A New Approach to Competitive Analysis: Approximating the Optimal Competitive Ratio

Elisabeth Günther* Olaf Maurer* Nicole Megow† Andreas Wiese‡

Abstract

We propose a new approach to competitive analysis by introducing the novel concept of online approximation schemes. Such scheme algorithmically constructs an online algorithm with a competitive ratio arbitrarily close to the best possible competitive ratio for *any* online algorithm. We study the problem of scheduling jobs online to minimize the weighted sum of completion times on parallel, related, and unrelated machines, and we derive both deterministic and randomized algorithms which are almost best possible among all online algorithms of the respective settings. Our method relies on an abstract characterization of online algorithms combined with various simplifications and transformations. We also contribute algorithmic means to compute the actual value of the best possible competitive ratio up to an arbitrary accuracy. This strongly contrasts all previous manually obtained competitiveness results for algorithms and, most importantly, it reduces the search for the optimal competitive ratio to a question that a computer can answer. We believe that our method can also be applied to many other problems and yields a completely new and interesting view on online algorithms.

*Department of Mathematics, Technische Universität Berlin, Germany. Email: {eguenth,maurer}@math.tu-berlin.de
Supported by the DFG Research center MATHEON *Mathematics for key technologies* in Berlin.

†Max-Planck-Institut für Informatik, Saarbrücken, Germany. Email: nmegow@mpi-inf.mpg.de

‡Department of Computer and System Sciences, Sapienza University of Rome, Italy. Email: wiese@dis.uniroma1.it
Supported by the DFG Focus Program 1307.

1 Introduction

Competitive analysis [13,27] is the most popular method for studying the performance of online algorithms. It provides an effective framework to analyze and classify algorithms based on their worst-case behavior compared to an optimal offline algorithm over an infinite set of input instances. For some problem types, e.g., online paging, competitive analysis may not be adequate to evaluate the performance of algorithms, but for a vast majority of online problems it is practical, natural, and yields meaningful results. A classical such problem is online scheduling to minimize the weighted average completion time. It has received lots of attention in the past two decades. For different machine environments, a long sequence of papers emerged introducing new techniques and algorithms, improving upper and lower bounds on the competitive ratio of particular algorithms as well as on the best possible competitive ratio that any online algorithm can achieve. Still, unsatisfactory gaps remain. As for most online problems, a provably optimal online algorithm, w.r.t. to competitive analysis, among *all* online algorithms is only known for very special cases.

In this work we close these gaps and present nearly optimal online scheduling algorithms. We provide *online approximation schemes* that compute algorithms with a competitive ratio that is at most a factor $1+\varepsilon$ larger than the optimal ratio for any $\varepsilon > 0$. To that end, we introduce a new way of designing online algorithms. Apart from structuring and simplifying input instances, we find an abstract description of online algorithms, which allows us to reduce the infinite-size set of online algorithms to a relevant set of finite size. This is the key for eventually allowing an enumeration scheme that finds an online algorithm with a competitive ratio arbitrarily close to the optimal one. Besides improving on previous algorithms, our method also provides an algorithm to compute the competitive ratio of the designed algorithm, and even the best possible competitive ratio, up to any desired accuracy. To the best of our knowledge, this is the first time that a competitive ratio, or even the optimal competitive ratio, are shown to be *computable* by some algorithm. And it is clearly in strong contrast to all previously given (lower) bounds that stem from *manually* designed input instances. Our result is surprising, as there are typically no means of enumerating all possible input instances and all possible online algorithms. Even for only one given algorithm, usually one cannot compute its competitive ratio, already due to difficulties like the halting problem. We show how to overcome these obstacles and open the doors to a computer-assisted design of online algorithms.

We believe that our method of abstraction for online algorithms can be applied successfully to other problems. Furthermore, we hope that this new approach to competitive analysis contributes to a better understanding of online algorithms and may lead to a new line of research in online optimization.

1.1 Problem Definition and Previous Results

Competitive analysis. Given a minimization problem, a deterministic online algorithm A is called ρ-*competitive* if, for any problem instance $\mathcal{I}$, it achieves a solution of value $\mathsf{A}(\mathcal{I}) \le \rho \cdot \mathsf{OPT}(\mathcal{I})$, where $\mathsf{OPT}(\mathcal{I})$ denotes the value of an optimal offline solution for the same instance $\mathcal{I}$. A randomized online algorithm is called ρ-competitive, if it achieves in expectation a solution of value $\mathbb{E}[\,\mathsf{A}(\mathcal{I})\,] \le \rho \cdot \mathsf{OPT}(\mathcal{I})$ for any instance $\mathcal{I}$. The *competitive ratio* ρ_{A} of A is the infimum over all ρ such that A is ρ-competitive. The minimum competitive ratio ρ^* achievable by any online algorithm is called *optimal*. Note that there are no requirements on the computational complexity of competitive algorithms. Indeed, the competitive ratio measures the best possible performance under the lack of information given unbounded computational resources.

We define an online approximation scheme as a procedure that computes a nearly optimal online algorithm and at the same time provides a nearly exact estimate of the optimal competitive ratio.

Definition 1.1. An *online approximation scheme* computes for a given $\varepsilon > 0$ an online algorithm A with a competitive ratio $\rho_{\mathsf{A}} \le (1+\varepsilon)\rho^*$. Moreover, it determines a value ρ' such that $\rho' \le \rho^* \le (1+\varepsilon)\rho'$.

Online scheduling. A scheduling instance consists of a fixed set of m machines and a set of jobs J, where

problem	deterministic lower bounds		deterministic upper bounds		randomized lower bounds		randomized upper bounds	
$1\|r_j, pmtn\|\sum C_j$	1		1	[21]	1		1	[21]
$1\|r_j, pmtn\|\sum w_jC_j$	1.073	[7]	1.566	[25]	1.038	[7]	4/3	[22]
$1\|r_j\|\sum C_j$	2	[12]	2	[12]	$\frac{e}{e-1} \approx 1.58$	[29]	$\frac{e}{e-1}$	[4]
$1\|r_j\|\sum w_jC_j$	2	[12]	2	[2]	$\frac{e}{e-1}$	[29]	1.686	[9]
$P\|r_j, pmtn\|\sum C_j$	1.047	[12]	5/4	[26]	1		5/4	[26]
$P\|r_j, pmtn\|\sum w_jC_j$	1.047	[30]	1.791	[26]	1		1.791	[26]
$P\|r_j\|\sum w_jC_j$	1.309	[30]	1.791	[26]	1.157	[24]	1.791	[26]
$R\|r_j\|\sum w_jC_j$	1.309	[30]	8	[11]	1.157	[24]	8	[11]

Table 1: Lower and upper bounds on the competitive ratio for deterministic and randomized online algorithms.

each job $j \in J$ has processing time $p_j \in \mathbb{Q}^+$, weight $w_j \in \mathbb{Q}^+$, and release date $r_j \in \mathbb{Q}^+$. The jobs arrive online over time, that is, each job becomes known to the scheduling algorithm only at its release date. We consider three different machine environments: identical parallel machines (denoted by P), related machines (Q) where each machine i has associated a speed s_i and processing a job j on machine i takes p_j/s_i time, and unrelated machines (R) where the processing time of a job j on each machine i is explicitly given as a value p_{ij}. The objective is to schedule the jobs on the given set of machines so as to minimize $\sum_{j\in J} w_jC_j$, where C_j denotes the completion time of job j. We consider the problem with and without preemption. Using the standard scheduling notation [10], we denote the non-preemptive (preemptive) problems that we consider in this paper by $Pm|r_j, (pmtn)|\sum w_jC_j$, $Qm|r_j, (pmtn)|\sum w_jC_j$, and $Rm|r_j, pmtn|\sum w_jC_j$.

Previous results. The offline variants of all these problems are NP-hard. This is true already for the special case of a single machine [14, 15]. However, polynomial-time approximation schemes have been developed [1], even when the number of machines is part of the input.

The online setting has been a highly active field of research in the past fifteen years. A whole sequence of papers appeared introducing new algorithms, new relaxations and analytical techniques that decreased the gaps between lower and upper bounds on the optimal competitive ratio [2–9, 11, 12, 16–20, 22–26, 29]. We do not intend to give a detailed history of developments; instead, we refer the reader to overviews, e.g., in [6, 18]. Table 1 summarizes the current state-of-the-art. Interestingly, despite the considerable effort, optimal competitive ratios are known only for $1|r_j, pmtn|\sum C_j$ [21] and for non-preemptive single-machine scheduling [2, 4, 12, 29]. In all other scheduling settings remain unsatisfactory, even quite significant gaps.

1.2 New Results and Methodology

In this paper, we introduce the concept of online approximation schemes and present such schemes for the scheduling problems $Pm|r_j, (pmtn)|\sum w_jC_j$, $Qm|r_j, (pmtn)|\sum w_jC_j$, and $Rm|r_j, pmtn|\sum w_jC_j$ (assuming a constant range of machines speeds in the case of $Qm|r_j|\sum w_jC_j$). For any $\varepsilon > 0$, we show that the competitive ratios of our new algorithms are by at most a factor $1 + \varepsilon$ larger than the respective optimal competitive ratios. We obtain such nearly optimal online algorithms for the deterministic as well as for the randomized setting. Moreover, we give an algorithm which estimates the optimal competitive ratio for these problems to any desired accuracy. These results reduce the long-time ongoing search for the best possible competitive ratio for the considered problems to a question that can be answered by a finite algorithm.

To achieve our results, we introduce a completely new way of designing online algorithms. We present a novel abstraction in which online algorithms are formalized as *algorithm maps*. Such a map receives as input a set of unfinished jobs together with the schedule computed so far. Based on this information, it returns a schedule for the next time instant. This view captures exactly how online algorithms operate under limited information. The total number of algorithm maps is unbounded. However, we show that there is a

finite subset which approximates the entire set. More precisely, for any algorithm map there is a map in our subset whose competitive ratio is at most by a factor $1 + \varepsilon$ larger. To achieve this reduction, we first apply several standard techniques, such as geometric rounding, time-stretch, and weight-shift, to transform and simplify the input problem without increasing the objective value too much; see, e.g., [1]. The key, however, is the insight that it suffices for an online algorithm to base its decisions on the currently unfinished jobs and a very *limited part* of the so far computed schedule—rather than the entire history. This allows for an enumeration of all relevant algorithm maps. For randomized algorithms we even show that we can restrict to instances with only constantly many jobs. As all our structural insights also apply to offline algorithms for the same problems and even more general variants, they might turn out to be useful for other settings as well.

