. By continuity of $\Delta_{j_i,j_{i'}}$, there must also be some $\varepsilon' < \varepsilon$ with

$$\Delta_{j_i,j_{i'}}(a+\varepsilon') \in (0, \Delta_{j_i,j_{i'}}(a+\varepsilon)), \tag{3.2}$$

and, using the definition of $\Delta_{j_i,j_{i'}}(a+\varepsilon')$ as well as linearity of the integral operator,

$$S'_{j_i}(a+\varepsilon') - S'_{j_{i'}}(a+\varepsilon') > 0. \tag{3.3}$$

Now note that Relation (3.2) implies $\ell_{j_i}(a+\varepsilon') > \ell_{j_{i'}}(a+\varepsilon')$ in A; hence A, by having Property (ii), A must schedule $j_{i'}$ at speed s if it schedules j_i at all, contradicting Inequality (3.3). □

The variants of LLF considered in the two related papers [PSTW02, AGM11] "at any time" pick m minimum-laxity jobs (w.r.t. 1- and s-laxity, respectively) and schedule them on one machine each. As Theorem 3.1 shows, however, such an algorithm does not exist in our continuous-time model while it obviously does in a discrete-time model. Without making this formal, we note that, indeed as the discretization step goes to zero, such an algorithm in the limit assigns as much work to every job as LLF_σ during every fixed interval.

We also note that, even though 1-laxity was considered in [PSTW02], their proof that shows an upper bound of $2 - 1/m$ on the speed requirement of LLF also works for LLF_σ whenever $\sigma \geq 1$. The lower-bound proof in [AGM11], however, is tailored to s-laxity, and it only shows a lower bound of $\varphi \approx 1.618$ on the speed requirement of LLF_σ for $\sigma \geq s$; for $\sigma < s$, it is easy to see that the same construction yields a non-trivial but smaller lower bound.

Another interesting observation is that, for any fixed instance in which no two deadlines are the same, the schedule computed by LLF_σ for sufficiently large σ is identical to that computed by EDF. So using the lower-bound result for EDF [PSTW02], it is not surprising that, for sufficiently large σ, there is a lower bound of $2 - 1/m$ on the speed requirement of LLF_σ. We are, however, able to show this result even for $\sigma = 1$ in the following subsection.

3.4.2 A Tight Lower Bound

We give a lower bound on the speed required by LLF_σ, matching the upper bound of $2 - 1/m$ given in the seminal paper [PSTW02]. Note that the construction has a structure similar to the construction which we used to show a lower bound on the machine requirement of LLF for tight jobs (Theorem 2.7). As we are aiming for a matching lower bound here, however, the present proof requires more exact calculations.

Theorem 3.2. *Let $s < 2 - \frac{1}{m}$. Then LLF_σ is not a speed-s algorithm for any $\sigma \geq 1$.*

We first give the following crucial lemma. W.l.o.g. we use arbitrary rational numbers as job parameters.

Lemma 3.1. *Let $2-1/m > s \geq 1$, $\sigma \geq 1$, and $\varepsilon \in (0,1)$. Then there exists an instance J_ε and a time $t^\star$ with the following properties:*

(i) There exists a feasible schedule $S^\star$ of J_ε on unit-speed machines with $p_{j'}(t^\star) = 0$, for all $j' \in J_\varepsilon$.

(ii) At time $t^\star$ in LLF_σ given speed s, there is exactly one unfinished job $j \in J_\varepsilon$ with $r_j = 0$, $d_j > t^\star$, and

$$\frac{\ell_j(t^\star)}{d_j - t^\star} = \varepsilon.$$

Further, j is run by LLF_σ only if it is currently the only unfinished job.

Proof. In this proof, we show the result for LLF_1, but it can be easily seen that the LLF_1 and the LLF_σ schedules of our instance J_ε are identical for all $\sigma > 1$. In the following, we simply use LLF as a symbol for LLF_1. The idea is that j runs uninterruptedly on one machine from its release date 0 through $t^\star = p_j$ in the feasible schedule $S^\star$. In LLF, on the other hand, we ensure that j is indeed only run if it is the only available unfinished job. To this end, the only jobs other than j that we release are jobs of the following type. At any time $t < t^\star$ when j is the only unfinished job in $S^\star$, we release m identical jobs j' with $d_{j'} = \min\{t + \ell_j(t), t^\star\}$, which we also call a *wave* of jobs. Since $d'_j = t + \ell_j(t)$ implies $\ell_{j'}(t') < \ell_j(t')$ for all $t' \in [r_{j'}, d_{j'})$, the LLF schedule has the desired property.

It remains to set the processing times of the jobs appropriately. For any $j' \neq j$, we set $p_{j'} = (d_{j'} - r_{j'}) \cdot (m-1)/m$. Observe that this relation of $r_{j'}$, $p_{j'}$, and $d_{j'}$ makes sure that all the j' from a certain wave can be finished in $S^\star$ at time $d_{j'}$ by letting them equally share the $m-1$ available machines. For any $t' \in [0, p_j]$ at which a wave gets finished in $S^\star$, we can now compute the laxity of job j in LLF at time t':

$$\begin{aligned}\ell_j(t') = d_j - p_j(t') - t' &= d_j - \left(p_j - s \cdot \left(t' - \frac{m-1}{ms} \cdot t'\right)\right) - t' \\ &= d_j - p_j - \left(2 - \frac{1}{m} - s\right) \cdot t'. \end{aligned} \tag{3.4}$$

As we have $s < 2 - 1/m$, the laxity of j is thus decreased by every wave. For the lemma to hold, it suffices to make sure that $\ell_j(p_j)/(d_j - p_j) = \varepsilon$ or, equivalently, using Equation 3.4, that

$$d_j - p_j - \left(2 - \frac{1}{m} - s\right) \cdot p_j = \varepsilon \cdot (d_j - p_j).$$

Solving for p_j yields

$$p_j = \frac{1-\varepsilon}{1-\varepsilon+\left(2-\frac{1}{m}-s\right)} \cdot d_j,$$

which is in the interval $(0, d_j)$ by $s < 2 - 1/m$. □

This lemma can be used directly to get a lower bound of $1 + 1/m$ by additionally releasing m sufficiently long zero-laxity jobs at the time when all jobs of J_ε could be finished in a feasible solution. To increase this bound to $2 - 1/m$, we have to apply the lemma several times to obtain a nested instance with $m - 2$ copies of the in LLF_σ unfinished job.

Proof of Theorem 3.2. We claim that for every $\varepsilon > (0, 1)$ and natural number $k \leq m$ there is an instance J_ε^k and a time $t^\star$ with the following properties:

(i′) There exists a feasible schedule $S^\star$ of J_ε^k on unit-speed machines with $p_{j'}(t^\star) = 0$, for all $j' \in J_\varepsilon^k$.

(ii′) At time $t^\star$ in LLF_σ given speed s, there are exactly k unfinished jobs $j \in J_\varepsilon^k$ with identical deadline $d_j > t^\star$ such that

$$\frac{\ell_j(t^\star)}{d_j - t^\star} = \varepsilon.$$

Note that $J_\varepsilon^1 = J_\varepsilon$. In the following we will refer to $t^\star$ from this statement as the *critical moment* of J_ε^k and to the corresponding jobs as *critical jobs*. We prove this claim inductively using Lemma 3.1: Assume it holds for $k \geq 1$. Consider J_ε and let j be the critical job. We define $\varepsilon' := p_j/d_j$.

Now J_ε^{k+1} is built as follows. We release $J_{\varepsilon'}^k$ at time 0, and let d be the deadline of its critical jobs. At the critical moment of $J_{\varepsilon'}^k$, we release a scaled-down version of J_ε such that d is the deadline of its critical job j. First observe that (i') follows directly from Part (i') of the induction hypothesis and Lemma 3.1 (i). To see (ii'), note that, due to Part (ii') of the induction hypothesis, there are $k < m$ copies (w.r.t. σ-laxity and deadline) of job j at that time in LLF_σ. Since LLF_σ on J_ε runs j only when j is the only job that is unfinished (Lemma 3.1 (ii)), all of these less than m jobs are treated by LLF_σ in the same way as j. Again using Lemma 3.1 (ii), this shows the claim with the critical moment of J_ε^{k+1} being the critical moment of the scaled-down version of J_ε.

Now let $k > (s-1) \cdot m/(1-\varepsilon)$. We add m additional zero-laxity jobs J having the critical moment of J_ε^k as release date and sharing the deadline with all critical jobs. By (i'), $J_\varepsilon^k \cup J$ is still feasible on unit-speed machines. On the other hand, LLF_σ fails on it as it has to finish a total work of

$$k \cdot (1-\varepsilon) + m > (s-1) \cdot m + m > s \cdot m$$

between the release date of the jobs in J and their deadlines, which exceeds the total available capacity $s \cdot m$.

We illustrate the construction in Figure 3.1. □

Without a proof, we note that setting $\sigma < 1$ requires LLF_σ to use arbitrarily high speed since this may make LLF_σ work on jobs that have a very late deadline, even though there may be jobs that have an early deadline and are very critical.

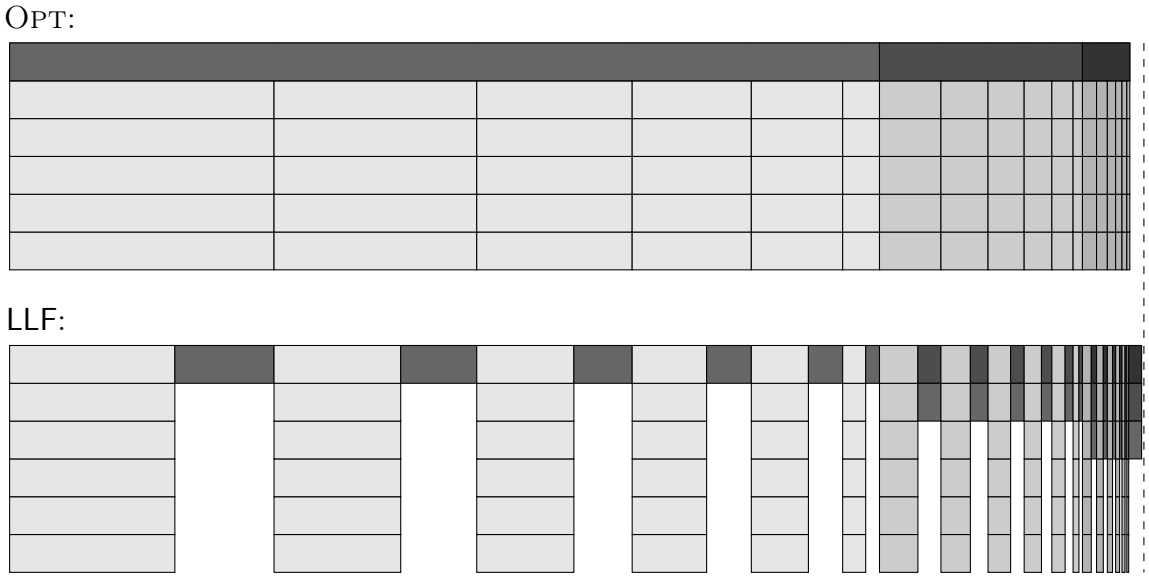

Figure 3.1: An illustration of our lower-bound construction for LLF. In particular, $m = 6$, and the shown schedules are the optimal schedule for $J^3_{0.15}$ and the LLF schedule using speed 1.6. Different shades of gray indicate different types of jobs and applications of Lemma 3.1, and the dashed line indicates the deadline of the three critical jobs.

3.4.3 Towards a Parameterized Upper Bound

Despite showing the same lower bound on the speed requirement, the lower-bound construction for LLF shown in the previous section is by far more complicated than the lower-bound construction for EDF [PSTW02]. In fact, while the lower bound for EDF only requires a single released date, we can show the following strong upper bound for LLF.

Theorem 3.3. *On instances with a single release date,* LLF_1 *is a unit-speed algorithm.*

Before we show this theorem, we note the following fact.

Proposition 3.1. *Let* j *and* j' *be two jobs that are available in* LLF *at the two points in time* $t < t'$*. Then* $\ell_j(t) \leq \ell_{j'}(t)$ *implies* $\ell_j(t') \leq \ell_{j'}(t')$*.*

The theorem can now be shown by a simple load argument.

Proof of Theorem 3.3. Suppose LLF_1 fails on an instance J with one release date, which we assume w.l.o.g. to be 0. Then there exists a time such that there is a negative-laxity job. Let $t^\star$ be the infimum of all these times, and let $j_1, \ldots, j_k$ be all the negative-laxity jobs at time $t^\star$ in LLF_1. We claim $k > m$: If this was not true, each zero-laxity job would be run at speed s at time $t^\star$, and no job's laxity would become negative. So $j_1, \ldots, j_k$ are the only jobs run by LLF_1 at time $t^\star$. We now claim that each job $j \in J$ must have received at least as much work in any other feasible schedule as in LLF_1 until time $t^\star$. To do so, distinguish three cases:

Case 1: The job j was not run by LLF_1 before $t^\star$. Then our claim clearly holds.