Our algorithmic scheme contributes more than an improved competitive ratio. It also outputs (up to a factor $1 + \varepsilon$) the exact value of the competitive ratio of the derived algorithm, which implies a $(1 + \varepsilon)$-estimate for the optimal competitive ratio. This contrasts strongly all earlier results, where (matching) upper and lower bounds on the competitive ratio of a particular and of all online algorithm had to be derived *manually*, instead of executing an algorithm using, e.g., a computer. In general, there are no computational means to determine the competitive ratio of an algorithm. It is simply not possible to enumerate all possible input instances. Even more, there are no general means of enumerating all possible online algorithms to determine the optimal competitive ratio. However, for the scheduling problems studied in this paper our extensive simplification of input instances and our abstract view on online algorithms allow us to overcome these obstacles, losing only a factor of $1 + \varepsilon$ in the objective.

Although the enumeration scheme heavily exploits unbounded computational resources, the resulting algorithm itself has polynomial running time. As a consequence, there are efficient online algorithms for the considered problems with almost optimal competitive ratios. Hence, the granted additional, even unbounded, computational power of online algorithms does not yield any significant benefit here.

The techniques derived in this paper provide a completely new and interesting view on the behavior of online algorithms. We believe that they contribute significantly to the understanding of such algorithms and possibly open a new line of research in which they yield even further insights. In particular, it seems promising that our methods could also be applied to other online problems.

Outline of the paper. In Section 2 we introduce several general transformations and observations that simplify the structural complexity of online scheduling in the setting of $\mathrm{Pm}|\, r_j, pmtn \,|\, \sum w_j C_j$. Based on this, we present our abstraction of online algorithms and develop an online approximation scheme in Section 3. Next, we sketch in Section 4 how to extend the techniques of Section 2 for the non-preemptive setting and more general machine environments such that the online approximation scheme of Section 3 remains applicable. Finally, we present online approximation schemes for the randomized setting in Section 5. Due to space constraints, most proofs can be found in the appendix.

2 General Simplifications and Techniques

In this section, we discuss several transformations that simplify the input and reduce the structural complexity of online schedules for $\mathrm{Pm}|\, r_j, pmtn \,|\, \sum w_j C_j$. Later, we outline how to adapt these for more general settings. Our construction combines several transformation techniques known for offline PTASs (see [1] and the references therein) and a new technique to subdivide an instance online into parts which can be handled separately. We will use the terminology that at $1 + O(\varepsilon)$ loss we can restrict to instances or schedules with certain properties. This means that we lose at most a factor $1 + O(\varepsilon)$, as $\varepsilon \to 0$, by limiting our attention to those. We bound several relevant parameters by constants. If not stated differently, any mentioned constant depends only on ε and m.

Lemma 2.1 ([1]). *At $1 + O(\varepsilon)$ loss we can restrict to instances where all processing times, release dates, and weights are powers of $1 + \varepsilon$, no job is released before time $t = 1$, and $r_j \geq \varepsilon \cdot p_j$ for all jobs j.*

This standard *geometric rounding* procedure used in the lemma above allows us to see intervals of the form $I_x := [R_x, R_{x+1})$ with $R_x := (1+\varepsilon)^x$ as atomic entities for which a schedule can be computed already at the beginning of the interval. This is possible since no further job is released before the beginning of the next interval. Moreover, we assume at $1+\varepsilon$ loss that all jobs that finish within I_x have completion time R_{x+1}.

Simplification within intervals. Our goal is to reduce the number of situations that can arise at the beginning of an interval. To this end, we partition the set of jobs released at time R_x into the set of *large* jobs L_x, with processing times at least $\varepsilon^3 R_x$, and the set of *small* jobs S_x with all remaining jobs. Running *Smith's Rule* [28] on small jobs allows us to group very small jobs to job *packs*, which we treat as single jobs. Together with Lemma 2.1 we obtain bounds on the lengths of jobs of each release date.

Lemma 2.2. *At $1+O(\varepsilon)$ loss, we can assume that for each interval I_x there are lower and upper bounds for the lengths of the jobs $S_x \cup L_x$ which are within a constant factor of R_x and the constants are independent of x. Also, the number of distinct processing times of jobs in each interval is upper-bounded by a constant.*

We look for jobs in S_x and L_x which can be excluded from processing within I_x at a loss of not more than $1+O(\varepsilon)$. This allows to bound the number of released jobs per interval.

Lemma 2.3. *At $1+O(\varepsilon)$ loss, we can restrict to instances where for each x, the number of jobs released at time R_x is bounded by a constant Δ.*

To prove the above claims, we use the technique of *time-stretching*, see [1]. In an online interpretation of this method, we shift the work assigned to any interval I_x to the interval I_{x+1}. This can be done at a loss of $1+\varepsilon$ and we obtain free space of size $\varepsilon \cdot I_{x'-1}$ in each interval $I_{x'}$. Again using time-stretching, we can show that no job needs to completed later than constantly many intervals after its release interval.

Lemma 2.4. *There is a constant s such that at $1+O(\varepsilon)$ loss we can restrict to schedules such that for each interval I_x there is a subinterval of I_{x+s-1} as large as the total volume of the jobs released at R_x during which only jobs in R_x are executed. We call this subinterval the* safety net *of interval I_x. We can assume that each job released at R_x finishes before time R_{x+s}.*

We can also simplify the complexity of the computed schedules by limiting the way jobs are preempted. We say that two large jobs are of the same *type* if they have the same processing time and the same release date. A job is *partially processed* if it has been processed, but not yet completed.

Lemma 2.5. *There is a constant $\mu > 0$ such that at $1+O(\varepsilon)$ loss we can restrict to schedules such that*

- *at the end of each interval, there are at most m large jobs of each type which are partially processed and each of them is processed to an extent which is a multiple of $p_j \cdot \mu$ and*
- *each small job finishes without preemption in the same interval where it started.*

Irrelevant history. The schedule that an online algorithm computes for an interval may depend on the set of currently unfinished jobs and possibly the entire schedule used so far. In the remainder of this section we show why we can assume that an online algorithm only takes a finite amount of history into account in its decision making, namely, the jobs with relatively large weight released in the last constantly many intervals.

Our strategy is to partition the time horizon into *periods*. For each integer $k \geq 0$, we define a period Q_k which consists of the s consecutive intervals $I_{k\cdot s}, ..., I_{(k+1)\cdot s-1}$. For ease of notation, we will treat a period Q as the set of jobs released in that period. For a set of jobs J we denote by $rw(J) := \sum_{j\in J} r_j w_j$ their *release weight*. Note that $rw(J)$ forms a lower bound on the quantity that these jobs must contribute to the objective in *any* schedule. Due to Lemma 2.4, we also obtain an upper bound of $(1+\varepsilon)^s \cdot rw(J)$ for the latter quantity.

The following lemma identifies when a period Q_{k+p} is in comparison with the preceding p periods *insignificant* enough to schedule all jobs within their safety net. In this case, the jobs of Q_{k+p} have no influence on the decisions for the further instance. Hence, we can reset completely and forget the past.

Lemma 2.6. *Let $Q_k, ..., Q_{k+p}$ be consecutive periods such that period Q_{k+p} is the first of this series with $rw(Q_{k+p}) \leq \frac{\varepsilon}{(1+\varepsilon)^s} \cdot \sum_{i=0}^{p-1} rw(Q_{k+i})$. Then at $1+\varepsilon$ loss we can move all jobs in Q_{k+p} to their safety nets.*

The above observation defines a natural partition of a given instance $\mathcal{I}$ into *parts* by the insignificant periods. Formally, let $a_1, ..., a_\ell$ be all ordered indices such that Q_{a_i} is insignificant compared to the preceding periods according to Lemma 2.6 ($a_0 := 0$). Let $a_{\ell+1}$ be the index of the last period. For each $i \in \{0, ..., \ell\}$ we define a part P_i consisting of all periods $Q_{a_i+1}, ..., Q_{a_{i+1}}$. Again, identify with P_i all jobs released in this part. We treat now each part P_i as a separate instance that we present to a given online algorithm. For the final output, we concatenate the computed schedules for the different parts. It then suffices to bound $\mathsf{A}(P_i)/\mathsf{OPT}(P_i)$ for each part P_i since $\mathsf{A}(\mathcal{I})/\mathsf{OPT}(\mathcal{I}) \leq \max_i\{\mathsf{A}(P_i)/\mathsf{OPT}(P_i)\} \cdot (1 + O(\varepsilon))$.

Lemma 2.7. *At $1 + O(\varepsilon)$ loss we can restrict to instances which consist of only one part.*

Each but the last period of one part fulfills the opposite condition of the one from Lemma 2.6. This implies exponential growth for the series of partial sums of release weights (albeit with a small growth factor). From this observation, we get:

Lemma 2.8. *There is a constant K such that the following holds: Let $Q_1, Q_2, ..., Q_p$ be consecutive periods such that $rw(Q_{i+1}) > \frac{\varepsilon}{(1+\varepsilon)^s} \cdot \sum_{\ell=1}^{i} rw(Q_\ell)$ for all i. Then in any schedule in which each job j finishes at time $r_j \cdot (1+\varepsilon)^s$ the latest (e.g., using the safety net) it holds that $\sum_{i=1}^{p-K-1} \sum_{j \in Q_i} w_j C_j \leq \varepsilon \cdot \sum_{i=p-K}^{p} \sum_{j \in Q_i} w_j C_j$.*

The objective value of one part is therefore dominated by the contribution of the last K periods of this part. We will need this later to show that at $1+\varepsilon$ loss we can assume that an online algorithm bases its decisions only on a constant amount of information. Denote the corresponding number of important intervals by $\Gamma := Ks$.