Case 2: We have $d_j \leq t^\star$. Then our claim clearly holds as well.

Case 3: The job j was run by LLF_1 before $t^\star$ and $d_j > t^\star$. Since $k > m$ and there is only one release date, there must be an $i \in \{1, \ldots, k\}$ such that $\ell_j(t) \leq \ell_{j_i}(t)$ at some $t < t^\star$. Since $d_j > t^\star$, Proposition 3.1 can be applies and implies that $\ell_j(t^\star) \leq \ell_{j_i}(t^\star)$, that is, $j = j_{i'}$ for some $i' \in \{1, \ldots, k\}$

Since there are more than m jobs available at time $t^\star$ and 0 is the only release date, LLF has used a machine capacity of $m \cdot t^\star$ until time $t^\star$. By the above reasoning, any feasible schedule also has to use this much machine capacity until $t^\star$, plus, to net get negative-laxity jobs, an additional non-negative capacity on job $j_1, \ldots, j_k$, exceeding the machine capacity it has until time $t^\star$; a contradiction. □

It seems plausible that an approximate version of this load argument also works for a larger number of release dates, leading us to the following conjecture.

Conjecture 3.1. *For every k there is an $\varepsilon > 0$ such that LLF_1 is a $(2-\varepsilon)$-speed algorithm for instances with at most k distinct release dates.*

3.5 Yardstick-Based Algorithms

In this section, we first (Subsection 3.5.1) define the Yardstick (YS) estimate as introduced in [LT99]. Then we address two approaches to "rounding" YS to obtain an algorithm with a speed requirement of smaller than $2 - 2/(m+1)$ (that is, FR's guarantee): While the first approach (Subsection 3.5.2) is shown to fail, we conjecture that a certain variant of the second approach (Subsection 3.5.3) yields a best-possible deadline-ordered algorithm.

3.5.1 Deadline-Ordered Algorithms and Yardstick

The YS estimate of the optimum becomes natural in the following setting: We would like to have an estimate that is *deadline-ordered*, meaning that it only depends on the relative order of deadlines instead of their actual values. To be able to meet all deadlines on m unit-speed machines, we do not actually construct a schedule, where each job is associated with a function that maps to $[0, 1]$, but we allow functions that map to $[0, m]$. In other words, we relax the constraint that each job may only be run on one machine at a time.

Indeed, the following estimate is guaranteed to meet all deadlines: At all times, we run the unfinished job with the minimum deadline on all available machines. Feasibility can be seen by a simple load argument. To get a more meaningful estimate, however, we will only run a job in parallel on multiple machines if we possibly need to catch up on an optimal schedule: Towards this, we call a job j *underworked* at t if it has received less work than it possibly could have (ignoring all other jobs), that is, if $p_j(t) > p_j - (t - r_j)$.

Now, to obtain the YS estimate, we do the following at all times t: We consider the unfinished jobs in order of their deadlines. If a job j is not underworked at t, j is scheduled on a single machine; if j is underworked, j is scheduled on all the remaining machines (not taken by smaller-deadline jobs). We stop going through the jobs whenever

all machines are filled up, that is, after going through m non-underworked jobs or after one underworked job was assigned. In [LT99], it is shown also via a simple load argument that the YS estimate meets all deadlines.

For some time t, we will also consider the estimate YS^t that is identical to YS for times in $[t, \infty)$ but only exists from time t on. For a job j, we will use f_j^t to denote its finishing time in YS^t (more formally, the supremum of all times when j is processed in YS^t). We will use x_j^t to denote the maximum of f_j^t and the supremum of all times when j is underworked in YS^t.

Examples of YS estimates will be given in the following subsections in the context of algorithms that build up on YS.

3.5.2 First Approach to Rounding Yardstick: YSS

We first define the algorithm YSS from [AGM11] and then show the new lower bound on its speed requirement.

Definition

Suppose we are given speed s. At every release date t, we will recompute the YSS schedule starting at time t, which we call YSS^t in the following. To do so, we compute the YS^t estimate of the instance consisting of all the jobs released up to t. Note that the processing times of a job j left at t may be different in YS and YSS; by $p_j(t)$ we will refer to the processing time left in YSS.

A preliminary schedule YSS^t is computed the following way: For each job j and at each time when j is not underworked and processed in YS^t, that is, during $[x_j^t, f_j^t]$, we process the job at the same speed as it is scheduled in YS^t (that is, at unit speed). The remaining workload $p_j(t) - (f_j^t - x_j^t)$ is assigned at speed s to the interval $[s_j^t, x_j^t]$, where we define

$$s_j^t := x_j^t - \frac{p_j(t) - (f_j^t - x_j^t)}{s},$$

that is, the workload is "stretched". It can be seen quite easily seen [AGM11] that this procedure never assigns work to times earlier than t.

To obtain the final schedule YSS^t, we need to turn the preliminary schedule into one that has a staircase profile after time t, meaning that the machine capacity it uses never increases over time. To achieve this, let $j_1, \ldots, j_k$ be the jobs with release date before or at t, and let YSS_i^t for $i = 1, \ldots, k$ be the current schedule YSS^t of jobs $j_1, \ldots, j_i$. We will ensure that YSS_i^t has a staircase profile after t for $i = 1$ up to k. Towards this, let a_i^ℓ be the point where there is the ℓ-th step in YSS_i^t after time $t = a_0^i$, that is, a time when the machine capacity YSS_i^t uses changes (if it exists).

So consider some i such that $\mathsf{YSS}_0^t, \ldots, \mathsf{YSS}_{i-1}^t$ all have a staircase profile (after t) but YSS_i^t has not. If we have $s_{j_i}^t \in (a_{i-1}^q, a_{i-1}^{q+1})$ for some q, we adjust the processing of j_i by distributing its processing in $[s_{j_i}^t, a_{i-1}^{q+1})$ uniformly over $[a_{i-1}^q, a_{i-1}^{q+1})$. Additionally, whenever there is a q such that more volume in YSS_i^t is assigned to $[a_{i-1}^q, a_{i-1}^{q+1})$ than to $[a_{i-1}^{q-1}, a_{i-1}^q)$, we move a suitable amount of j_i's processing from $[a_{i-1}^{q+1}, a_{i-1}^{q+2})$ to $[a_{i-1}^q, a_{i-1}^{q+1})$ to equalize the volumes. Note that this procedure terminates, that it produces a unique solution, and that it never increases the speed that j_i is run at to more than s.

We note that, to prove that YSS is an speed-s algorithm for some s, we have to prove that it never assigns a workload of more than $s \cdot m$ to any point in time – YSS intrinsically meets all deadlines. Quite differently, algorithms like EDF and LLF intrinsically never assign more work than $s \cdot m$ to any point in time, but they may not meet all deadlines.

Lower Bounds

We now refute the claim made in [AGM11] that the speed requirement of YS is

$$\frac{1}{1-(1-\frac{1}{m})^m},$$

which is $e/(e-1) \approx 1.58$ in the limit for $m \to \infty$.

Theorem 3.4. *For $m = 2, 3$ and every $\varepsilon > 0$, YSS is not a speed-$(2 - \frac{1}{m} - \varepsilon)$ algorithm.*

We first describe the lower-bound instance and then give a separate proof for $m = 2$ and $m = 3$. We will later choose k appropriately and define the instance $J_{m,k}$ the following way. It consists of *stretched* jobs $j_1, \ldots, j_{m-1}$ as well as *unstretched* jobs, where the latter ones are further divided into k *iterations*. It holds that $d_{j_1} < \cdots < d_{j_{m-1}}$ and, furthermore, every unstretched job has a deadline strictly smaller than d_{j_1}. As there will only be m *unfinished* unstretched jobs available at any time in the YS estimate of $J_{m,k}$, the order of deadlines of unstretched jobs does not play a role.

All the strechted jobs are released at time 0 and we set

$$p_{j_i} = k \cdot \left(\frac{m}{m-1}\right)^i.$$

The unstretched jobs from the i-th iteration are released at time $i - 1$ (that is, the stretched jobs and the jobs from first iteration are released at the same time). Here, the first iteration consists of m jobs, $m-1$ of which have processing time 1 and one of which has processing time 2. In each of the following iterations except for the k-th one, the released jobs are identical to those released in the first iteration, except for one of the smaller jobs which is missing. In contrast, the k-th and final iteration lacks the longer job. We name the jobs from the i-th iteration $j_{i,1}, j_{i,2}, \ldots$ in increasing order of processing times. All the jobs are listed in Table 3.1.

First note that the instance is designed in such a way that, after the release of each iteration except for the last one, the m jobs with the smallest deadlines consist of $m - 1$ jobs with remaining processing time 1 and another one with remaining processing time 2. None of these jobs is underworked. After the release of the last iteration, m jobs with remaining processing time 1 each have the smallest deadlines and they are all not underworked. Consequently, the stretched jobs are never worked on in YS until time k. In YS^k, by the choice of their processing times, they stop being underworked exactly at their finishing times. We illustrate the yardstick schedule in Figure 3.2.

Also note that we do not have to argue that our instance is feasible – since YSS is a deadline-ordered algorithm, we only have to specify the order of deadlines and not the exact values. Clearly, for each order of deadlines on any instance, there are (very large) values of these deadlines such that the instance is feasible.

We now analyze the behavior of YSS on $J_{m,k}$ and first consider the case $m = 2$.

Iteration	j	r_j	p_j
1	$j_{1,1}$	0	1
	$\vdots$	$\vdots$	$\vdots$
	$j_{1,m-1}$	0	1
	$j_{1,m}$	0	2
$i \in \{2, \dots, k-1\}$	$j_{i,1}$	$i-1$	1
	$\vdots$	$\vdots$	$\vdots$
	$j_{i,m-2}$	$i-1$	1
	$j_{i,m-1}$	$i-1$	2
k	$j_{k,1}$	$k-1$	1
	$\vdots$	$\vdots$	$\vdots$
	$j_{k,m-1}$	$k-1$	1
stretched	j_1	0	$k \cdot \frac{m}{m-1}$
	$\vdots$	$\vdots$	$\vdots$
	j_{m-1}	0	$k \cdot (\frac{m}{m-1})^{m-1}$

Table 3.1: An overview of the instance $J_{m,k}$. The jobs are listed in increasing order of deadlines.

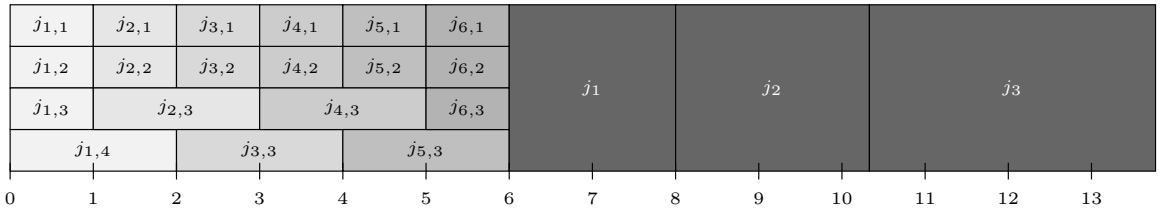

Figure 3.2: A visualization of the YS estimate for instance $J_{4,6}$. Different shades of gray indicate different iterations.

Proof of Theorem 3.4 for $m = 2$. Suppose YSS is using speed $s < 3/2$ on $J_{2,k}$ where k will be set later. There is only one stretched job, which we simply call j. By α_i for $1 \le i \le k-1$ we will denote the speed at which YSS^{i-1} (that is, the schedule that is computed after the release of the i-th iteration) runs j within $[i, i+1)$. We claim that there is an i such that $\alpha_i = s$. We will later see that we can set $k = i + 1$ then.

First observe that we have

$$s_j^0 = x_j^0 - \frac{p_j(i-1) - (f_j^0 - x_j^0)}{s} = 3 - \frac{3}{s} < 1, \tag{3.5}$$

where we use $s < 3/2$. Thus, the volume $s \cdot (s_j^0 - 1)$ gets evenly distributed over the interval $[0, 1)$. If the volume scheduled in $[0, 1)$ already exceeds that in $[1, 2)$ (or is equal to it), we have $\alpha_i = s$. So assume some volume of j gets shifted from $[1, 2)$ to $[0, 1)$ to make the aggregate volume scheduled in these intervals equal. Clearly, still $\alpha_i > 1$.

So assume that for $2 \le i \le k-2$, we have $\alpha_{i-1} < s$. It will turn out inductively that $\alpha_{i-1} > 1$. Now consider YSS^{i-1}, and note that $f_j^{i-1} = f_j^{i-2}$ and $x_j^{i-1} = x_j^{i-2} + 2$.

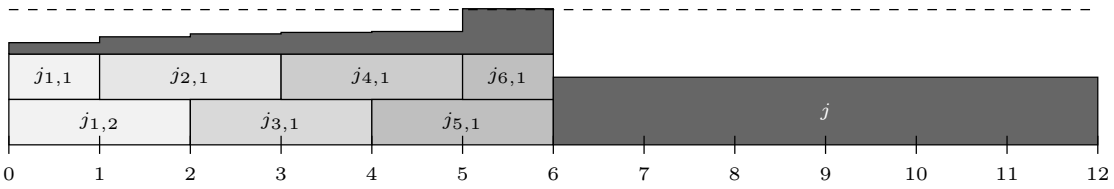

Figure 3.3: An illustration of the schedule of $J_{2,6}$ computed by YSS for $s = 1.49$. The dashed line indicates the total capacity of $2 \cdot 1.49 = 2.98$, which is exceeded (by 0.02) throughout $[5, 6)$.

This means that the volume that YSS^{i-1} has to schedule of j within $[i-1, x_j^{i-1}) = [i-1, 2i+1)$ is the volume that YSS^{i-2} schedules of j within $[i-2, x_j^{i-2}) = [i-2, 2i-1)$ plus 2. Since YSS^{i-1} can schedule as much volume of j in $[i+1, 2i-1)$ as YSS^{i-2} and a volume of $2s$ of j in $[2i-1, 2i+1)$, we get that YSS^{i-1} has to schedule a volume of $\alpha_{i-1} + s + 2 - 2s$ within $[i-1, i+1)$. By $\alpha_{i-1} > 1$ and $s < 3/2$, this volume is larger than s, implying $s_j^{i-1} < 1$. So the volume $s \cdot (s_j^{i-1} - 1)$ gets evenly distributed over $[i-1, i)$. If the volume scheduled in $[i-1, i)$ now exceeds (or is equal to) that scheduled in $[i, i+1)$, we have $\alpha_i = s$. Otherwise the volume

$$\alpha_{i-1} + s + 2 - 2s$$

gets distributed over $[i-1, i+1)$ so as to equalize the aggregate volume scheduled in $[i-1, i)$ and $[i, i+1)$, meaning

$$\alpha_i = 1 + \frac{\alpha_{i-1} + s + 2 - 2s - 1}{2} = \frac{\alpha_{i-1} + 3 - s}{2},$$

approaching $3 - s > s$, so we must have $\alpha_i = s$ for some i. We then set $k = i + 1$.

Now consider YSS^{k-1}, and note that $f_j^{k-1} = x_j^{k-1} = x_j^{k-2} + 1 = 2k$, meaning that the volume of j that YSS^{k-1} has to schedule within $[k-1, 2k)$ is the volume that YSS^{k-2} schedules of j within $[k-1, 2k-1)$ plus 1. Notice that both YSS^{k-1} and YSS^{k-2} run j at speed s in $[k, 2k-1)$, and YSS^{k-1} also runs j at speed s in $[2k-1, 2k)$, meaning that YSS^{k-1} schedules j at speed $\alpha_{k-1} + 1 - s = 1$ in $[k-1, k)$. Including the volume of jobs $j_{k-1,1}$ and $j_{k,1}$, the total volume that YSS^{k-1} now schedules within $[k-1, k)$ is $3 > 2s$.

We give an illustration in Figure 3.3. □

The proof for $m = 3$ is slightly more complicated as there are two stretched jobs j_1 and j_2.

Proof of Theorem 3.4 for $m = 3$. Consider YSS using speed $s < 5/3$ on $J_{3,k}$, where k will be set later. First note that, in YSS^i for $1 < i < k$, j_1 is scheduled at speed $(3-s)/2$ throughout the interval $[i, i+1)$ and at speed s throughout the interval $[i+1, i+2)$. So the analysis will be focussed on how j_2 is scheduled. Similar to the case $m = 2$, for $k/2 < i \leq k-2$ define α_i to be the speed at which YSS^{i-1} runs j_2 in $[i, i+1)$. We claim that there is an i such that $\alpha_i = s$. Then we will set $k = i + 1$.

Note that the argument for $m = 3$ is still more difficult than the one for $m = 2$: Whether or not $x_{j_2}^i < f_{j_1}^i$ holds influences how the release of another iteration changes

the processing of j_2. Note that, however, we have $f^i_{j_1} = 3k/2$, $x^i_{j_1} = 3i/2 + 2$, and $x^i_{j_2} = 2x^i_{j_1} = 3i + 4$ for all i with $x^i_{j_2} \leq f^i_{j_1}$, meaning that we have $x^i_{j_2} \geq f^i_{j_1}$ for $i \geq k/2$.

Also, YSS^i may not schedule j_2 within $[i, i+1)$ at all. We argue that this can only happen for a constant number of iterations. Suppose that YSS^{i+1} for $1 \leq i \leq k-2$ does not schedule j_2 in $[i+1, i+2)$. Then, since $x^{i+1}_{j_2} \geq x^i_{j_2} + 3/2$, YSS^{i+1} has to schedule a volume of j_2 of at least $3/2 \cdot (2-s) > 0$ more within $[i+2, i+3)$ than YSS^i within $[i+1, i+2)$. Since j_2 can only run at a speed of s, $\mathsf{YSS}^{i^\star}$ has to schedule j_2 in $[i^\star, i^\star+1)$ for some $i^\star$ large enough. It is also easy to see that every YSS^i for $i^\star < i \leq k-1$ schedules j_2 in $[i, i+1)$. So suppose k is chosen large enough such that YSS^i schedules within j_2 within $[i, i+1)$ for all $k/2 \leq i \leq k-2$.

Now assume $\alpha_{i-1} < s$ for $k/2 < i \leq k-1$. First note that we assume that the aggregate volumes scheduled by YSS^{i-2} in $[i-2, i-1)$ and $[i-1, i)$ are identical since otherwise $\alpha_{i-1} = s$. Hence, $3 + (3-s)/2 < 1 + s + \alpha_{i-1}$, which implies

$$\alpha_{i-1} > \frac{7}{2} - \frac{3s}{2}. \tag{3.6}$$

Further, note that $f^{i-1}_{j_2} = f^{i-2}_{j_2}$ and $x^{i-1}_{j_2} = x^{i-2}_{j_2} + 3/2$. Using $x^{i-1}_{j_2} \geq f^{i-1}_{j_1}$, this means that the volume that YSS^{i-1} has to schedule of j_2 within $[i-1, x^{i-1}_{j_2})$ is the volume that YSS^{i-2} schedules of j_2 within $[i-2, x^{i-2}_{j_2})$ plus $3/2$. Since YSS^{i-1} can schedule as much volume of j_2 in $[i+1, x^{i-2}_{j_2})$ as YSS^{i-2} and a volume of $3s/2$ of j in $[x^{i-2}_{j_2}, x^{i-1}_{j_2})$, we get that YSS^{i-1} has to schedule a volume of $\alpha_{i-1} + s + 3/2 - 3s/2$ within $[i-1, i+1)$.

By Inequality (3.6) and $s < 5/3$, this volume is larger than s. So the excess of volume w.r.t. s gets evenly distributed over $[i-1, i)$. If the volume scheduled in $[i-1, i)$ now exceeds (or is equal to) that scheduled in $[i, i+1)$, we have $\alpha_i = s$. Otherwise the volume

$$\alpha_{i-1} + s + \frac{3}{2} - \frac{3s}{2}$$

gets distributed over $[i-1, i+1)$ so as to equalize the aggregate volume scheduled in $[i-1, i)$ and $[i, i+1)$, meaning

$$\alpha_i = \frac{7}{2} - \frac{3s}{2} + \frac{\alpha_{i-1} + s + \frac{3}{2} - \frac{3s}{2} - \frac{7}{2} + \frac{3s}{2}}{2} = \frac{\alpha_{i-1} + 5 - 2s}{2},$$

approaching $5 - 2s > s$, so we must have $\alpha_i = s$ for some i. We then set $k = i + 1$.

Now consider YSS^{k-1}, and note that $f^{k-1}_j = x^{k-1}_j = x^{k-2}_j + 1$ for both $j \in \{j_1, j_2\}$, meaning that the volume of j that YSS^{k-1} has to schedule within $[k-1, x^{k-1}_j)$ is the volume that YSS^{k-2} schedules of j within $[k-1, x^{k-1}_j - 1)$ plus 1. Notice that both YSS^{k-1} and YSS^{k-2} run j at speed s in $[k, x^{k-1}_j - 1)$, and YSS^{k-1} also runs j at speed s in $[x^{k-1}_j - 1, x^{k-1}_j)$, meaning that YSS^{k-1} schedules j at speed $\alpha_{k-1} + 1 - s = 1$ in $[k-1, k)$. Hence, each of the jobs j_1, j_2, $j_{k-1,2}$, $j_{k,1}$, and $j_{k,2}$ contributes a volume of 1 to the interval $[k-1, k)$ in YSS^{k-1}, implying that the total volume scheduled in this interval is $5 > 3s$. □

While we are not able to show upper bounds for any $m \geq 2$ or lower bounds for $m \geq 4$, experiments have lead us to the following conjecture.

Conjecture 3.2. YSS *requires a speed of exactly* $2 - \frac{1}{m}$ *for every* m.

Further note that this conjecture implies that YSS improves upon neither EDF nor LLF on any number of machines [PSTW02], meaning that, for the considered problem, the best known algorithm to date is FR [LT99].

3.5.3 Second Approach to Rounding YS: YS-A

As can be seen from Figure 3.3, YSS leaves machine capacity unused even at times when an underworked job does not receive speed s. Hence, it is a natural question if forcing YSS to use this unused capacity in a natural way does yield a speed-$e/(e-1)$ algorithm. To do so, we take the following approach. At each release date t, our algorithm YS-A also computes the YS estimate YS^t. Like YSS, the schedule $\mathsf{YS\text{-}A}^t$, which YS-A uses from t on if no more jobs are released, schedules a job j exactly as in YS^t whenever it is not underworked. The remaining processing volume of j in YS-A at t is transferred to a new job j' whose deadline is x_j^t. We then apply a (preferably busy) algorithm A to schedule the set of new jobs on the machine capacity that is not used by the non-underworked parts of jobs. In the following we discuss using EDF, LLF_1, and LLF_s, whose definitions we need to slightly extend so as to handle machine capacity that varies over time (at finitely many points).

YS-EDF

The definition of EDF for varying machine capacity is straightforward: At time 0, at all times when a job gets released or finished, and whenever the machine capacity changes, we recompute: If c is the current machine capacity and there are at most $\lceil c/s \rceil$ unfinished jobs available, each of them is scheduled at speed s. Otherwise, let $j_1, \dots j_{\lceil c/s \rceil}$ be the $\lceil c/s \rceil$ unfinished jobs with minimum deadline. Then $j_1, \dots j_{\lfloor c/s \rfloor}$ are run at speed s each. If the remaining capacity $c/s - \lfloor c/s \rfloor$ is strictly positive, $j_{\lceil c/s \rceil}$ is run at this speed.

Unfortunately, in the combination with YS, EDF (at least asymptotically) does not require less speed than when used separately.

Theorem 3.5. YS-EDF *is not a speed-*$(2-\varepsilon)$ *algorithm for any* ε.

Proof. We describe an instance for arbitrary m and will later set m so that YS-EDF using speed $2-\varepsilon$ fails on this instance. All jobs are released at time 0. The m jobs with earliest deadline have processing time 1. The jobs with the next larger deadlines have a total processing time $m \cdot (1-\varepsilon)$, and it is easy to see that we can choose their number and processing times such that they are all finished until time $2-\varepsilon$ in YS, and YS-EDF uses the entire machine capacity in $[0,1)$ for all jobs described so far. Finally, the job $j^\star$ with the largest deadline has a sufficiently large processing time such that it becomes non-underworked at x_j^0.

Note that, for $m \to \infty$, we have $x_j^0 \to 2-\varepsilon$. Since YS-EDF uses the entire machine capacity in $[0,1)$ for jobs other than $j^\star$, $j^\star$ is only started at time 1 in YS-EDF. As YS-EDF uses speed $2-\varepsilon$, $j^\star$ is can in the limit only be finished at

$$1 + \frac{2-\varepsilon}{2-\varepsilon} = 2 > 2-\varepsilon = x_j^0,$$

that is, YS-EDF fails for sufficiently large m. □

YS-LLF_1

We define LLF_σ for machine capacity that varies over time. Similar to Algorithm 3.1, let $j_1, \dots, j_k$ be the available unfinished jobs at time t in order of their σ-laxities. Further,

let c be the available machine capacity at t. If $c/s \geq k$, each job $j_1, \ldots, j_k$ is run at speed s. Otherwise, each job with a strictly smaller σ-laxity than $j_{\lceil c/s \rceil}$ is run at speed s, and all the jobs with the same laxity as $j_{\lceil c/s \rceil}$ equally share the remaining machine capacity.