This enables us to partition the jobs into relevant and irrelevant jobs. Intuitively, a job is irrelevant if it is released very early (cf. Lemma 2.8) or its weight is very small in comparison to some other job. The subsequent lemma states that the irrelevant jobs can almost be ignored for the objective value of a schedule.

Definition 2.9. A job j is *irrelevant at time R_x* if it was irrelevant at time R_{x-1}, or $r_j < R_{x-\Gamma}$, or it is *dominated at time R_x*. This is the case if there is a job j', either released at time R_x or already relevant at time R_{x-1} with release date at least $R_{x-\Gamma}$, such that $w_j < \frac{\varepsilon}{\Delta \cdot \Gamma \cdot (1+\varepsilon)^{\Gamma+s}} w_{j'}$. Otherwise, a job released until R_x is *relevant at time R_x*. Denote the respective subsets of some job set J by $\mathrm{Rel}_x(J)$ and $\mathrm{Ir}_x(J)$.

Lemma 2.10. *Consider a schedule of one part in which each job j finishes at time $r_j \cdot (1+\varepsilon)^s$ the latest (e.g., using the safety net) and let x be an interval index in this part. Then $\sum_{j \in \mathrm{Ir}_x(J)} C_j w_j \leq O(\varepsilon) \cdot \sum_{j \in \mathrm{Rel}_x(J)} C_j w_j$.*

The above lemma implies that at $1 + O(\varepsilon)$ loss we can restrict to online algorithms which schedule the remaining part of a job in its safety net, once it has become irrelevant.

3 Abstraction of Online Algorithms

In this section we show how to construct an online approximation scheme based on the observations of Section 2. To do so, we restrict ourselves to such simplified instances and schedules. In Section 4, we will sketch adjustments of these simplifications that are necessary to apply our framework to other problems.

The key idea is to characterize the behavior of an online algorithm by a map: For each interval, the map gets as input the schedule computed so far and all information about the currently unfinished jobs. Based on this information, the map outputs how to schedule the available jobs within this interval.

More precisely, we define the input by a *configuration* and the output by an *interval-schedule*.

Definition 3.1. An *interval-schedule S* for an interval I_x is defined by

- the index x of the interval,
- a set of jobs $J(S)$ available for processing in I_x together with the properties r_j, p_j, w_j of each job $j \in J(S)$ and its already finished part $f_j < p_j$ up to R_x,
- for each job $j \in J(S)$ the information whether j is relevant at time R_x, and
- for each job $j \in J(S)$ and each machine i a value q_{ij} specifying for how long j is processed by S on machine i during I_x.

An interval-schedule is called *feasible* if there is a feasible schedule in which the jobs of $J(S)$ are processed corresponding to the q_j values within the interval I_x. Denote the set of feasible interval-schedules as $\mathcal{S}$.

Definition 3.2. A *configuration* C for an interval I_x consists of

- the index x of the interval,
- a set of jobs $J(C)$ released up to time R_x together with the properties r_j, p_j, w_j, f_j of each job $j \in J(C)$,
- an interval-schedule for each interval $I_{x'}$ with $x' < x$.

The set of all configurations is denoted by $\mathcal{C}$. A configuration C for an interval I_x such that at time R_x, and not earlier, no jobs are left unprocessed is an *end-configuration.*

We say that an interval-schedule S is *feasible for a configuration* C if the set of jobs in $J(C)$ which are unfinished at time R_x matches the set $J(S)$ with respect to release dates, total and remaining processing time, weight and relevance of the jobs.

Instead of online algorithms we work from now on with *algorithm maps*, which are defined as functions $f : \mathcal{C} \to \mathcal{S}$. An algorithm map determines a schedule $f(\mathcal{I})$ for a given scheduling instance $\mathcal{I}$ by iteratively applying f to the corresponding configurations. W.l.o.g. we consider only algorithm maps f such that $f(C)$ is feasible for each configuration C and $f(\mathcal{I})$ is feasible for each instance I. Like for online algorithms, we define the competitive ratio ρ_f of an algorithm map f by $\rho_f := \max_{\mathcal{I}} f(\mathcal{I})/\mathsf{OPT}(\mathcal{I})$. Due to the following observation, algorithm maps are a natural generalization of online algorithms.

Proposition 3.3. *For each online algorithm* A *there is an algorithm map* f_{A} *such that when* A *is in configuration* $C \in \mathcal{C}$ *at the beginning of an interval* I_x*, algorithm* A *schedules the jobs according to* $f_{\mathsf{A}}(C)$*.*

Recall, that we restrict our attention to algorithm maps describing online algorithms which obey the simplifications introduced in Section 2. The essence of such online algorithms are the decisions for the relevant jobs. To this end, we define equivalence classes for configurations and for interval-schedules. Intuitively, two interval-schedules (configurations) are equivalent if we can obtain one from the other by scalar multiplication with the same value, while ignoring the irrelevant jobs.

Definition 3.4. Let S, S' be two feasible interval-schedules for two intervals $I_x, I_{x'}$. Denote by $J_{\mathrm{Rel}}(S) \subseteq J(S)$ and $J_{\mathrm{Rel}}(S') \subseteq J(S')$ the relevant jobs in $J(S)$ and $J(S')$. Let further $\sigma : J_{\mathrm{Rel}}(S) \to J_{\mathrm{Rel}}(S')$ be a bijection and y an integer. The interval-schedules S, S' are (σ, y)-*equivalent* if $r_{\sigma(j)} = r_j(1+\varepsilon)^{x'-x}$, $p_{\sigma(j)} = p_j(1+\varepsilon)^{x'-x}$, $f_{\sigma(j)} = f_j(1+\varepsilon)^{x'-x}$, $q_{\sigma(j)} = q_j(1+\varepsilon)^{x'-x}$ and $w_{\sigma(j)} = w_j(1+\varepsilon)^{y}$ for all $j \in J_{\mathrm{Rel}}(S)$. The interval-schedules S, S' are *equivalent* (denoted by $S \sim S'$) if a map σ and an integer y exist such that they are (σ, y)-equivalent.

Definition 3.5. Let C, C' be two configurations for two intervals $I_x, I_{x'}$. Denote by $J_{\mathrm{Rel}}(C), J_{\mathrm{Rel}}(C')$ the jobs which are relevant at times $R_x, R_{x'}$ in C, C', respectively. Configurations C, C' are *equivalent* (denoted by $C \sim C'$) if there is a bijection $\sigma : J_{\mathrm{Rel}}(C) \to J_{\mathrm{Rel}}(C')$ and an integer y such that

- $r_{\sigma(j)} = r_j(1+\varepsilon)^{x'-x}$, $p_{\sigma(j)} = p_j(1+\varepsilon)^{x'-x}$, $f_{\sigma(j)} = f_j(1+\varepsilon)^{x'-x}$ and $w_{\sigma(j)} = w_j(1+\varepsilon)^{y}$ for all $j \in J_{\mathrm{Rel}}(C)$, and
- the interval-schedules of I_{x-k} and $I_{x'-k}$ are (σ, y)-equivalent when restricted to the jobs in $J_{\mathrm{Rel}}(C)$ and $J_{\mathrm{Rel}}(C')$ for each $k \in \mathbb{N}$.

A configuration C is *realistic* for an algorithm map f if there is an instance $\mathcal{I}$ such that if f processes $\mathcal{I}$ then at time R_x it is in configuration C. The following lemma shows that we can restrict the set of algorithm maps under consideration to those which treat equivalent configurations equivalently. We call algorithm maps obeying this condition in addition to the restrictions of Section 2 *simplified algorithm maps*.

Lemma 3.6. *At $1 + O(\varepsilon)$ loss we can restrict to algorithm maps f such that $f(C) \sim f(C')$ for any two equivalent configurations C, C'.*

Proof. Let f be an algorithm map. For each equivalence class $\mathcal{C}' \subseteq \mathcal{C}$ of the set of configurations we pick a representative C which is realistic for f. We define a new algorithm map $\bar{f}$ by defining $\bar{f}(C')$ to be the interval-schedule for C' which is equivalent to $f(C)$ for each configuration $C' \in \mathcal{C}'$. One can show by induction that $\bar{f}$ is always in a configuration such that an equivalent configuration is realistic for f. Hence, equivalence classes without realistic configurations for f are not relevant. We claim that $\rho_{\bar{f}} \leq (1 + O(\varepsilon))\rho_f$.

Consider an instance $\bar{\mathcal{I}}$. We show that there is an instance $\mathcal{I}$ such that $\bar{f}(\bar{\mathcal{I}})/\mathsf{OPT}(\bar{\mathcal{I}}) \leq (1 + O(\varepsilon))f(\mathcal{I})/\mathsf{OPT}(\mathcal{I})$ which implies the claim. Consider an interval $I_{\bar{x}}$. Let $\bar{C}$ be the end-configuration obtained when $\bar{f}$ is applied iteratively on $\bar{\mathcal{I}}$. Let C be the representative of the equivalence class of $\bar{C}$, which was chosen above and which is realistic for f (C is also an end-configuration). Therefore, there is an instance $\mathcal{I}$ such that C is reached at time R_x when f is applied on $\mathcal{I}$. Hence, I is the required instance since the relevant jobs dominate the objective value (see Lemma 2.10) and $C \sim \bar{C}$. □

Lemma 3.7. *There are only constantly many simplified algorithm maps. Each simplified algorithm map can be described using finite information.*

Proof. From the simplifications introduced in Section 2 follows that the domain of the algorithm maps under consideration contains only constantly many equivalence classes of configurations. Also, the target space contains only constantly many equivalence classes of interval-schedules. For an algorithm map f which obeys the restrictions of Section 2, the interval-schedule $f(C)$ is fully specified when knowing only C and the equivalence class which contains $f(C)$ (since the irrelevant jobs are moved to their safety net anyway). Since $f(C) \sim f(C')$ for a simplified algorithm map f if $C \sim C'$, we conclude that there are only constantly many simplified algorithm maps. Finally, each equivalence class of configurations and interval-schedules can be characterized using only finite information, and hence the same holds for each simplified algorithm map. □

The next lemma shows that up to a factor $1 + \varepsilon$ worst case instances of simplified algorithm maps span only constantly many intervals. Using this property, we will show in the subsequent lemmas that the competitive ratio of a simplified algorithm map can be determined algorithmically up to a $1 + \varepsilon$ factor.