For $\sigma = 1$, however, it turns out that $\mathsf{YS\text{-}LLF}_\sigma$ is even worse than just LLF_σ.

Theorem 3.6. $\mathsf{YS\text{-}LLF}_1$ *is not a speed-s algorithm for any s.*

Proof. Suppose $\mathsf{YS\text{-}LLF}_1$ is a speed-s algorithm for some s. Our counter example is defined as follows and works for arbitrary m. All jobs are released at time 0. The m jobs with the smallest deadline each have processing time 1. We release another k jobs $j_1, \ldots, j_k$ with $d_{j_1} < \cdots < d_{j_k}$ and processing time $p_{j_i} = (m/(m-1))^i$. Note that each of these k jobs has initial laxity 0 in LLF_1, causing all of them to be run at the same speed until $x^0_{j_1}$. If we choose k large enough (depending on s and m), j_1 does not receive enough processing. □

$\mathsf{YS\text{-}LLF}_s$

The lower-bound instance for $\mathsf{YS\text{-}LLF}_1$ with $k = m - 1$ jobs is exactly the lower-bound instance that shows that no deadline-ordered algorithm (such as $\mathsf{YS\text{-}A}$ for any A) is a speed-s algorithm for any $s < e/(e-1)$ [LT99]. Now consider LLF_s instead of LLF_1, that is, the variant of LLF that uses its actual speed as laxity speed. Interestingly, it can be checked that $\mathsf{YS\text{-}LLF}_s$ needs to have *exactly* speed $e/(e-1)$ to schedule the lower-bound instance for $\mathsf{YS\text{-}LLF}_1$ feasibly, no matter what k is. In fact, we have the following conjecture.

Conjecture 3.3. $\mathsf{YS\text{-}LLF}_s$ *is a speed-$e/(e-1)$ algorithm.*

3.6 Open Problems

Even though online deadline scheduling with speed augmentation was first considered almost 20 years ago and serious efforts have been made within the community, there have only been small improvement upon upper and lower bounds. It is thus a very intriguing question whether there exists an algorithm with asymptotical speed requirement smaller than 2. While we conjecture that YSS does not improve upon this asymptotical requirement (Conjecture 3.2), it seems promising to consider $\mathsf{YS\text{-}LLF}_s$ since this algorithm may fix the main issue of YSS by leaving no capacity unused for underworked jobs (Conjecture 3.3). It also seems very plausible that considering the actual values of deadlines instead of only their order (like YSS and $\mathsf{YS\text{-}LLF}_s$ do) yields a (further) decrease in the speed requirement.

Chapter 4

Chasing Convex Bodies and Functions

Bibliographic information. Most of the results presented in this section are published in [ABN+16]. The lower bound for one-dimensional function chasing is taken from [BGK+15].

This section deals with the chasing problems that were informally introduced in Section 1.1. We first give a formal definition and necessary notations in Section 4.1. After naming related (Section 4.2) and new results (Section 4.3), we present our results for (lazy) convex-body chasing in Section 4.4, and we extend this problem to convex-function chasing in Section 4.5.

4.1 Preliminaries

We introduce three different chasing problems in increasing order of generality. We describe each of them for an arbitrary parameter $d \in \mathbb{N}$, which refers to the dimension of the Euclidean space they are set in. By $p_0 \in \mathbb{R}^d$ we denote the origin, and we use ρ to denote the standard Euclidean metric in $\mathbb{R}^d$.

Convex-Body Chasing. A server starting at p_0 receives an online sequence of convex bodies $B_0, \ldots, B_n \subseteq \mathbb{R}^d$. As a response to each body B_i, the server has to move into that body, that is, it has to specify a point $p_i \in B_i$ and move to it. The cost incurred is the total distance moved, that is,

$$\sum_{i=1}^{n} \rho(p_{i-1}, p_i).$$

Lazy Convex-Body Chasing. The server starts again at p_0, and part of its input is again a sequence of convex bodies $B_0, \ldots, B_n \subseteq \mathbb{R}^d$. The difference to convex-body chasing is that, along with each convex body B_i, a slope $\varepsilon_i \in \mathbb{R}_+$ is presented, and, as a response, the online algorithm may choose any point $p_i \in \mathbb{R}^d$ to move to. Moving to a point p_i outside B_i, however, incurs an additional cost (*laziness cost*) of $\varepsilon_i \rho(p_i, B_i)$ where $\rho(p_i, B_i) \coloneqq \inf_{p \in B_i} \rho(p_i, p)$. This yields a total cost of

$$\sum_{i=1}^{n} \rho(p_{i-1}, p_i) + \varepsilon_i \rho(p_i, B_i).$$

	Convex-Body Chasing	Lazy Convex-Body Chasing	Convex-Function Chasing
Input	sequence of convex bodies $B_1, \ldots, B_n \subseteq \mathbb{R}^d$	sequence of convex bodies $B_1, \ldots, B_n \subseteq \mathbb{R}^d$ along with slopes $\varepsilon_1, \ldots, \varepsilon_n \in \mathbb{R}_+$	sequence of convex functions $f_1, \ldots, f_n : \mathbb{R}^d \to \mathbb{R}_+$
Output	sequence of points $p_1 \in B_1, \ldots, p_n \in B_n$	sequence of points $p_1, \ldots, p_n \in \mathbb{R}^d$	sequence of points $p_1, \ldots, p_n \in \mathbb{R}^d$
Cost	$\sum_{i=1}^{n} \rho(p_{i-1}, p_i)$	$\sum_{i=1}^{n} \rho(p_{i-1}, p_i) + \varepsilon_i \rho(p_i, B_i)$	$\sum_{i=1}^{n} \rho(p_{i-1}, p_i) + f_i(p_i)$

Table 4.1: A comparison of the three different chasing problems considered in this chapter.

Convex-Function Chasing. Like in the previous two problems, the server starts at p_0. Instead of convex bodies and possibly slopes, the input is now a sequence of (differentiable) convex functions $f_1, \ldots, f_n : \mathbb{R}^d \to \mathbb{R}$. Similar to lazy convex-body chasing, the server may move to any $p_i \in \mathbb{R}^d$ as a response to a convex function f_i. In addition to the distance moved, the server incurs a cost (the *function cost*) of $f_i(p_i)$, leading to the total cost

$$\sum_{i=1}^{n} \rho(p_{i-1}, p_i) + f_i(p_i).$$

Remarks. We give a summary of the three problems in Table 4.1. Note that convex-body chasing is indeed a special case of lazy convex-body chasing where ε_i is chosen sufficiently large for all i. Lazy convex-body chasing is in turn a special case of convex-function chasing: To see this, observe that, for a convex body $B \subseteq \mathbb{R}^d$ and slope ε, the function $f(p) = \varepsilon\rho(p, B)$ for $p \in \mathbb{R}^d$ is a convex function.

Special Cases. We will occasionally consider subclasses of the class of convex bodies. For such a subclass $\mathcal{B}$, we call (lazy) $\mathcal{B}$ chasing that special case of (lazy) convex-body chasing where all convex bodies have to be elements of $\mathcal{B}$. Considered subclasses are hyperplanes, affine subspaces (that is, intersections of hyperplanes), and line segments.

Competitive Analysis. The goal for all considered problems is to be competitive (as introduced in Chapter 1). Let P be one the above problems, and denote its set of possible input sequences by $\mathcal{I}$. For $c \in \mathbb{R}_+$ and a deterministic algorithm A, we say that A is *c-competitive* for P if

$$|\mathsf{A}(I)| \leq c \cdot |\textsc{Opt}|,$$

for all instances $I \in \mathcal{I}$, where Opt denotes the minimum-cost solution for instance I and $|\cdot|$ denotes the cost of a solution. Accordingly, a randomized algorithm A (against an oblivious adversary) is called c-competitive for P if

$$\mathbb{E}[|\mathsf{A}(I)|] \leq c \cdot |\textsc{Opt}|.$$

Reductions. For some of the problems, we will not describe an explicit competitive algorithm but a reduction to another problem for which a competitive algorithm exists: Let P_1 and P_2 be chasing problems, let $\mathcal{I}_1$ and $\mathcal{I}_2$ be the sets of respective possible input sequences, and let $\mathcal{O}_1$ and $\mathcal{O}_2$ be the sets of respective possible output sequences. We call a pair of functions $f : \mathcal{I}_1 \to \mathcal{I}_2, g : \mathcal{O}_2 \to \mathcal{O}_1$ that can be computed online an *α-approximate reduction* from P_1 to P_2 if the following holds: For every c-competitive algorithm A for P_2 that outputs $\mathsf{A}(I_2)$ for $I_2 \in \mathcal{I}_2$, the algorithm that outputs $g(\mathsf{A}(f(I_1)))$ for input $I_1 \in \mathcal{I}_1$ is αc-competitive. When we describe these reductions, we will, for the sake of readability, implicitly give the functions f and g.

4.2 Related Work

Convex-body chasing was first considered by Friedman and Linial [FL93] as a fundamental online problem. They gave an 28.53-competitive algorithm for line chasing in an arbitrary number of dimensions. While they observe that simple greedy algorithms are not competitive for this problem, their algorithm is rather complicated: It is based on a reduction to a continuous version of the problem, and their algorithm for the latter problem works in phases that each consist of a mass of 2π of angle. During each phase, their algorithm moves greedily and, at the end of each phase, it moves towards a certain position at which a server that only moved little in the last phase could be. The analysis then uses the distance to the position of the optimum as a potential function.

Building up on their result for line chasing, they gave an $\mathcal{O}(1)$-competitive algorithm for convex-body chasing in two dimensions. It is easy to see that this problem can be reduced to halfplane chasing in two dimensions. Their algorithm for the latter problem again considers the continuous version of the problem and works in phases. To make a similar potential-function argument as before work, one would like to again, at the end of each phase, estimate the position of an optimal offline server and move towards it. To get a good estimate, however, Friedman and Linial need to distinguish various different cases regarding the geometry of the intersection of the halfspaces of the previous phase.

Friedman and Linial have also considered higher dimensions and have given a reduction from hyperplane chasing in d dimensions to lazy hyperplane chasing in $d-1$ dimensions, but they were not able to solve the latter problem even for $d = 3$. Instead, they gave a simple lower bound of $\sqrt{d}$ on the competitive ratio of any algorithm for hyperplane chasing in d dimensions. It however remained an open question whether competitive algorithms exist for hyperplane chasing in higher dimensions, which will be answered (positively) in this chapter.

Convex-function chasing has so far only been considered for $d = 1$. It was introduced by Lin et al. [LWAT13] as online convex optimization in the context of right-sizing of data centers. In the same paper, a 3-competitive algorithm was given which solves a convex program at each time step. A new randomized algorithm was presented in [ABL+13], and it was claimed to be 2-competitive. The analysis was, however, flawed [Wie15], but it has been fixed [ABL+15]. Independently, another randomized algorithm was given [BGK+15]. The authors also observed that any randomized algorithm can be derandomized without any loss in the competitive ratio. They also gave a simple 3-competitive memoryless algorithm, and showed that this ratio cannot be improved by any memoryless algorithm.

The same problem was also considered in the context of regret minimization [Han57], and it was shown that a sublinear-regret algorithm exists but that this is incompatible with $\mathcal{O}(1)$-competitiveness [ABL+13].

While convex-function chasing in higher dimensions cannot be found in the literature as such, the more general problem of metrical task systems can be found [BLS92]. Here, an arbitrary metrical space is considered, and the assumption of convexity is dropped. The optimal competitive ratio for deterministic algorithms is settled to be $2k-1$ [BLS92], where k is the cardinality of the base set of the considered metric. On the other hand, for randomized algorithms, there remains a gap between $\Omega(\log n/\log\log n)$ [BLMN03, BBM06] and $\mathcal{O}(\log^2 n \log\log n)$ [FM03].

4.3 Contribution and Outline

The main result of this chapter is a $2^{\mathcal{O}(d)}$-competitive algorithm for affine-subspace chasing in d dimensions, which is an $\mathcal{O}(1)$-competitive algorithm when d is fixed. To obtain this result, we first use a simple insight [FL93] that reduces this problem to hyperplane chasing in d dimensions. To get a competitive algorithm for d-dimensional hyperplane chasing, we first apply an $\mathcal{O}(1)$-approximate reduction to $(d-1)$-dimensional lazy hyperplane chasing (Subsection 4.4.1), which is a discretized version of a reduction already given in [FL93]. This problem we can then reduce to $(d-1)$-dimensional (non-lazy) hyperplane chasing by a new reduction that we describe below. Consequently, by composing both reductions, we get an $\mathcal{O}(1)$-approximate reduction from d-dimensional hyperplane chasing to $(d-1)$-dimensional hyperplane chasing.

One problem that we can reduce to for which an explicit $\mathcal{O}(1)$-algorithm is known is two-dimensional line chasing [FL93]. We simplify the algorithm, give a simpler analysis, and slightly improve the competitive ratio from 28.5 to 28.1 (Subsection 4.4.3). We also note that our algorithm can be generalized to an algorithm for line-segment chasing in arbitrary dimensions with the same competitive ratio. Alternatively to using one of these $\mathcal{O}(1)$-competitive algorithms for line chasing in two dimensions, we can also apply our dimension reduction another time and reduce to the trivial problem of point chasing in one dimension, for which clearly a 1-competitive algorithm exists. In Figure 4.1, we illustrate how we obtain the $2^{\mathcal{O}(d)}$-competitive algorithm for hyperplane chasing in d dimensions.

To obtain out new reduction, we first formulate a *randomized* reduction for lazy convex-body chasing to (non-lazy) convex-body chasing, which is then derandomized (Subsection 4.4.2). Intuitively, a convex body's relevance in lazy convex-body chasing correlates with its slope, and slopes can be assumed w.l.o.g. to be at most 1. Our algorithm for lazy convex-body chasing simply hands over a convex body with slope ε to the algorithm for convex-body chasing with probability ε, and discards it with probability $1-\varepsilon$. Our algorithm then moves according to the line-chasing algorithm, and the deterministic version of it moves to the expected position of the randomized one, taken over all random events so far. An analysis that elegantly relates slopes to probabilities proves that this reduction is indeed $\mathcal{O}(1)$-approximate.

We also give a new lower bound of 1.86 on the competitive ratio of any function-chasing algorithm in one dimension (Subsection 4.5).

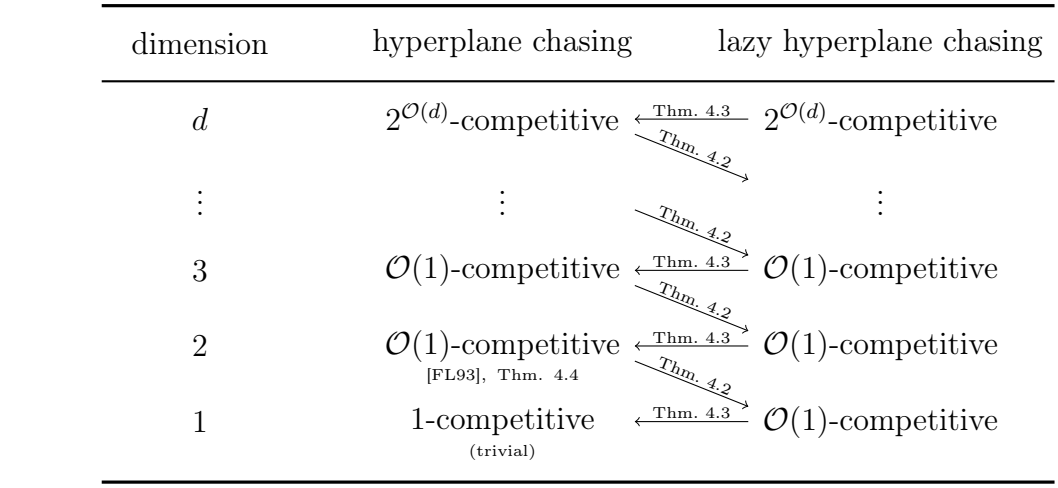

Figure 4.1: The interplay of our reductions that yields a $2^{\mathcal{O}(d)}$-competitive algorithm for (lazy) hyperplane chasing in d dimensions.

We remark that, up to constants factors, our competitiveness results also hold for other metrics whose values are guaranteed to be within constant factors of that of the Euclidean metric, such as other L^p norms.

4.4 Convex-Body Chasing

The goal of this section is proving the following theorem.

Theorem 4.1. *For any $d \in \mathbb{N}$, there is a $2^{\mathcal{O}(d)}$-competitive algorithm for affine-subspace chasing in d dimensions.*

As shown by Friedman and Linial [FL93], each affine subspace can be replaced by a sufficient number of hyperplanes intersecting in this affine subspace. It then suffices to show that there is a $2^{\mathcal{O}(d)}$-competitive algorithm for *hyperplane* chasing in d dimensions. To do so, the proof strategy (illustrated in Figure 4.1) is the following. We first show the discrete version of the reduction from hyperplane chasing in d dimensions to lazy hyperplane chasing in $d-1$ dimensions [FL93] (Subsection 4.4.1). We then present a new reduction from lazy hyperplane chasing to (non-lazy) hyperplane chasing in $d-1$ dimensions (Subsection 4.4.2). Even though this already suffices to prove the theorem using the $\mathcal{O}(1)$-competitive algorithm for line chasing [FL93] in two dimensions or the trivial 1-competitive algorithm for point chasing in one dimensions, we give a simplified algorithm for line chasing in Subsection 4.4.3.

4.4.1 A Dimension Reduction for Hyperplanes

In this subsection, we prove the following theorem.

Theorem 4.2. *There exists an $\mathcal{O}(1)$-approximate deterministic reduction from hyperplane chasing in $\mathbb{R}^d$ to lazy hyperplane chasing in $\mathbb{R}^{d-1}$.*

We reduce via an auxiliary problem, hyperplane chasing without consecutive parallel hyperplanes. We split the proof into two lemmata, the first of which contains the reduction to this problem.

Lemma 4.1. *In $\mathbb{R}^d$, there exists an $\mathcal{O}(1)$-approximate deterministic reduction from hyperplane chasing (HC) to hyperplane chasing without consecutive parallel hyperplanes (HC*).*

Proof. Assume we are given a c_{HC^*}-competitive algorithm $\mathsf{A}_{\mathrm{HC}^*}$ for the HC* problem. We describe how to transform it into an c_{HC}-competitive algorithm A_{HC} for the HC problem where $c_{\mathrm{HC}} = \mathcal{O}(1) \cdot c_{\mathrm{HC}^*}$. Every time a new hyperplane B_i arrives that is not parallel to B_{i-1}, we hand it over to $\mathsf{A}_{\mathrm{HC}^*}$ and move accordingly (subject to translations described below). Otherwise we simply project the old position onto B_i. We do not hand over any plane to $\mathsf{A}_{\mathrm{HC}^*}$, but we modify all forthcoming planes and movements: Let v be the unit normal vector of the two planes pointing from B_i towards B_{i-1}, and let d be the distance of the two planes. Every time a plane $B_{i'}$ with $i' > i$ is handed over to $\mathsf{A}_{\mathrm{HC}^*}$, we translate it by $d \cdot v$, subject to further translations due to further parallel planes. When moving according to $\mathsf{A}_{\mathrm{HC}^*}$'s output, we translate back this point along all the normal vectors that the current plane was translated by.

Now consider $\textsc{Opt}_{\mathrm{HC}}$ in the HC problem. We note that, at a total loss of a factor of $\sqrt{2}$ in the costs, every time a plane B_i arrives that is parallel to the previous one B_{i-1}, we can replace its position p_i by the projection p'_i of p_{i-1} onto B_i. To see this, consider a maximal connected sequence of parallel hyperplanes $B_i, \dots, B_j$ and compute

$$\begin{aligned}\rho(p_i, p'_{i+1}) + \sum_{k=i+1}^{j-1} \rho(p'_k, p'_{k+1}) + \rho(p'_j, p_{j+1}) &\le \rho(p_i, p'_j) + \rho(p'_j, p_j) + \rho(p_j, p_{j+1}) \\ &\le \sqrt{2} \cdot \rho(p_i, p_j) + \rho(p_j, p_{j+1}) \\ &\le \sqrt{2} \cdot \sum_{k=i}^{j-1} \rho(p_k, p_{k+1}) + \rho(p_j, p_{j+1}),\end{aligned}$$

where we use the triangle inequality in the first and third step. In the second step, we use the fact that the triangle formed by p_i, p'_j, and p_j has a right angle at p'_i. By applying the inequality for all maximal connected sequences of parallel hyperplanes, we get indeed a loss of a factor at most $\sqrt{2}$.

Now consider an input sequence $I_{\mathrm{HC}} = B_i, \dots, B_n$ for the HC problem and the corresponding sequence I_{HC^*} for the HC* problem. We denote the corresponding optima by $\textsc{Opt}_{\mathrm{HC}}$ and $\textsc{Opt}_{\mathrm{HC}^*}$, respectively, and the output sequence of A_{HC} is $p_1^A, \dots, p_n^A$. Us-

ing our reduction, we get

$$\begin{aligned}
|\mathsf{A}_{\mathrm{HC}}(I_{\mathrm{HC}})| &= \sum_{\substack{i:B_i,B_{i+1}\\ \text{not parallel}}} \rho(p_i^A, p_{i+1}^A) + \sum_{\substack{i:B_i,B_{i+1}\\ \text{parallel}}} \rho(B_i, B_{i+1}) \\
&= |\mathsf{A}_{\mathrm{HC}^*}(I_{\mathrm{HC}^*})| + \sum_{\substack{i:B_i,B_{i+1}\\ \text{parallel}}} \rho(B_i, B_{i+1}) \\
&\leq c_{\mathrm{HC}^*} \cdot |\textsc{Opt}_{\mathrm{HC}^*}| + \sum_{\substack{i:B_i,B_{i+1}\\ \text{parallel}}} \rho(B_i, B_{i+1}) \\
&\leq \sqrt{2} \cdot c_{\mathrm{HC}^*} \cdot |\textsc{Opt}_{\mathrm{HC}}|,
\end{aligned}$$

where the last step follows by noticing that $\textsc{Opt}_{\mathrm{HC}^*}$ defines a solution to the HC problem and applying the above considerations. □

We now give the second part of the reduction.

Lemma 4.2. *There exists an $\mathcal{O}(1)$-approximate deterministic reduction from hyperplane chasing without consecutive parallel hyperplanes (HC*) in $\mathbb{R}^d$ to lazy hyperplane chasing (LHC) in $\mathbb{R}^{d-1}$.*

Proof. Consider an arbitrary instance $I_{\mathrm{HC}} = B_1^{\mathrm{HC}}, \ldots, B_n^{\mathrm{HC}}$ of the HC* problem in $\mathbb{R}^d$ (for simplicity just called hyperplane chasing or HC in the following). We will show how to transform I_{HC} into the instance $I_{\mathrm{LHC}} = B_1^{\mathrm{LHC}}, \ldots, B_n^{\mathrm{LHC}}$ of the LHC problem in $\mathbb{R}^{d-1}$, and how to translate back a solution $p_1^{\mathrm{LHC}}, \ldots, p_n^{\mathrm{LHC}}$ for I_{LHC} to a solution $p_1^{\mathrm{HC}}, \ldots, p_n^{\mathrm{HC}}$ for I_{HC}. We will then show that, for any single step from p_i^{LHC} to p_{i+1}^{LHC} taken in the LHC problem and the corresponding step from p_i^{HC} to p_{i+1}^{HC} in the HC problem, the costs of both only differ by a constant factor. Formally, we show that there exist constants c_1 and c_2 such that

$$c_1 \leq \frac{\rho(p_i^{\mathrm{HC}}, p_{i+1}^{\mathrm{HC}})}{\rho(p_i^{\mathrm{LHC}}, p_{i+1}^{\mathrm{LHC}}) + \varepsilon_i \rho(p_{i+1}^{\mathrm{LHC}}, B_{i+1}^{\mathrm{LHC}})} \leq c_2 \tag{4.1}$$

for any i and positions p_i^{LHC} and p_{i+1}^{LHC}. This will directly imply that our reduction is $\mathcal{O}(1)$-approximate, i.e., any c_{LHC}-competitive algorithm- for LHC in $\mathbb{R}^{d-1}$ can be transformed into a c_{HC}-competitive algorithm for HC in $\mathbb{R}^d$ with $c_{\mathrm{HC}} = \mathcal{O}(1) \cdot c_{\mathrm{LHC}}$.