Lemma 3.8. *There is a constant E such that for any instance I and any simplified algorithm map f there is a realistic end-configuration $\tilde{C}$ for an interval $I_{\tilde{x}}$ with $\tilde{x} \leq E$ which is equivalent to the corresponding end-configuration when f is applied to I.*

Proof. Consider a simplified algorithm map f. For each interval I_x, denote by $\mathcal{C}_x^f$ the set of realistic equivalence classes for I_x, i.e., the equivalence classes which have a realistic representative for I_x. Since there are constantly many equivalence classes and thus constantly many *sets* of equivalence classes, there must be a constant E independent of f such that $\mathcal{C}_{\bar{x}}^f = \mathcal{C}_{\bar{x}'}^f$ for some $\bar{x} < \bar{x}' \leq E$. Since f is simplified it can be shown by induction that $\mathcal{C}_{\bar{x}+k}^f = \mathcal{C}_{\bar{x}'+k}^f$ for any $k \in \mathbb{N}$, i.e., f *cycles* with period length $\bar{x}' - \bar{x}$.

Consider now some instance I and let C with interval I_x be the corresponding end-configuration when f is applied to I. If $x \leq E$ we are done. Otherwise there must be some $k \leq \bar{x}' - \bar{x}$ such that $\mathcal{C}_{\bar{x}+k}^f = \mathcal{C}_x^f$ since f cycles with this period length. Hence, by definition of $\mathcal{C}_{\bar{x}+k}^f$ there must be a realistic end-configuration $\tilde{C}$ which is equivalent to C for the interval $I_{\tilde{x}}$ with $\tilde{x} := \bar{x} + k \leq E$. □

Lemma 3.9. *Let f be a simplified algorithm map. There is an algorithm which approximates ρ_f up to a factor $1+\varepsilon$, i.e., it computes a value ρ' with $\rho' \leq \rho_f \leq (1+O(\varepsilon))\rho'$.*

Proof sketch. By Lemma 2.10, the relevant jobs in a configuration dominate the entire objective value. In particular, we do not need to know the irrelevant jobs of a configuration if we only want to approximate its objective value up to a factor of $1+O(\varepsilon)$. For an end-configuration C denote by $val_C(J_{\text{Rel}}(C))$ the objective value of the jobs in $J_{\text{Rel}}(C)$ in the history of C. We define $r(C) := val_C(J_{\text{Rel}}(C))/\mathsf{OPT}(J_{\text{Rel}}(C))$ to be the achieved competitive ratio of C when restricted to the relevant jobs. According to Lemma 3.8, it suffices to construct the sets $C_0^f, ..., C_E^f$ in order to approximate the competitive ratio of all end-configurations in these sets. We start with C_0^f and determine $f(C)$ for one representant C of each equivalence class $C \in C_0^f$. Based on this we determine the set C_1^f. We continue inductively to construct all sets C_x^f with $x \leq E$.

We define $r_{\max}$ to be the maximum ratio $r(C)$ for an end-configuration $C \in \cup_{0\leq x\leq E} C_x^f$. Due to Lemma 3.8 and Lemma 2.10 the value $r_{\max}$ implies the required ρ' fulfilling the properties claimed in this lemma. □

Our main algorithm works as follows. We first enumerate all simplified algorithm maps. For each simplified algorithm map f we approximate ρ_f using Lemma 3.9. We output the map f with the minimum (approximated) competitive ratio. Note that the resulting online algorithm has polynomial running time: All simplifications of a given instance can be done efficiently and for a given configuration, the equivalence class of the schedule for the next interval can be found in a look-up table of constant size.

Theorem 3.10. *We obtain an online approximation scheme for* $\text{Pm}|\, r_j, pmtn \,|\sum w_j C_j$.

4 Extensions to Other Settings

4.1 Non-preemptive Scheduling

When preemption is not allowed, the definition of the safety net (Lemma 2.4) needs to be adjusted since we cannot ensure that at the end of each interval I_{x+s} there is a machine idle. However, we can guarantee that there is a reserved space *somewhere* in $[R_x, R_{x+s})$ to process the small and big jobs in $S_x \cup L_x$. Furthermore, we cannot enforce that a big job j is processed for exactly a certain multiple of $p_j\mu$ in each interval (Lemma 2.5). To solve this, we pretend that we could preempt j and ensure that after j has been preempted its machine stays until j continues. Next, we can no longer assume that each part can be treated independently (Lemma 2.7). Since some of the remaining jobs at the end of a part may have already started processing, we cannot simply move them to their safety net. Here we use the following simplification.

Lemma 4.1. *Let* $\text{first}(i)$ *denote the job that is released first in part P_i. At $1+\varepsilon$ loss, we can restrict to instances such that $\sum_{\ell=1}^{i-1} rw(P_\ell) \leq \frac{\varepsilon}{(1+\varepsilon)^s} \cdot rw(\text{first}(i))$, i.e.,* $\text{first}(i)$ *dominates all previous parts.*

Therefore, at $1+\varepsilon$ loss it is enough to consider only the currently running jobs from the previous part and the last Γ intervals from the current part when taking decisions. Finally, we add some minor modifications to handle the case that a currently running job is dominated by some other job. With these adjustments, we have only constantly many equivalence classes for interval-schedules and configurations, which allows us to construct an online approximation scheme as in Section 3.

Theorem 4.2. *We obtain an online approximation scheme for* $\text{Pm}|\, r_j \,|\sum w_j C_j$.

4.2 Scheduling on Related Machines

In the related machine setting, each machine i has associated a speed s_i, such that processing job j on machine i takes p_j/s_i time units. W.l.o.g. the slowest machine has unit speed. Let $s_{\max}$ denote the maximum

speed in an instance. An adjusted version of Lemma 2.1 can ensure that at $1+\varepsilon$ loss $r_j \geq \varepsilon\, p_j / s_{\max}$ for all jobs j (rather than $r_j \geq \varepsilon p_j$). Furthermore, we can bound the number of distinct processing times and the number of released jobs of each interval, using similar arguments as in the unit-speed case.

Lemma 4.3. *At $1+O(\varepsilon)$ loss we can restrict to instances where for each release date the number of released jobs and the number of distinct processing times is bounded by some constant depending only on ε, m, and $s_{\max}$.*

We establish the safety net for the jobs of each release date R_x only on the *fastest* machine and thereby ensure the condition of Lemma 2.4 in the related machine setting. For the non-preemptive setting we incorporate the adjustments introduced in Section 4.1. Since at $1+\varepsilon$ loss we can round the speeds of the machines to powers of $1+\varepsilon$ we obtain the following result.

Theorem 4.4. *We obtain online approximation schemes for* $\mathrm{Qm}|\, r_j, pmtn \,|\, \sum w_j C_j$ *and* $\mathrm{Qm}|\, r_j \,|\, \sum w_j C_j$*, assuming that the speeds of any two machines differ by at most a constant factor.*

In the preemptive setting we can strengthen the result and give an online approximation scheme for the case that machine speeds are part of the input, that is, we obtain a nearly optimal competitive ratio for *any* speed vector. The key is to bound the variety of different speeds. To that end, we show that at $1+\varepsilon$ loss a very fast machine can simulate $m-1$ very slow machines.

Lemma 4.5. *For* $\mathrm{Qm}|\, r_j, pmtn \,|\, \sum w_j C_j$*, at $1+O(\varepsilon)$ loss, we can restrict to instances in which $s_{\max}$ is bounded by m/ε.*

As speeds are geometrically rounded, we have for each value m only finitely many speed vectors. Thus, our enumeration scheme finds a nearly optimal online algorithm with a particular routine for each speed vector.

Theorem 4.6. *We obtain an online approximation scheme for* $\mathrm{Qm}|\, r_j, pmtn \,|\, \sum w_j C_j$.

4.3 Preemptive Scheduling on Unrelated Machines

When each job j has its individual processing time p_{ij} on machine i, the problem complexity increases significantly. We restrict to preemptive scheduling and show how to decrease the complexity to apply our approximation scheme. The key is to bound the range of the finite processing times for each job (which is unfortunately not possible in the non-preemptive case, see [1] for a counterexample).

Lemma 4.7. *At $1+\varepsilon$ loss we can restrict to instances in which for each job j the ratio of its finite processing times is bounded by m/ε.*

The above lemma allows us to introduce the notion of *job classes*. Two jobs j, j' are of the same class if they have finite processing times on exactly the same machines and $p_{ij}/p_{ij'} = p_{i'j}/p_{i'j'}$ for any two such machines i and i'. For fixed m, the number of different job classes is bounded by a constant W.

For each job class, we define large and small tasks similar to the identical machine case: for each job j we define a value $\tilde{p}_j := \max_i\{p_{ij} | p_{ij} < \infty\}$ and say a job is *large* if $\tilde{p}_j \geq \varepsilon^2 r_j / W$ and *small* otherwise. For each job class separately, we perform the adjustments of Section 2. This yields the following lemma.

Lemma 4.8. *At $1+O(\varepsilon)$ loss we can restrict to instances and schedules such that*

- *for each job class, the number of distinct values $\tilde{p}_j$ of jobs j with the same release date is bounded by a constant,*
- *for each job class, the number of jobs with the same release date is bounded by a constant $\tilde{\Delta}$,*
- *a large job j is only preempted at integer multiples of $\tilde{p}_j \cdot \tilde{\mu}$ for some constant $\tilde{\mu}$ and small jobs are never preempted and finish in the same interval where they start.*

The above lemmas imply that both, the number of equivalence classes of configurations and the number of equivalence classes for interval-schedules are bounded by constants. Thus, we can apply the enumeration scheme from Section 3.