The reduction works as follows. At all times, we keep an isometric (i.e., distance-preserving) mapping f_i whose domain is the $\mathbb{R}^{d-1}$ space that A_{LHC} moves in. While the target set of f_0 is an arbitrary hyperplane $B_0^{\mathrm{HC}} \subset \mathbb{R}^d$ that contains p_0^{HC} and is not parallel to B_1^{HC}, the target set of f_i for $i > 0$ is the hyperplane B_i^{HC}. Consequently, each f_i can be viewed as a coordinate system within B_i^{HC}, and it is bijective. Initially, we only require $f_0(p_0^{\mathrm{LHC}}) = p_0^{\mathrm{HC}}$. In the same way, this function establishes the relation between p_i^{LHC} and p_i^{HC} for $i > 1$:

$$p_i^{\mathrm{HC}} := f_i(p_i^{\mathrm{LHC}}).$$

In particular, when a new hyperplane B_i^{HC} arrives, it intersects B_{i-1}^{HC} in a $(d-2)$-dimensional subspace S_i, and we obtain f_i by rotating the coordinate system in B_{i-1}^{HC} given by f_{i-1} around S_i onto B_i^{HC}. More formally,

$$f_i(x) := R_i \cdot f_{i-1}(x), \text{ for all } x \in \mathbb{R}^{d-1},$$

where R_i is the d-dimensional rotation matrix around S_i through the angle of intersection α_i of B_{i-1}^{HC} and B_i^{HC}. Obviously, f_i inherits isometry from f_{i-1}. Now the hyperplane that we present in the LHC problem is S_i within the current coordinate system, that is,

$$B_i^{\mathrm{LHC}} := f_i^{-1}(S_i) = f_i^{-1}(B_{i-1}^{\mathrm{HC}} \cap B_i^{\mathrm{HC}}),$$

where f_i^{-1} exists because f_i is bijective. We also set

$$\varepsilon_i := \alpha_i,$$

where α_i is given in radians, which will be shown to make the costs of both algorithms comparable.

In the remainder of this proof, we show Inequality (4.1) for all i. We first observe that instead of moving from p_i^{HC} to p_{i+1}^{HC} via the direct path, we can also move the following way. We first move to $R_{i+1}^{-1} \cdot p_{i+1}^{\mathrm{HC}}$ (i.e., p_{i+1}^{HC} rotated back onto B_i^{HC}) via the direct path and then to p_{i+1}^{HC} via the arc that $R_i^{-1} \cdot p_{i+1}^{\mathrm{HC}}$ follows when rotated onto B_{i+1}^{HC}. To obtain the lower bound in Inequality (4.1), we also observe that this movement exactly incurs the costs in the LHC problem:

$$\begin{aligned}\rho(p_i^{\mathrm{HC}}, p_{i+1}^{\mathrm{HC}}) &\leq \rho(p_i^{\mathrm{HC}}, R_{i+1}^{-1} \cdot p_{i+1}^{\mathrm{HC}}) + \alpha_{i+1} \cdot \rho(R_{i+1}^{-1} \cdot p_{i+1}^{\mathrm{HC}}, S_{i+1}) \\ &= \rho(p_i^{\mathrm{LHC}}, p_{i+1}^{\mathrm{LHC}}) + \varepsilon_{i+1} \cdot \rho(p_{i+1}^{\mathrm{LHC}}, B_{i+1}^{\mathrm{LHC}}),\end{aligned}$$

where we use the triangle inequality in the first step and the fact that f_{i+1} and the rotation induced by R_{i+1} are isometries in the second step.

For the upper bound, we consider the triangle formed by the points used above, i.e., p_i^{HC}, $R_{i+1}^{-1} \cdot p_{i+1}^{\mathrm{HC}}$, and p_{i+1}^{HC}. We note that the length of the side between p_i^{HC} and p_{i+1}^{HC} is exactly the cost incurred in the HC problem. Next, we show that the sum of lengths of the two other sides is by a constant factor comparable to the costs incurred in the LHC problem. Towards this, consider the unique (2-dimensional) plane in which the (above) arc that $R_{i+1}^{-1} \cdot p_{i+1}^{\mathrm{HC}}$ follows when rotated onto B_{i+1}^{HC} lies, and let r be the plane's intersection point with S_{i+1}. As R_{i+1}^{-1} is a rotation matrix, the distance from q to both p_{i+1}^{HC} and $R_{i+1}^{-1} \cdot p_{i+1}^{\mathrm{HC}}$ is identical, meaning that the triangle formed by these three points is equal-sided. Since α_{i+1} is the angle at r in this triangle, we can compute:

$$\begin{aligned}\rho(R_{i+1}^{-1} \cdot p_{i+1}^{\mathrm{HC}}, p_{i+1}^{\mathrm{HC}}) &= 2\cos\left(\frac{\alpha_{i+1}}{2}\right) \cdot \rho(p_{i+1}^{\mathrm{HC}}, q) \\ &= 2\cos\left(\frac{\alpha_{i+1}}{2}\right) \cdot \rho(p_{i+1}^{\mathrm{HC}}, B_{i+1}^{\mathrm{HC}}) \\ &\geq \frac{2\sqrt{2}}{\pi} \cdot \alpha_{i+1} \cdot \rho(p_{i+1}^{\mathrm{LHC}}, B_{i+1}^{\mathrm{LHC}}). \qquad (4.2)\end{aligned}$$

Here the last bound follows by optimizing over all $\alpha_{i+1} \in (0, \pi/2]$ (recall that α_{i+1} is the smaller angle of intersection) and again using that f_{i+1} is an isometry. Using the latter fact once more and Inequality (4.2), we get

$$\begin{aligned}&\rho(p_i^{\mathrm{HC}}, R_{i+1}^{-1} \cdot p_{i+1}^{\mathrm{HC}}) + \rho(R_{i+1}^{-1} \cdot p_{i+1}^{\mathrm{HC}}, p_{i+1}^{\mathrm{HC}}) \\ \geq &\frac{2\sqrt{2}}{\pi} \cdot \left(\rho(p_i^{\mathrm{LHC}}, p_{i+1}^{\mathrm{LHC}}) + \varepsilon_{i+1} \cdot \rho(p_{i+1}^{\mathrm{LHC}}, B_{i+1}^{\mathrm{LHC}})\right), \qquad (4.3)\end{aligned}$$

as claimed.

Thus, it remains to show that the length of the side between p_i^{HC} and p_{i+1}^{HC} is by a constant factor comparable to the sum of lengths of the two other sides of the considered triangle (formed by the latter two and $R_{i+1}^{-1} \cdot p_{i+1}^{\mathrm{HC}}$). To this end, note that the angle of intersection β between B_{i+1}^{HC} and the line connecting $R_{i+1}^{-1} \cdot p_{i+1}^{\mathrm{HC}}$ and p_{i+1}^{HC} is $(\pi - \alpha_{i+1})/2 \in [\pi/4, \pi/2]$ by the considerations above. Thus, the angle at $R_{i+1}^{-1} \cdot p_{i+1}^{\mathrm{HC}}$ in the triangle is some $\gamma \in [\beta, \pi - \beta] \subseteq [\pi/4, 3\pi/4]$. Applying the law of sines, we have for the diameter d of the triangle's circumcircle that

$$\begin{aligned}
\rho(p_i^{\mathrm{HC}}, p_{i+1}^{\mathrm{HC}}) &= \sin(\gamma) \cdot d \\
&\geq \frac{\sin\gamma}{2} \cdot \left(\rho(R_{i+1}^{-1} \cdot p_{i+1}^{\mathrm{HC}}, p_i^{\mathrm{HC}}) + \rho(R_{i+1}^{-1} \cdot p_{i+1}^{\mathrm{HC}}, p_{i+1}^{\mathrm{HC}})\right) \\
&\geq \frac{1}{2\sqrt{2}} \cdot \left(\rho(R_{i+1}^{-1} \cdot p_{i+1}^{\mathrm{HC}}, p_i^{\mathrm{HC}}) + \rho(R_{i+1}^{-1} \cdot p_{i+1}^{\mathrm{HC}}, p_{i+1}^{\mathrm{HC}})\right), \qquad (4.4)
\end{aligned}$$

using d as an upper bound for any side of the triangle in the first step, and $\gamma \in [\pi/4, 3\pi/4]$ in the second step. Now we combine Inequalities (4.3) and (4.4) to

$$\rho(p_i^{\mathrm{HC}}, p_{i+1}^{\mathrm{HC}}) \geq \frac{1}{\pi} \cdot \left(\rho(p_i^{\mathrm{LHC}}, p_{i+1}^{\mathrm{LHC}}) + \varepsilon_{i+1} \cdot \rho(p_{i+1}^{\mathrm{LHC}}, B_{i+1}^{\mathrm{LHC}})\right),$$

which gives the claim. □

This completes the proof of Theorem 4.2.

4.4.2 Eliminating Laziness

In this subsection, we show the following theorem.

Theorem 4.3. *Let $\mathcal{B}$ be a class of convex bodies in d dimensions. Then there is an $\mathcal{O}(1)$-approximate deterministic reduction from lazy $\mathcal{B}$ chasing to (non-lazy) $\mathcal{B}$ chasing.*

Note that, in particular, by setting $\mathcal{B}$ to be the class of *all* convex bodies, this theorem yields a reduction from lazy convex-body chasing to (non-lazy) convex-body chasing.

Proof. We build a randomized $\mathcal{O}(c_{\mathrm{C}})$-competitive algorithm A_{LC} for lazy-convex body chasing from a deterministic c_{C}-competitive algorithm A_{C} for (non-lazy) convex-body chasing and then explain how to derandomize it. We first modify the input instance by replacing each lazy convex body (B_i, ε_i) whose slope ε_i is greater than 1 by $\lceil \varepsilon_i \rceil$ lazy convex bodies, each having B_i as the convex body. The first $\lfloor \varepsilon_i \rfloor$ of these lazy convex bodies will have slope 1, and the potentially remaining convex body will have slope $\varepsilon_i - \lfloor \varepsilon_i \rfloor$. This modification does not affect the optimal cost and will not decrease the online cost. From now on, we will assume that any input instance for lazy convex-body chasing is of this modified form. It is easy to see how one can go back from a solution to the modified input to one to the original input without increasing the cost, since our algorithm w.l.o.g. never moves away from the convex body that just arrived.

Now A_{LC} is defined as follows. Upon the arrival of a new lazy convex body (B_i, ε_i), the algorithm with (independent) probability $1 - \varepsilon_i$ does not move, and with probability ε_i passes B_i to A_{C} and moves to the location to which A_{C} moves. Notice that in the modified

input instance every slope is a real number in $[0, 1]$, and thus probabilities ε_i and $1 - \varepsilon_i$ are all well defined.

Consider a particular input instance I_{LC} of lazy convex-body chasing, as defined before. Let I_{C} denote the random variable representing the sequence of convex bodies passed to A_{C}. Let $\textsc{Opt}_{\mathrm{LC}}$ be the optimal solution for lazy convex-body chasing on I_{LC}. Let $\textsc{Opt}_{\mathrm{C}}$ be a random variable equal to the optimal solution for convex-body chasing on I_{C}.

The $\mathcal{O}(c_{\mathrm{C}})$-competitiveness of A_{LC} will now follow from the following sequence of inequalities:

$$\begin{aligned}
\mathbb{E}[|\mathsf{A}_{\mathrm{LC}}(I_{\mathrm{LC}})|] &= \mathcal{O}(\mathbb{E}[|\mathsf{A}_{\mathrm{C}}(I_{\mathrm{C}})|]) && (4.5)\\
&= \mathcal{O}(c_{\mathrm{C}} \cdot \mathbb{E}[|\textsc{Opt}_{\mathrm{C}}|]) && (4.6)\\
&= \mathcal{O}(c_{\mathrm{C}} \cdot |\textsc{Opt}_{\mathrm{LC}}|). && (4.7)
\end{aligned}$$

We show the inequalities one by one, starting with Inequality (4.5). We explain the following sequence of inequalities below:

$$\begin{aligned}
\mathbb{E}[|\mathsf{A}_{\mathrm{LC}}(I_{\mathrm{LC}})|] &= \sum_i \mathbb{P}[B_i \in I_{\mathrm{C}}]\ \mathbb{E}[\text{Cost of } \mathsf{A}_{\mathrm{C}} \text{ on } B_i \mid B_i \in I_{\mathrm{C}}]\\
&\quad + \sum_i \mathbb{P}[B_i \notin I_{\mathrm{C}}]\ \mathbb{E}[\text{Cost of } \mathsf{A}_{\mathrm{LC}} \text{ on } B_i \mid B_i \notin I_{\mathrm{C}}] && (4.8)\\
&= \sum_i \varepsilon_i\, \mathbb{E}[\text{Cost of } \mathsf{A}_{\mathrm{C}} \text{ on } B_i \mid B_i \in I_{\mathrm{C}}]\\
&\quad + \sum_i (1 - \varepsilon_i)\ \mathbb{E}[\varepsilon_i\, \rho(p_{i-1}, B_i)] && (4.9)\\
&\leq \sum_i \varepsilon_i\, \mathbb{E}[\text{Cost of } \mathsf{A}_{\mathrm{C}} \text{ on } B_i \mid B_i \in I_{\mathrm{C}}]\\
&\quad + \sum_i \varepsilon_i\, \mathbb{E}[\rho(p_{i-1}, B_i)] && (4.10)\\
&\leq 2 \sum_i \varepsilon_i\, \mathbb{E}[\text{Cost of } \mathsf{A}_{\mathrm{C}} \text{ on } B_i \mid B_i \in I_{\mathrm{C}}] && (4.11)\\
&= 2 \sum_i \mathbb{P}[B_i \in I_{\mathrm{C}}]\ \mathbb{E}[\text{Cost of } \mathsf{A}_{\mathrm{C}} \text{ on } B_i \mid B_i \in I_{\mathrm{C}}] && (4.12)\\
&= 2\, \mathbb{E}[|\mathsf{A}_{\mathrm{C}}(I_{\mathrm{C}})|] && (4.13)
\end{aligned}$$

Equation (4.8) follows from the definition of expectation, conditional expectation, and linearity of expectation. Notice that all expectations involving B_i only depend on the history up until B_i arrives. Equation (4.9) follows from the fact that B_i is added to I_{C} with probability ε_i, and, if B_i is not added, then A_{LC} pays ε_i times the distance to B_i. Inequality (4.10) follows from the linearity of expectation and since $1 - \varepsilon_i \leq 1$. Inequality (4.11) holds since A_{C} has to move to each $B_i \in I_{\mathrm{C}}$ and thus in expectation has to pay at least $\mathbb{E}[\rho(p_{i-1}, B_i)]$ (note that, only by independence of the random events, the expected position of A_{C} in case $B_i \in I_{\mathrm{C}}$ is identical to the expected position of A_{LC}). Equation (4.12) holds since $B_i \in I_{\mathrm{C}}$ with probability ε_i. Equation (4.13) follows by

linearity of expectation and the definition of conditional expectation. Hence, we get Inequality (4.5).

Next, note that Inequality (4.6) follows by the assumption that A_{C} is c_{C}-competitive. It remains to show Inequality (4.7), for which it suffices to construct a solution S for I_{C} with expected cost $\mathcal{O}(|\textsc{Opt}_{\mathrm{LC}}|)$. To do so, for each i denote by p_i the position that $\textsc{Opt}_{\mathrm{LC}}$ moves to as a response to B_i. We define two types of costs for S, *basic* and *detour* costs (which will be compared to movement and laziness cost, respectively): For each $B_i \in I_{\mathrm{C}}$, S first moves to the last point $p_{i'}$ with $i' < i$ and $B_{i'} \in I_{\mathrm{C}}$ (if it exists) incurring detour cost. Then it moves to p_i incurring basic cost, and then the shortest way possible to B_i incurring detour cost again. Now, by the triangle inequality, the basic cost of S is at most the movement cost of $\textsc{Opt}_{\mathrm{LC}}$. The probability that S incurs a detour cost for convex body B_i in step i is ε_i, and, when it incurs a detour cost, this detour cost is at most $2/\varepsilon_i$ times the laziness cost incurred by $\textsc{Opt}_{\mathrm{LC}}$. Thus the expected cost for S on I_{T} is $\mathcal{O}(|\textsc{Opt}_{\mathrm{LC}}|)$. This proves that A_{LC} is $\mathcal{O}(c_{\mathrm{C}})$-competitive.

As in [BGK+15], we derandomize A_{LC} to get a deterministic algorithm A_{D} by always moving to the expected position of A_{LC}. We will show that $\mathsf{A}_{\mathrm{LC}}^{\mathrm{D}}$ has the same competitive ratio as A_{LC}. More specifically, let p_i be a random variable denoting the position of A_{LC} as a response to convex body B_i. Then $\mathsf{A}_{\mathrm{LC}}^{\mathrm{D}}$ sets its position to $p_i^{\mathrm{D}} := \mathbb{E}[p_i]$.

We analyze the competitive ratio of this algorithm: We have that for each step i, A_{LC}'s expected cost is $\mathbb{E}[\rho(p_{i-1}, p_i)] + \varepsilon_i \mathbb{E}[\rho(p_i, B_i)]$. On the other hand, $\mathsf{A}_{\mathrm{LC}}^{\mathrm{D}}$'s cost is

$$\rho(\mathbb{E}[x_i], \mathbb{E}[p_{i-1}^{\mathrm{D}}]) + \varepsilon_i \rho(\mathbb{E}[p_i^{\mathrm{D}}], B_i) = \rho(\mathbb{E}[x_i], \mathbb{E}[p_{i-1}]) + \varepsilon_i \rho(\mathbb{E}[p_i], B_i).$$

By a generalization of Jensen's inequality (e.g., [MO79, Proposition B.1]), and by the convexity of our distance function ρ, we have, for each i

$$\mathbb{E}[\rho(p_i, p_{i-1})] \geq \rho(\mathbb{E}[p_i], \mathbb{E}[p_{i-1}])$$

and

$$\mathbb{E}[\rho(p_i, B_i)] \geq \rho(\mathbb{E}[p_i], B_i).$$

Summing over all i completes the analysis. □

4.4.3 A Simple Line-Chasing Algorithm

We show the following theorem.

Theorem 4.4. *There is an $\mathcal{O}(1)$-competitive for line-segment chasing in any dimension.*

For full lines, this theorem was already shown by Friedman and Linial [FL93]. If we restrict our algorithm to full lines, it is significantly simpler than their algorithm.

Our algorithm is stated as follows. Let P_i be a plane containing the line E_i that is parallel to the line E_{i-1} (note this is uniquely defined when E_i and E_{i-1} are not parallel). The algorithm first moves to the closest point $h_i \in P_i$. The algorithm then moves to the closest point g_i in E_i.

- If E_{i-1} and E_i are parallel, then the algorithm stays at g_i.

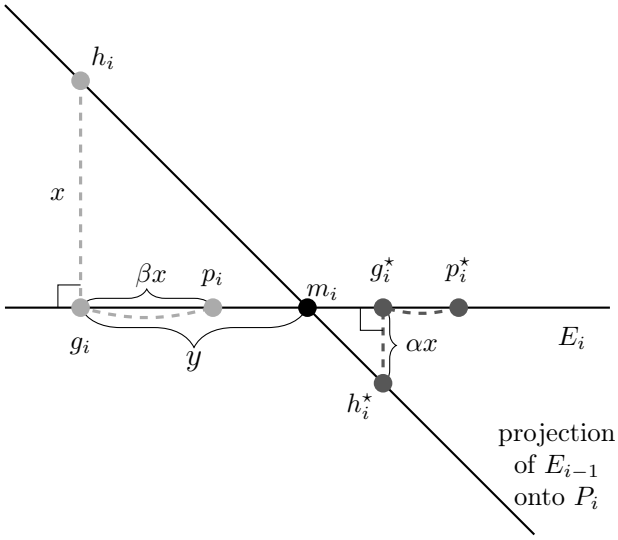

Figure 4.2: An overview of the used points and distances in P_i.

- If E_{i-1} and E_i are not parallel, let m_i be the intersection of E_i and the projection of E_{i-1} onto P_i. Let $\beta \in (0, 1)$ be some constant that we shall set later. The algorithm makes an adjustment by moving toward m_i along E_i until it has traveled a distance of $\beta \cdot \rho(h_i, g_i)$, or until it reaches m_i, whatever happens first.

- Finally, the algorithm moves to the closest point p_i in B_i.

We show that this algorithm satisfies the statement of the theorem.

Proof. Initially assume that all the line segments are lines. Let $h_i^\star$ be the projection of the adversary's position, just before B_i arrives, onto P_i. We will assume that the adversary first moves to $h_i^\star$, and then to the nearest point $g_i^\star$ on B_i, and then to some arbitrary final point $p_i^\star$ on B_i. This assumption increases the adversary's movement cost by at most a factor of $\sqrt{3}$ (a similar observation is made in [FL93]). We will charge the algorithm's cost for moving to P_i to the adversary's cost of moving to P_i. Thus by losing at most a factor of $\sqrt{3}$ in the competitive ratio, we can assume that the adversary and the algorithm are at positions $h_i^\star$ and h_i right before B_i arrives.

Let $\textsc{Alg}_i$ and $\textsc{Opt}_i$ denote the movement cost of the algorithm and the adversary, respectively, in response to B_i. Let the potential function Φ_i be $\delta \cdot \rho(p_i, p_i^\star)$ for some to be determined constant $\delta > 1$. To show that the algorithm is c-competitive it will be sufficient to show that, for each i,

$$\textsc{Alg}_i + \Phi_i - \Phi_{i-1} \leq c \cdot \textsc{Opt}_i. \tag{4.14}$$

We will only initially consider the adversary's movement cost until it reaches $g_i^\star$. Equation (4.14) will continue to hold for any adversary's movement after $g_i^\star$ if

$$c \geq \delta. \tag{4.15}$$

We consider three cases. The notation that we will use is illustrated in Figure 1.

In the first case assume $\rho(h_i^\star, g_i^\star) \geq \gamma x$, where $x = \rho(h_i, g_i)$, and $\gamma > 0$ is a constant that we define later. Intuitively, this is the easiest case as the adversary's cost will pay for both the algorithm's movement cost and the increase in the potential due to this movement. For Equation (4.14) to hold, it is sufficient that

$$(1+\beta)x + \delta((1+\beta)x + \gamma x) \leq c\gamma x,$$

or equivalently

$$c \geq \frac{(1+\beta) + \delta((1+\beta)+\gamma)}{\gamma}. \tag{4.16}$$

In the remaining two cases assume that $\rho(h_i^\star, g_i^\star) = \alpha x \leq \gamma x$, and let $y = \rho(m_i, g_i)$.

In the second case assume that $x \leq y$. Intuitively in this case the decrease in the potential due to the algorithm's adjustment on B_i decreases the potential enough to pay for the algorithm's movement costs. Since $\beta < 1$ and $x \leq y$, the algorithm will not have to stop at m_i when moving a distance of βx on B_i toward m_i. If

$$\gamma \leq 1 - \beta, \tag{4.17}$$

the algorithm will also not cross $g_i^\star$ while adjusting on B_i, as this would contradict the assumption that $\rho(h_i^\star, g_i^\star) \leq \gamma x$. Equation (4.14) will be hardest to satisfy when $\Phi_i - \Phi_{i-1}$ is maximal. This will occur when $g_i^\star$ is maximally far from g_i, which in turn occurs when the points $g_i^\star$, m_i, and g_i lie on B_i in that order. In that case, Equation (4.14) evaluates to

$$(1+\beta)x + \delta((1+\alpha)y - \beta x - (1+\alpha)\sqrt{x^2+y^2}) \leq c\alpha x.$$

Setting $L = \frac{1+\beta-\delta\beta-c\alpha}{\delta(1+\alpha)}$, this is equivalent to

$$Lx + y \leq \sqrt{x^2+y^2}. \tag{4.18}$$

When $x \leq y$, Equation (4.18) holds if $L \leq 0$. This in turn holds when

$$\delta \geq \frac{1+\beta}{\beta}. \tag{4.19}$$

Finally we consider the third case that $x \geq y$. Intuitively in this case the decrease in the potential due to the algorithm's movement toward B_i decreases the potential enough to pay for the algorithms movement costs. First consider the algorithm's move from p_{i-1} to g_i. Again, the value of $\Phi_i - \Phi_{i-1}$ is maximized when $g_i^\star$ is on the other side of m_i as is g_i. In that case, the value of $\Phi_i - \Phi_{i-1}$ due to this move is at most

$$\delta((1+\alpha)y - (1+\alpha)\sqrt{x^2+y^2}).$$

Note that the maximum possible increase in the potential due to the algorithm moving from g_i to p_i is $\delta\beta x$. Thus for Equation (4.14) to hold, it is sufficient that

$$(1+\beta)x + \delta\beta x + \delta((1+\alpha)y - (1+\alpha)\sqrt{x^2+y^2}) \leq c\alpha x.$$

Setting $L = \frac{1+\beta+\delta\beta-c\alpha}{\delta(1+\alpha)}$, this is equivalent to

$$Lx + y \leq \sqrt{x^2+y^2}. \tag{4.20}$$

When $x \geq y$, Equation (4.20) holds if $L \leq \sqrt{2} - 1$. This in turn holds when

$$\delta \geq \frac{1+\beta}{\sqrt{2}-1-\beta}. \tag{4.21}$$

We now need to find a feasible setting of β, γ, δ, and compute the resulting competitive ratio. Setting $\beta = \frac{\sqrt{2}-1}{2}$ and $\gamma = \frac{3-\sqrt{2}}{2}$, and $\delta = \frac{\sqrt{2}+1}{\sqrt{2}-1}$ one can see that equations (4.17), (4.19), and (4.21) hold. The minimum c satisfying (4.16) is $c \approx 16.22$ and also satisfies (4.15). Then given that we overestimate the adversary's cost by a most a factor of $\sqrt{3}$, this gives us a competitive ratio for lines of approximately 28.1, slightly improving upon the competitive ratio obtained in [FL93].