Theorem 4.9. *We obtain an online approximation scheme for* $\mathrm{Rm}|\, r_j, pmtn \,|\, \sum w_j C_j$.

5 Randomized algorithms

When algorithms are allowed to make random choices and we consider expected values of schedules, we can restrict to instances which span only constantly many periods. Assuming the simplifications of Section 2, this allows a restriction to instances with a constant number of jobs.

Lemma 5.1. *For randomized algorithms, at* $1+\varepsilon$ *loss we can restrict to instances in which all jobs are released in at most* $(1+\varepsilon)^s/\varepsilon$ *consecutive periods.*

Proof idea. Beginning at a randomly chosen period Q_i with $i \in [0, M)$, with $M := \lceil (1+\varepsilon)^s/\varepsilon \rceil$, we move all jobs released in Q_{i+kM}, $k = 0, 1, \ldots$, to their safety net. At $1+\varepsilon$ loss, this gives us a partition into parts, at the end of which no job remains, and we can treat each part independently. □

A randomized online algorithm can be viewed as a function that maps every possible configuration C to a probability distribution of interval-schedules which are feasible for C. To apply our algorithmic framework from the deterministic setting that enumerates all algorithm maps, we discretize the probability space and define discretized algorithm maps. To this end, let $\bar{\Gamma}$ denote the maximum number of intervals in instances with at most $(1+\varepsilon)^s/\varepsilon$ periods.

Definition 5.2 (Discretized algorithm maps)**.** Let $\bar{\mathcal{C}}$ be the set of configurations for intervals I_x with $x \le \bar{\Gamma}$, let $\bar{\mathcal{S}}$ be the set of interval-schedules for intervals I_x with $x \le \bar{\Gamma}$, and let $\delta > 0$. A *δ-discretized algorithm map* is a function $f : \bar{\mathcal{C}} \times \bar{\mathcal{S}} \to [0,1]$ such that $f(C,S) = k \cdot \delta$ with some $k \in \mathbb{N}_0$ for all $C \in \bar{\mathcal{C}}$ and $S \in \bar{\mathcal{S}}$, and $\sum_{S \in \bar{\mathcal{S}}} f(C,S) = 1$ for all $C \in \bar{\mathcal{C}}$.

The following lemma shows that by restricting to δ-discretized algorithm maps we do not lose too much in the competitive ratio.

Lemma 5.3. *There is a value* $\delta > 0$ *such that for any (randomized) algorithm map* f *there is a* δ*-discretized randomized algorithm map* g *with* $\rho_g \le \rho_f (1+\varepsilon)$.

Proof idea. Let f be a randomized algorithm map and let $\delta > 0$ such that $1/\delta \in \mathbb{N}$. We define a new δ-discretized algorithm map g. For each configuration C we define the values $g(C,S)$ such that $\lfloor f(C,S)/\delta \rfloor \cdot \delta \le g(C,S) \le \lceil f(C,S)/\delta \rceil \cdot \delta$ and $\sum_{S \in \mathcal{S}} g(C,S) = 1$. To see that $\rho_g \le (1+\varepsilon)\rho_f$, consider an instance $\mathcal{I}$ and a possible schedule $S(\mathcal{I})$ for $\mathcal{I}$. There is a probability p that f outputs $S(\mathcal{I})$. We show that the schedules with large probability p dominate $\mathbb{E}[\, f(\mathcal{I}) \,]/\mathsf{OPT}(\mathcal{I})$. We show further that if p is sufficiently large, the probability that g produces $S(\mathcal{I})$ is in $[p/(1+\varepsilon), p(1+\varepsilon))$, which implies the Lemma. □

Like in the deterministic case, we can now show that at $1+\varepsilon$ loss it suffices to restrict to *simplified δ-discretized algorithm maps* which treat equivalent configurations equivalently, similar to Lemma 3.6 (replacing Γ by $\bar{\Gamma}$ in Definition 2.9 of the irrelevant jobs). As there are only constantly many of these maps, we enumerate all of them, test each map for its competitive ratio, and select the best of them.

Theorem 5.4. *We obtain randomized online approximation schemes for* $\mathrm{Pm}|\, r_j, (pmtn) \,|\, \sum w_j C_j$, $\mathrm{Qm}|\, r_j, pmtn \,|\, \sum w_j C_j$, *and* $\mathrm{Rm}|\, r_j, pmtn \,|\, \sum w_j C_j$ *and for* $\mathrm{Qm}|\, r_j \,|\, \sum w_j C_j$ *with a bounded range of machine speeds.*

References

[1] F. N. Afrati, E. Bampis, C. Chekuri, D. R. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein, and M. Sviridenko. Approximation schemes for minimizing average weighted completion time with release dates. In *Proceedings of the 40th IEEE Symposium on the Foundations of Computer Science (FOCS)*, pages 32–43, 1999.

[2] E. J. Anderson and C. N. Potts. On-line scheduling of a single machine to minimize total weighted completion time. *Mathematics of Operations Research*, 29:686–697, 2004.

[3] S. Chakrabarti, C. A.Phillips, A. S. Schulz, D. B. Shmoys, C. Stein, and J. Wein. Improved algorithms for minsum criteria. In *Proceedings of the 23rd International Conference on Automata, Languages and Programming (ICALP)*, volume 1099, pages 646–657, 1996.

[4] C. Chekuri, R. Motwani, B. Natarajan, and C. Stein. Approximation techniques for average completion time scheduling. *SIAM Journal on Computing*, 31:146–166, 2001.

[5] C. Chung, T. Nonner, and A. Souza. Srpt is 1.86-competitive for completion time scheduling. In M. Charikar, editor, *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1373–1388. SIAM, 2010.

[6] J. Correa and M. Wagner. LP-based online scheduling: From single to parallel machines. *Mathematical Programming*, 119:109–136, 2009.

[7] L. Epstein and R. van Stee. Lower bounds for on-line single-machine scheduling. *Theoretical Computer Science*, 299:439–450, 2003.

[8] M. X. Goemans. Improved approximation algorithms for scheduling with release dates. In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 591–598, 1997.

[9] M. X. Goemans, M. Queyranne, A. S. Schulz, M. Skutella, and Y. Wang. Single machine scheduling with release dates. *SIAM Journal on Discrete Mathematics*, 15:165–192, 2002.

[10] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[11] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Mathematics of Operations Research*, 22:513–544, 1997.

[12] H. Hoogeveen and A. P. A. Vestjens. Optimal on-line algorithms for single-machine scheduling. In W. H. Cunningham, S. T. McCormick, and M. Queyranne, editors, *Proceedings of the 5th International Conference on Integer Programming and Combinatorial Optimization (IPCO)*, volume 1084, pages 404–414, 1996.

[13] A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator. Competitive snoopy paging. *Algorithmica*, 3:70–119, 1988.

[14] J. Labetoulle, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Preemptive scheduling of uniform machines subject to release dates. In W. R. Pulleyblank, editor, *Progress in Combinatorial Optimization*, pages 245–261. Academic Press Canada, Waterloo, ON, Canada, 1984.

[15] J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:243–362, 1977.

[16] P. Liu and X. Lu. On-line scheduling of parallel machines to minimize total completion times. *Computers and Operations Research*, 36:2647–2652, 2009.

[17] X. Lu, R. A. Sitters, and L. Stougie. A class of on-line scheduling algorithms to minimize total completion time. *Operations Research Letters*, 31:232–236, 2003.

[18] N. Megow. *Coping with incomplete information in scheduling—stochastic and online models*. PhD thesis, Technische Universität Berlin, 2006. Published by Cuvillier, Göttingen, Germany, 2007.

[19] N. Megow and A. S. Schulz. On-line scheduling to minimize average completion time revisited. *Operations Research Letters*, 32(5):485–490, 2004.

[20] C. A. Phillips, C. Stein, and J. Wein. Minimizing average completion time in the presence of release dates. *Mathematical Programming*, 82:199–223, 1998.

[21] L. Schrage. A proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 16:687–690, 1968.

[22] A. S. Schulz and M. Skutella. The power of α-points in preemptive single machine scheduling. *Journal of Scheduling*, 5:121–133, 2002.

[23] A. S. Schulz and M. Skutella. Scheduling unrelated machines by randomized rounding. *SIAM Journal on Discrete Mathematics*, 15:450–469, 2002.

[24] S. S. Seiden. A guessing game and randomized online algorithms. In *Proceedings of the 32nd ACM Symposium on the Theory of Computing (STOC)*, pages 592–601, 2000.

[25] R. Sitters. Competitive analysis of preemptive single-machine scheduling. *Operations Research Letters*, 38(6):585–588, 2010.

[26] R. Sitters. Efficient algorithms for average completion time scheduling. In *Proceedings of the 14th International Conference on Integer Programming and Combinatorial Optimization, (IPCO)*, volume 6080, pages 411–423. Springer, 2010.

[27] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985.

[28] W. E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3:59–66, 1956.

[29] L. Stougie and A. P. A. Vestjens. Randomized on-line scheduling: How low can't you go? *Operations Research Letters*, 30:89–96, 2002.

[30] A. P. A. Vestjens. *On-line Machine Scheduling*. PhD thesis, Eindhoven University of Technology, Netherlands, 1997.

A Proofs of Section 2

First, we will show that the number of distinct processing times of large jobs in each interval can be upper-bounded by a constant. To achieve this, we partition the jobs of an instance into large and small jobs. With respect to a release date R_x we say that a job j with $r_j = R_x$ is *large* if $p_j \geq \varepsilon^2 I_x = \varepsilon^3 R_x$ and *small* otherwise. Abusing notation, we refer to $|I_x|$ also by I_x. Note that $I_x = \varepsilon \cdot (1+\varepsilon)^x$.

Lemma A.1. *The number of distinct processing times of jobs in each set L_x is bounded by $4 \log_{1+\varepsilon} \frac{1}{\varepsilon}$.*

Proof. For any $j \in L_x$ the processing time p_j is a power of $1+\varepsilon$, say $p_j = (1+\varepsilon)^y$. Hence, we have that $\varepsilon^3 (1+\varepsilon)^x < (1+\varepsilon)^y \leq \frac{1}{\varepsilon}(1+\varepsilon)^x$. The number of integers y which satisfy the above inequalities is bounded by $4 \log_{1+\varepsilon} \frac{1}{\varepsilon}$, which yields the constant claimed in the lemma. □

Furthermore, we can bound the number of large jobs of each job size which are released at the same time.