If B_i is a line segment, then we need to account for the additional movement along E_i to reach B_i. However, as we set $\delta > 1$, the decrease in the potential can pay for this movement cost. □

4.5 Convex-Function Chasing

In this section, we investigate convex-function chasing and show a new lower bound on the achievable ratio for one-dimensional convex-function chasing.

Theorem 4.5. *There is no randomized c-competitive algorithm for convex-function chasing in one dimension when $c < 1.86$.*

As shown in [BGK+15], algorithms for function chasing in one dimension can be derandomized without increasing the competitive ratio, allowing us to restrict to deterministic algorithm in the proof of Theorem 4.5. The proof idea is similar to that in the proof of Lemma 4.3.

Lemma 4.3 (Bansal et al. [BGK+15])**.** *If there exists a randomized c-competitive algorithm for convex-function chasing in one dimension, there also exists a deterministic such algorithm.*

We now give the proof of the theorem.

Proof of Theorem 4.5. Due to Lemma 4.3, we can restrict to deterministic algorithms without loss of generality. Let A be an arbitrary c-competitive deterministic algorithm.

We now define our adversarial strategy. The initial position is 0. Then some number of functions of the form $\varepsilon|1-x|$ arrive. We will be interested in the limit as ε approaches 0. Then some number, possibly none, of functions of the form $\varepsilon|x|$ arrive.

For the deterministic algorithm A, let $b(s)$ denote the position of A after s/ε functions of the type $\varepsilon|1-x|$ have arrived. Intuitively, if $b(s)$ is too small for large enough s, then it has a high function cost on the first s/ε functions whereas the optimal solution would have moved immediately to the point 1 only incurring the moving cost. Alternately, if the position $b(s)$ is sufficiently far to the right (i.e., close to 1), then the adversary can introduce a very long sequence of requests of type $\varepsilon|x|$, forcing the algorithm to eventually move back to 0 incurring the movement cost of $b(s)$ again. In this case, the optimal solution would have stayed at 0.

Formally, the total function cost of A at time s/ε is at least $b(s) + \int_0^s (1 - b(y))\, dy$. Now, if the adversary introduces an infinite sequence of functions of the form $\varepsilon|x|$, then the best that the online algorithm can do is to move immediately to the origin incurring an additional movement cost of $b(s)$. Meanwhile, the optimal solution would have stayed at 0 throughout incurring a total cost of s. Hence, if the online algorithm is c-competitive, we must have, for all s,

$$2b(s) + \int_0^s (1 - b(y))\, dy \leq cs. \tag{4.22}$$

Alternately, if the functions $\varepsilon|1 - x|$ keep appearing forever, the online algorithm eventually moves to 1 and its total cost is therefore at least $1 + \int_0^\infty (1 - b(y))\, dy$ and the optimal solution would have moved to 1 at time 0 and only incurred the movement cost of 1. Hence, we also have

$$1 + \int_0^\infty (1 - b(y))\, dy \leq c. \tag{4.23}$$

This establishes the dichotomy using which we complete our lower bound proof. Indeed, define $G(s) = \int_0^s (1 - b(y))\, dy$. Then, $G'(s) = 1 - b(s)$ and we can write (4.22) as, for all s we have

$$G'(s) \geq \frac{1}{2}\left(2 - cs + G(s)\right) \tag{4.24}$$

and (4.23) is simply

$$G(\infty) \leq c - 1\,.$$

Now, notice that in order to minimize $G(\infty)$, we may assume that (4.24) is satisfied with equality for all s (this can only reduce $G(s)$, which in turn reduces $G'(s)$ further), which in turn gives us a unique solution to G.

Now, writing (4.24) as equality and differentiating w.r.t s, we get the first-order differential equation $b(s) = 2b'(s) - c + 1$. It is a simple calculation to verify that its unique solution satisfying $b(0) = 0$ is $b(s) = (c - 1) \cdot (e^{s/2} - 1)$. But now, we can plug this into $G(\infty)$ to get that

$$\int_0^{2\ln\frac{c}{c-1}} (1 - b(s))\, ds + 1 = \int_0^{2\ln\frac{c}{c-1}} \left(1 - (c-1)\left(e^{s/2} - 1\right)\right) ds + 1 \leq c.$$

Evaluation of the integral and simplification yields

$$2\ln\frac{c}{c-1} - (c-1)\left(\frac{2c}{c-1} - 2\ln\frac{c}{c-1} - 2\right) + 1 = 2c\ln\frac{c}{c-1} - 1 \leq c,$$

which is false for $c < 1.86$. □

We strongly conjecture that this lower bound is not tight. A promising idea to prove a stronger lower bound would be considering instances that do not alternate only once between functions of the two types but an arbitrary number of times.

4.6 Open Problems

Whereas we have presented a new $\mathcal{O}(1)$-competitive algorithm for hyperplane chasing in an arbitrary fixed dimensions d, the asymptotical behavior of this competitive ratio is not settled. It is an interesting open problem for future research to close the gap between $2^{\mathcal{O}(d)}$ and $\sqrt{d}$.

For convex-function chasing, so far $\mathcal{O}(1)$-competitive algorithms are only known for one dimension. It is an interesting open question whether such an algorithm also exists for higher dimensions. Also, for one-dimensional function chasing, there still remains a gap between 1.86 and 2 for the optimal competitive ratio to be in.

Bibliography

[ABL+13] Lachlan L. H. Andrew, Siddharth Barman, Katrina Ligett, Minghong Lin, Adam Meyerson, Alan Roytman, and Adam Wierman. A tale of two metrics: Simultaneous bounds on competitiveness and regret. In *Conference on Learning Theory (COLT)*, pages 741–763, 2013.

[ABL+15] Lachlan L. H. Andrew, Siddharth Barman, Katrina Ligett, Minghong Lin, Adam Meyerson, Alan Roytman, and Adam Wierman. A tale of two metrics: Simultaneous bounds on competitiveness and regret. *CoRR*, abs/1508.03769, 2015.

[ABN+16] Antonios Antoniadis, Neal Barcelo, Michael Nugent, Kirk Pruhs, Kevin Schewior, and Michele Scquizzato. Chasing convex bodies and functions. In *Latin American Theoretical Informatics Symposium (LATIN)*, pages 68–81, 2016.

[AGM11] S. Anand, Naveen Garg, and Nicole Megow. Meeting deadlines: How much speed suffices? In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 232–243, 2011.

[AGM15] S. Anand, Naveen Garg, and Nicole Megow. Erratum for "Meeting deadlines: How much speed suffices?". Available at `https://www-m9.ma.tum.de/foswiki/pub/Allgemeines/NicoleMegowPapers/erratum.pdf`, 2015.

[BBK+94] Shai Ben-David, Allan Borodin, Richard M. Karp, Gábor Tardos, and Avi Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11(1):2–14, 1994.

[BBM06] Yair Bartal, Béla Bollobás, and Manor Mendel. Ramsey-type theorems for metric spaces with applications to online problems. *J. Comput. Syst. Sci.*, 72(5):890–921, 2006.

[BGK+15] Nikhil Bansal, Anupam Gupta, Ravishankar Krishnaswamy, Kirk Pruhs, Kevin Schewior, and Clifford Stein. A 2-competitive algorithm for online convex optimization with switching costs. In *Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 96–109, 2015.

[BKP07] Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Speed scaling to manage energy and temperature. *J. ACM*, 54(1), 2007.

[BLMN03] Yair Bartal, Nathan Linial, Manor Mendel, and Assaf Naor. On metric Ramsey-type phenomena. In *ACM Symposium on Theory of Computing (STOC)*, pages 463–472, 2003.

[BLS92] Allan Borodin, Nathan Linial, and Michael E. Saks. An optimal on-line algorithm for metrical task system. *J. ACM*, 39(4):745–763, 1992.

[CEH+04] Mark Cieliebak, Thomas Erlebach, Fabian Hennecke, Birgitta Weber, and Peter Widmayer. Scheduling with release times and deadlines on a minimum number of machines. In *International Conference on Theoretical Computer Science (TCS)*, pages 217–230, 2004.

[CGKN04] Julia Chuzhoy, Sudipto Guha, Sanjeev Khanna, and Joseph Naor. Machine minimization for scheduling jobs with interval constraints. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 81–90, 2004.

[CKM06] M. Chrobak and C. Kenyon-Mathieu. SIGACT news online algorithms Column 10: Competitiveness via doubling. *SIGACT News*, 37(4):115–126, 2006.

[CLT05] Ho-Leung Chan, Tak Wah Lam, and Kar-Keung To. Nonmigratory online deadline scheduling on multiprocessors. *SIAM J. Comput.*, 34(3):669–682, 2005.

[CMS16a] Lin Chen, Nicole Megow, and Kevin Schewior. An $\mathcal{O}(\log m)$-competitive algorithm for online machine minimization. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 155–163, 2016.

[CMS16b] Lin Chen, Nicole Megow, and Kevin Schewior. The power of migration in online machine minimization. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2016. To appear.

[DM89] Michael L. Dertouzos and Aloysius K. Mok. Multiprocessor on-line scheduling of hard-real-time tasks. *IEEE Trans. on Software Eng.*, 15(12):1497–1506, 1989.

[DMPY14] Nikhil R. Devanur, Konstantin Makarychev, Debmalya Panigrahi, and Grigory Yaroslavtsev. Online algorithms for machine minimization. *CoRR*, abs/1403.0486, 2014.

[FF56] Lester R. Ford and Delbert R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8(3):399–404, 1956.

[FL93] Joel Friedman and Nathan Linial. On convex body chasing. *Discrete & Computational Geometry*, 9:293–321, 1993.

[FM03] Amos Fiat and Manor Mendel. Better algorithms for unfair metrical task systems and applications. *SIAM J. Comput.*, 32(6):1403–1422, 2003.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, New York, 1979.

[Gra69] Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal of Applied Mathematics*, 17(2):416–429, 1969.

[Han57] James Hannan. Approximation to bayes risk in repeated play. *Contributions to the Theory of Games*, 3(2):97–139, 1957.

[Hor74] W. A. Horn. Some simple scheduling algorithms. *Naval Research Logistics Quarterly*, 21(1):177–185, 1974.

[KCRW12] Mong-Jen Kao, Jian-Jia Chen, Ignaz Rutter, and Dorothea Wagner. Competitive design and analysis for machine-minimizing job scheduling problem. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 75–84, 2012.

[KMRS88] Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel Dominic Sleator. Competitive snoopy caching. *Algorithmica*, 3:77–119, 1988.

[KNST99] Anton J. Kleywegt, Vijay S. Nori, Martin W. P. Savelsbergh, and Craig A. Tovey. Online resource minimization. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 576–585, 1999.

[KP98] Bala Kalyanasundaram and Kirk Pruhs. Maximizing job completions online. In *Algorithms - ESA*, pages 235–246, 1998.

[KP00] Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, 2000.

[KP01] Bala Kalyanasundaram and Kirk Pruhs. Eliminating migration in multi-processor scheduling. *J. Algorithms*, 38(1):2–24, 2001.

[KT81] Hal A. Kierstead and William T. Trotter. An extremal problem in recursive combinatorics. *Congressus Numerantium*, 33:143–153, 1981.

[LT99] Tak Wah Lam and Kar-Keung To. Trade-offs between speed and processor in hard-deadline scheduling. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 623–632, 1999.

[LWAT13] Minghong Lin, Adam Wierman, Lachlan L. H. Andrew, and Eno Thereska. Dynamic right-sizing for power-proportional data centers. *IEEE/ACM Trans. Netw.*, 21(5):1378–1391, 2013.

[MO79] Albert W. Marshall and Ingram Olkin. *Inequalities: Theory of Majorization and Its Application*. Academic Press, New York, 1979.

[Pru10] Kirk Pruhs. In *Collection of Open Problems in Scheduling*, Dagstuhl Scheduling Seminar, 2010.

[PSTW02] Cynthia A. Phillips, Clifford Stein, Eric Torng, and Joel Wein. Optimal time-critical scheduling via resource augmentation. *Algorithmica*, 32(2):163–200, 2002.

[Sah13] Barna Saha. Renting a cloud. In *IARCS Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 437–448, 2013.

[ST85] Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.

[Wie15] Adam Wierman. Personal communication, 2015.

[YZ09] Guosong Yu and Guochuan Zhang. Scheduling with a minimum number of machines. *Oper. Res. Lett.*, 37(2):97–101, 2009.