Lemma A.2. *Without loss, we can restrict to instances with $|L_x| \leq (m/\varepsilon^2 + m) 4 \log_{1+\varepsilon} \frac{1}{\varepsilon}$ for each set L_x.*

Proof. Let $L_{x,p} \subseteq L_x$ denote the set of jobs in L_x with processing time p. By an exchange argument, one can show that in an optimal schedule at each point in time at most m jobs in $L_{x,p}$ are partially (i.e., to some extent but not completely) processed. Since $p_j \geq \varepsilon^2 I_x$ for each job $j \in L_x$, at most $m/\varepsilon^2 + m$ jobs in $L_{x,p}$ are touched within I_x. By an exchange argument we can assume that they are the $m/\varepsilon^2 + m$ jobs with the largest weight in $L_{x,p}$. Hence, the release date of all other jobs in $L_{x,p}$ can be moved to R_{x+1} without any cost. Since due to Lemma A.1 there are at most $4 \log_{1+\varepsilon} \frac{1}{\varepsilon}$ distinct processing times p of large jobs in L_x, the claim follows. □

We now just need to take care of the small jobs. Denote by w_j/p_j the *Smith's ratio* of a job j. An ordering where the jobs are ordered non-increasingly by their Smith's ratios is an ordering according to *Smith's rule*. The next lemma shows that scheduling the small jobs according to Smith's Rule is almost optimal and small jobs do not even need to be preempted or to cross intervals. For a set of jobs J we define $p(J) := \sum_{j \in J} p_j$.

Lemma A.3. *At $1+\varepsilon$ loss we can restrict to schedules such that for each interval I_x the small jobs scheduled within this interval are chosen by Smith's Rule from the set $\bigcup_{x' \leq x} S_{x'}$, no small job is preempted, any small job finishes in the same interval where it started and $p(S_x) \leq m \cdot I_x$ for each interval I_x.*

Proof. By an exchange argument one can show that it is optimal to schedule the small jobs by Smith's Rule if they can be arbitrarily divided into smaller jobs (where the weight is divided proportional to the processing time of the smaller jobs). Start with such a schedule and stretch time once. The gained free space is enough to finish all small jobs which are partially scheduled in each interval.

For the last claim of the lemma, note that the total processing time in each interval I_x is mI_x. Order the small jobs non-increasingly by their Smith's Ratios and pick them until the total processing time of picked jobs just does not exceed mI_x. The release date of all other jobs in S_x can be safely moved to R_{x+1} since due to our modifications we would not schedule them in I_x anyway. □

Lemma 2.4 (restated)**.** *There is a constant s such that at $1 + O(\varepsilon)$ loss we can restrict to schedules such that for each interval I_x there is a subinterval of I_{x+s-1} as large as the total volume of the jobs released at R_x during which only jobs in R_x are executed. We call this subinterval the* safety net *of interval I_x. We can assume that each job released at R_x finishes before time R_{x+s}.*

Proof. By Lemmas A.3 and A.2 we bound $p(S_x) + p(L_x)$ by

$$\begin{aligned} p(S_x) + p(L_x) &\leq m \cdot I_x + (m/\varepsilon^2 + m) \cdot \left(4 \log_{1+\varepsilon} \frac{1}{\varepsilon}\right) \cdot \frac{1}{\varepsilon} (1+\varepsilon)^x \\ &\leq m \cdot (1+\varepsilon)^x \left(\varepsilon + \frac{8}{\varepsilon^3} \log_{1+\varepsilon} \frac{1}{\varepsilon}\right) \\ &= \varepsilon \cdot I_{x+s-1} \end{aligned}$$

for a suitable constant s, depending on ε and m. Stretching time once, we gain enough free space at the end of each interval I_{x+s-1} to establish the safety net for each job set $p(S_x) + p(L_x)$.

□

Lemma A.4. *There is a constant d such that we can at $1+O(\varepsilon)$ loss restrict to instances such that $p_j > \frac{\varepsilon}{2d} \cdot I_x$ for each job $j \in S_x \cup L_x$.*

Proof. We call a job j *tiny* if $p_j \leq \frac{\varepsilon}{2d} \cdot I_x$. Let $T_x = \{j_1, j_2, ..., j_{|T_x|}\}$ denote all tiny jobs released at R_x. W.l.o.g. assume that they are ordered non-increasingly by their Smith's Ratios w_j/p_j. Let ℓ be the largest integer such that $\sum_{i=1}^{\ell} p_i \leq \frac{\varepsilon}{d} \cdot I_x$. We define the pack $P_x^1 := \{j_1, ..., j_\ell\}$. We denote by $\sum_{i=1}^{\ell} p_i$ the processing time of pack P_x^1 and by $\sum_{i=1}^{\ell} w_i$ its weight. We continue iteratively until we assigned all tiny jobs to packs. By definition of the processing time of tiny jobs, the processing time of all but possibly the last pack released at time R_x is in the interval $[\frac{\varepsilon}{2d} \cdot I_x, \frac{\varepsilon}{d} \cdot I_x]$.

Using timestretching, we can show that at $1 + O(\varepsilon)$ loss all tiny jobs of the same pack are scheduled in the same interval on the same machine. Here we use that in any schedule obeying Smith's Rule and using the safety net (see Lemma 2.4) in each interval there is at most one partially but unfinished pack from each of at most s previous release dates. Hence, we can treat the packs as single jobs whose processing time and weight matches the respective values of the packs. Also, at $1 + \varepsilon$ loss we can ensure that also the very last pack has a processing time in $[\frac{\varepsilon}{2d} \cdot I_x, \frac{\varepsilon}{d} \cdot I_x]$. Finally, at $1 + O(\varepsilon)$ loss we can ensure that the processing times and weights of the new jobs (which replace the packs) are powers of $1 + \varepsilon$. □

Lemma A.5. *Assume that there is a constant d such that $p_j > \frac{\varepsilon}{2d} \cdot I_x$ for each job $j \in S_x$. Then at $1 + O(\varepsilon)$ loss, the number of distinct processing times of jobs each set S_x is upper-bounded by* $(\log_{1+\varepsilon} \varepsilon \cdot 2d)$.

Proof. From the previous lemmas, we have

$$\frac{e^2}{2d} \cdot (1+\varepsilon)^x < (1+\varepsilon)^y < \varepsilon^3 (1+\varepsilon)^x.$$

The number of integers y satisfying these inequalities is upper-bounded by the claimed constant. □

Lemma 2.2 now follows from the lemmas A.1 and A.5. Lemma 2.3 follows from lemmas A.3, A.4 and A.2. Next, we prove Lemma 2.5:

Proof of Lemma 2.5. The claim about the number of partially processed jobs of each type can be assumed without any loss. For the extent of processing, note that due to Lemmas A.2, 2.4, and A.4 there is a constant c such that at each time R_x the total processing time of unfinished large jobs is bounded by $c \cdot R_x$. We stretch time once. The gained space is sufficient to schedule $p_j \cdot \mu$ processing units of each unfinished large job j (for an appropriately chosen universal constant μ). This allows us to enforce the claim. The claim about the non-preemptive behavior of small jobs follows from Lemma A.3. □

Proof of Lemma 2.6. In any schedule the jobs in $\cup_{i=0}^{p-1} Q_{k+i}$ contribute at least $\sum_{i=0}^{p-1} rw(Q_{k+i})$ towards the objective. If we move all jobs in Q_{k+p} to their safety nets, they contribute at most

$$\begin{aligned}\sum_{j\in Q_{k+p}} r_j\,(1+\varepsilon)^s \cdot w_j &= (1+\varepsilon)^s \cdot rw(Q_{k+p}) \\ &\leq \varepsilon \cdot \sum_{i=0}^{p-1} rw(Q_{k+i}) \\ &\leq \varepsilon \cdot OPT\end{aligned}$$

to the objective. □

Proof of Lemma 2.7. We modify a given online algorithm such that each part is treated as a separate instance. To bound the cost in the competitive ratio, we show that $\frac{\mathsf{A}(\mathcal{I})}{\mathsf{OPT}(\mathcal{I})} \leq \max_i\{\frac{\mathsf{A}(P_i)}{\mathsf{OPT}(P_i)}\} \cdot (1 + O(\varepsilon))$. By the above lemmas, there is a $(1 + O(\varepsilon))$-approximative (offline) solution in which at the end of each part P_i each job has either completed or has been moved to its safety net. Denote this solution by $\mathsf{OPT}'(I)$ and by $\mathsf{OPT}'(P_i)$ its respective part for each part P_i. Note that $\mathsf{OPT}'(\mathcal{I}) = \sum_i \mathsf{OPT}'(P_i)$. Then,

$$\frac{\sum_{i=1}^{k} \mathsf{A}(P_i))}{\mathsf{OPT}(\mathcal{I})} \leq \frac{\sum_{i=1}^{k} \mathsf{A}(P_i))}{\sum_{i=1}^{k} \mathsf{OPT}'(P_i)} \cdot (1 + O(\varepsilon)) \leq \max_{i=1,\ldots,k}\{\frac{\mathsf{A}(P_i)}{\mathsf{OPT}(P_i)}\} \cdot (1 + O(\varepsilon)).$$

□

Proof of Lemma 2.8. We show that $(1+\varepsilon)^s \sum_{i=1}^{p-K-1} rw(Q_i) < \varepsilon \cdot \sum_{i=p-K}^{p} rw(Q_i)$ for a sufficiently large value K. This will then be the claimed constant. Let $\delta' := \frac{\varepsilon}{(1+\varepsilon)^s}$. By assumption, we have that $rw(Q_{i+1}) > \delta' \cdot \sum_{\ell=1}^{i} rw(Q_\ell)$ for each i. This implies that $\frac{rw(Q_{i+1})}{\sum_{\ell=1}^{i+1} rw(Q_\ell)} > \frac{\delta'}{1+\delta'}$ for each i. Hence,

$$\frac{\sum_{\ell=1}^{i} rw(Q_\ell)}{rw(Q_{i+1}) + \sum_{\ell=1}^{i} rw(Q_\ell)} \leq 1 - \frac{\delta'}{1+\delta'} < 1$$

for each i and hence,

$$\sum_{\ell=1}^{i} rw(Q_\ell) \leq (1 - \frac{\delta'}{1+\delta'}) \sum_{\ell=1}^{i+1} rw(Q_\ell).$$

In other words, if we remove Q_{i+1} from $\cup_{\ell=1}^{i+1} Q_\ell$ then the total release weight of the set decreases by a factor of at least $1 - \delta'/(1+\delta') < 1$. For any K this implies that

$$\sum_{i=1}^{p-K-1} rw(Q_i) < \left(1 - \frac{\delta'}{1+\delta'}\right)^K \sum_{\ell=1}^{p} rw(Q_\ell)$$

and hence

$$\sum_{i=1}^{p-K-1} rw(Q_i) < \frac{1}{1 - \left(1 - \frac{\delta'}{1+\delta'}\right)^K} \left(1 - \frac{\delta'}{1+\delta'}\right)^K \sum_{i=p-K}^{p} rw(Q_i).$$

By choosing K sufficiently large, the claim follows. □

Proof of Lemma 2.10. We partition $\mathrm{Ir}_x(J)$ into two groups: $\mathrm{Ir}_x^{\mathrm{old}}(J) := \{j \in \mathrm{Ir}_x(J) | r_j < R_{x-\Gamma}\}$ and $\mathrm{Ir}_x^{\mathrm{new}}(J) := \{j \in \mathrm{Ir}_x(J) | r_j \geq R_{x-\Gamma}\}$. Lemma 2.8 implies that

$$(1+\varepsilon)^s rw(\mathrm{Ir}_x^{\mathrm{old}}(J)) \leq \varepsilon \cdot rw(\mathrm{Ir}_x^{\mathrm{new}}(J) \cup \mathrm{Rel}_x(J)) \tag{A.1}$$

(recall that the former value is an upper bound on the total weighted completion time of the jobs in $\mathrm{Ir}_x^{\mathrm{old}}(J)$). For every job $j \in \mathrm{Ir}_x^{\mathrm{new}}(J)$ there must be a job $j' \in \mathrm{Ir}_x^{\mathrm{old}}(J) \cup \mathrm{Rel}_x(J)$ such that $w_j < \frac{\varepsilon}{\Delta\Gamma \cdot (1+\varepsilon)^{\Gamma+s}} w_{j'}$. We say that such a job j' *dominates* j. At most Δ jobs are released at the beginning of each interval and hence $|\mathrm{Ir}_x^{\mathrm{new}}(J)| \leq \Delta\Gamma$. In particular, if $\mathrm{dom}(j')$ denotes all jobs in $\mathrm{Ir}_x^{\mathrm{new}}(J)$ which are dominated by j' then

$$\sum_{j \in \mathrm{dom}(j')} w_j r_j \leq \Delta\Gamma \frac{\varepsilon}{\Delta\Gamma \cdot (1+\varepsilon)^{\Gamma+s}} w_{j'} r_{j'} \cdot (1+\varepsilon)^{\Gamma}$$

This implies that

$$\begin{aligned}
(1+\varepsilon)^s rw(\mathrm{Ir}_x^{\mathrm{new}}(J)) &\leq (1+\varepsilon)^s \sum_{j \in \mathrm{Ir}_x^{\mathrm{new}}(J)} w_j r_j \\
&\leq \sum_{j' \in \mathrm{Ir}_x^{\mathrm{old}}(J) \cup \mathrm{Rel}_x(J)} \Delta\Gamma \frac{\varepsilon}{\Delta\Gamma \cdot (1+\varepsilon)^{\Gamma+s}} w_{j'} r_{j'} \cdot (1+\varepsilon)^{\Gamma+s} \\
&\leq \varepsilon \cdot \sum_{j' \in \mathrm{Ir}_x^{\mathrm{old}}(J) \cup \mathrm{Rel}_x(J)} w_{j'} r_{j'} \\
&= \varepsilon \cdot (rw(\mathrm{Ir}_x^{\mathrm{old}}(J)) + rw(\mathrm{Rel}_x(J)))
\end{aligned}$$

Together with Inequality A.1 this implies that

$$\begin{aligned}
(1+\varepsilon)^s rw(\mathrm{Ir}_x(J)) &= (1+\varepsilon)^s (rw(\mathrm{Ir}_x^{\mathrm{new}}(J) + rw(\mathrm{Ir}_x^{\mathrm{old}}(J))) \\
&\leq \Big(\varepsilon \cdot (rw(\mathrm{Ir}_x^{\mathrm{old}}(J)) + rw(\mathrm{Rel}_x(J))\Big) + (\varepsilon \cdot rw(\mathrm{Ir}_x^{\mathrm{new}}(J) \cup \mathrm{Rel}_x(J))) \\
&\leq 2\varepsilon \cdot rw(\mathrm{Rel}_x(J)) + \varepsilon(rw(\mathrm{Ir}_x(J))
\end{aligned}$$

and the latter inequality implies that

$$\begin{aligned}
\sum_{j \in \mathrm{Ir}_x(J)} C_j w_j &\leq (1+\varepsilon)^s rw(\mathrm{Ir}_x(J)) \\
&\leq 2\varepsilon \frac{(1+\varepsilon)^s}{(1+\varepsilon)^s - \varepsilon} rw(\mathrm{Rel}_x(J)) \\
&\leq 3\varepsilon \cdot rw(\mathrm{Rel}_x(J))
\end{aligned}$$

□

B Proofs of Section 4

Lemma B.1. *In the non-preemptive setting, at $1+O(\varepsilon)$ loss we can ensure that at the end of each interval I_x,*

- *there are at most m large jobs from each type which are partially (i.e., neither fully nor not at all) processed, and*
- *for each partially but not completely processed large job j there is a value $k_{x,j}$ such that j is processed for at least $k_{x,j} \cdot p_j \cdot \mu$ time units in I_x,*

- *we calculate the objective with adjusted completion times $\bar{C}_j = R_{c(j)}$ for some value $c(j)$ for each job j such that $\sum_{x<c(j)} k_{x,j} \cdot p_j \cdot \mu \geq p_j$.*

Proof. Note that the first property holds for any non-preemptive schedule and is listed here only for the sake of clarity. The other two properties can be shown similiarly as in the proof of Lemma 2.5. □

Proof of Lemma 4.1. Assume that we have an online algorithm A with competitive factor ρ_{A} on instances in which for every i the first job first(i) released in part P_i satisfies $\sum_{\ell=1}^{i-1} rw(P_\ell) \leq w_{\text{first}(i)}\varepsilon/(1+\varepsilon)^s$ (i.e., first(i) dominates all previously released parts). Based on A we construct a new algorithm A' for arbitrary instances with competitive ratio at most $(1+\varepsilon)\rho_{\mathsf{A}}$: When a new part P_i begins, we scale the weights of all jobs in P_i such that $\sum_{\ell=1}^{i-1} rw(P_\ell) \leq w'_{\text{first}(i)}\varepsilon/(1+\varepsilon)^s$, where the values w'_j denote the adjusted weights. Denote by $\bar{I}(i)$ the resulting instance up to (and including) part P_i. We schedule the resulting instance using A. We take the computed schedule for each part P_i and use it for the jobs with their original weight, obtaining a new algorithm A'. The following calculations shows that this procedure costs only a factor $1+\varepsilon$. To this end, we proof that for any instance I it holds that

$$\frac{\mathsf{A}'(\mathcal{I})}{\mathsf{OPT}(\mathcal{I})} \leq \max_i \frac{\mathsf{A}(\bar{I}(i))}{\mathsf{OPT}(\bar{I}(i))} \cdot (1+O(\varepsilon)) \leq (1+O(\varepsilon))\rho_{\mathsf{A}}.$$

For each P_i we define $\mathsf{A}'(\mathcal{I}|P_i)$ to be the amount that the jobs in P_i contribute in $\mathsf{A}'(\mathcal{I})$. Similarly, we define $\mathsf{OPT}(\mathcal{I}|P_i)$. We have that

$$\frac{\mathsf{A}'(\mathcal{I})}{\mathsf{OPT}(\mathcal{I})} \leq \max_i \frac{\mathsf{A}'(\mathcal{I}|P_i)}{\mathsf{OPT}(\mathcal{I}|P_i)} \leq \max_i \frac{\mathsf{A}'(\mathcal{I}|P_i)}{\mathsf{OPT}(P_i)}.$$

We claim that for each i holds $\frac{\mathsf{A}'(\mathcal{I}|P_i)}{\mathsf{OPT}(P_i)} \leq (1+O(\varepsilon)) \cdot \frac{\mathsf{A}(\bar{I}(i))}{\mathsf{OPT}(\bar{I}(i))}$. For each part P_i let v_i denote the scale factor of the weight of each job in $\bar{I}(i)$ in comparison to its original weight. The optimum for the instance $\bar{I}(i)$ can be bounded by

$$\mathsf{OPT}(\bar{I}(i)) \quad \leq \quad \mathsf{OPT}(P_i) \cdot v_i + (1+\varepsilon)^s \sum_{\ell=1}^{i-1} rw(P_\ell) \leq \mathsf{OPT}(P_i) \cdot v_i + \varepsilon \cdot r_{\text{first}(i)} \cdot w'_{\text{first}(i)} \leq (1+\varepsilon)\,\mathsf{OPT}(P_i) \cdot v_i.$$

Furthermore holds by construction $\mathsf{A}'(\mathcal{I}|P_i) \cdot v_i \leq \mathsf{A}(\bar{I}^i)$. Thus, $\max_i \frac{\mathsf{A}'(I|P_i)}{\mathsf{OPT}(P_i)} \leq \max_i \frac{\mathsf{A}(\bar{I}^i)}{\mathsf{OPT}(\bar{I}^i)} \cdot (1+O(\varepsilon))$. □

Proof of Lemma 4.5. Given a schedule on related machines with speed values $s_1, ..., s_{\max}$, we stretch time twice. Thus, we gain in each interval I_x free space of size εI_x on the fastest machine. For each machine whose speed is at most $\frac{\varepsilon}{m} s_{\max}$, we take its schedule of the interval I_x and simulate it on the fastest machine. Thus, those slow machines are not needed and can be removed. The remaining machines have speeds in $[\frac{\varepsilon}{m} s_{\max}, s_{\max}]$. Assuming the slowest machines has unit speed gives the desired bound. □

Proof of Lemma 4.7. Consider a schedule for an instance which does not satisfy the property. We stretch time twice and thus we gain a free space of εI_x in each interval I_x. Consider some I_x and a job j which is scheduled in I_x. Let i be a fastest machine for j. We remove the processing volume of j scheduled in I_x on slow machines i' with $p_{i'j} > \frac{m}{\varepsilon} p_{ij}$ and schedule it on i in the gained free space. This way, we obtain a feasible schedule even if a job never runs on a machine where it is slow. Thus, we can set $p_{i'j} = \infty$ if there is a fast machine i such that $p_{ij} \leq \frac{\varepsilon}{m} p_{i'j}$. □

C Proofs of Section 5

Proof of Lemma 5.1. Let A be a randomized online algorithm with a competitive ratio of ρ_{A} on instances which span at most $(1+\varepsilon)^s/\varepsilon$ periods. We construct a new randomized online algorithm A' which works on arbitrary instances $\mathcal{I}$ such that $\rho_{\mathsf{A}'} \leq \rho_{\mathsf{A}}(1+\varepsilon)$. At the beginning of A', we choose an offset $o \in \{0, ..., M-1\}$ uniformly at random with $M := \lceil (1+\varepsilon)^s/\varepsilon \rceil$. In instance $\mathcal{I}$, we move all jobs to their safety net which are released in periods $Q \in \mathcal{Q} := \{Q_i | i \equiv o \bmod M\}$. This splits the instance into parts $P_1, ..., P_k$ where each part P_ℓ consists of the periods $Q_{o+(\ell-1)\cdot M}, ..., Q_{o+\ell\cdot M-1}$. Note that at the end of each part no job remains. We need to bound the increase in the total expected cost caused by moving all jobs in periods in $\mathcal{Q}$ to their safety nets. This increase is bounded by

$$\begin{aligned}
\mathbb{E}\left[\sum_{Q\in\mathcal{Q}}\sum_{j\in Q}(1+\varepsilon)^s r_j\cdot w_j\right] &\leq (1+\varepsilon)^s\,\mathbb{E}\left[\sum_{Q\in\mathcal{Q}} rw(Q)\right]\\
&\leq (1+\varepsilon)^s\,\frac{1}{M}\sum_{Q\in\mathcal{I}} rw(Q)\\
&\leq \varepsilon\cdot rw(\mathcal{I})\\
&\leq \varepsilon\cdot \mathsf{OPT}(\mathcal{I}).
\end{aligned}$$

Thus, the total expected cost of the computed schedule is

$$\begin{aligned}
\mathbb{E}\left[\varepsilon\cdot\mathsf{OPT}(\mathcal{I}) + \sum_{i=1}^{k}\mathsf{A}(P_i)\right] &\leq \varepsilon\cdot\mathsf{OPT}(\mathcal{I}) + \sum_{i=1}^{k}\rho_{\mathsf{A}}\cdot\mathsf{OPT}(P_i)\\
&\leq \varepsilon\cdot\mathsf{OPT}(\mathcal{I}) + \rho_{\mathsf{A}}\cdot\mathsf{OPT}(\mathcal{I})\\
&\leq (\rho_{\mathsf{A}}+\varepsilon)\cdot\mathsf{OPT}(\mathcal{I})\\
&\leq \rho_{\mathsf{A}}(1+\varepsilon)\cdot\mathsf{OPT}(\mathcal{I}).
\end{aligned}$$

Thus, at $1+\varepsilon$ loss in the competitive ratio we can restrict to parts I_i which span a constant number of periods. □

Proof of Lemma 5.3. Consider an instance $\mathcal{I}$. Let $\delta > 0$ and $k \in \mathbb{N}$ be values to be determined later with the property that $1/\delta \in \mathbb{N}$. For each configuration C and each interval-schedule S we define a value $g(C,S)$ such that $\left\lfloor \frac{f(C,S)}{\delta} \right\rfloor \cdot \delta \leq g(C,S) \leq \left\lceil \frac{f(C,S)}{\delta} \right\rceil \cdot \delta$ and $\sum_{S\in\mathcal{S}} g(C,S) = 1$. Now we want to bound ρ_g.

The idea is that for determining the ratio $\mathbb{E}[\,g(\mathcal{I})\,]/\mathsf{OPT}(\mathcal{I})$ it suffices to consider schedules $S(\mathcal{I})$ which are computed with sufficiently large probability. We show that also f computes them with almost the same probability. Let $S(\mathcal{I})$ denote a schedule for the entire instance $\mathcal{I}$. We denote by $P_f(S(\mathcal{I}))$ and $P_g(S(\mathcal{I}))$ the probability that f and g compute the schedule $S(\mathcal{I})$ when given the instance $\mathcal{I}$. Assume that $P_f(S(\mathcal{I})) \geq k\cdot\delta$. Denote by $C_0, ..., C_{\bar{\Gamma}-1}$ the configurations that algorithms are faced with when computing $S(\mathcal{I})$, i.e., each configuration C_x contains the jobs which are released but unfinished at the beginning of interval I_x in $S(\mathcal{I})$ and as history the schedule $S(\mathcal{I})$ restricted to the intervals $I_0, ..., I_{x-1}$. Denote by S_x the schedule of $S(\mathcal{I})$ in the interval I_x. Hence, $P_f(S(\mathcal{I})) = \prod_{x=0}^{\bar{\Gamma}-1} f(C_x, S_x)$. Note that from $P_f(S(\mathcal{I})) \geq k\cdot\delta$ follows that $f(C_x,S_x) \geq k\cdot\delta$ for all x. For these schedules, $P_g(S(\mathcal{I}))$ is not much larger since

$$P_g(S(\mathcal{I})) = \prod_{x=0}^{\bar{\Gamma}-1} g(C_x,S_x) \leq \prod_{x=0}^{\bar{\Gamma}-1}\frac{k+1}{k} f(C_x,S_x) \leq \left(\frac{k+1}{k}\right)^{\bar{\Gamma}} P_f(S(\mathcal{I})).$$

Let $\mathcal{S}(\mathcal{I})$ denote the set of all schedules for $\mathcal{I}$. We partition $\mathcal{S}(\mathcal{I})$ into schedule sets $\mathcal{S}_H^g(\mathcal{I}) := \{S(\mathcal{I}) | P_g(S(\mathcal{I})) \geq k \cdot \delta\}$ and $\mathcal{S}_L^g(\mathcal{I}) := \mathcal{S}(\mathcal{I}) \setminus \mathcal{S}_H(\mathcal{I})$. We estimate the expected value of a schedule computed by algorithm map g on $\mathcal{I}$ by

$$\begin{aligned}
\mathbb{E}[\, g(\mathcal{I})\,] &= \sum_{S(\mathcal{I}) \in \mathcal{S}_H^g(\mathcal{I})} P_g(S(\mathcal{I})) \cdot S(\mathcal{I}) + \sum_{S(\mathcal{I}) \in \mathcal{S}_L^g(\mathcal{I})} P_g(S(\mathcal{I})) \cdot S(\mathcal{I}) \\
&\leq \sum_{S(\mathcal{I}) \in \mathcal{S}_H^g(\mathcal{I})} P_f(S(\mathcal{I})) \cdot \left(\frac{k+1}{k}\right)^{\bar{\Gamma}} \cdot S(\mathcal{I}) + |\mathcal{S}(\mathcal{I})| \cdot k \cdot \delta \cdot (1+\varepsilon)^s \cdot rw(\mathcal{I}) \\
&\leq \left(\frac{k+1}{k}\right)^{\bar{\Gamma}} \sum_{S(\mathcal{I}) \in \mathcal{S}_H^g(\mathcal{I})} P_f(S(\mathcal{I})) \cdot S(\mathcal{I}) + |\mathcal{S}(\mathcal{I})| \cdot k \cdot \delta \cdot (1+\varepsilon)^s \cdot rw(\mathcal{I}) \\
&\leq \left(\frac{k+1}{k}\right)^{\bar{\Gamma}} \mathbb{E}[\, f(\mathcal{I})\,] + |\mathcal{S}(\mathcal{I})| \cdot k \cdot \delta \cdot (1+\varepsilon)^s \cdot rw(\mathcal{I}).
\end{aligned}$$

We choose k such that $\left(\frac{k+1}{k}\right)^{\bar{\Gamma}} \leq 1+\varepsilon/2$ and δ such that $|\mathcal{S}(\mathcal{I})| \cdot k \cdot \delta \cdot (1+\varepsilon)^s \leq \varepsilon/2$ for all instances $\mathcal{I}$ (note here that $|\mathcal{S}(\mathcal{I})|$ can be upper bounded by a value independent of $\mathcal{I}$ since our instances contain only constantly many jobs). This yields

$$\frac{\mathbb{E}[\, g(\mathcal{I})\,]}{\mathsf{OPT}(\mathcal{I})} \leq (1+\varepsilon/2) \cdot \frac{\mathbb{E}[\, f(\mathcal{I})\,]}{\mathsf{OPT}(\mathcal{I})} + \varepsilon/2 \cdot \frac{rw(\mathcal{I})}{\mathsf{OPT}(\mathcal{I})} \leq (1+\varepsilon) \cdot \frac{\mathbb{E}[\, f(\mathcal{I})\,]}{\mathsf{OPT}(\mathcal{I})},$$

and we conclude that $\rho_g \leq (1+\varepsilon)\rho_f$. □