A New Approach to Online Scheduling: Approximating the Optimal Competitive Ratio

ELISABETH LÜBBECKE, OLAF MAURER, NICOLE MEGOW, and ANDREAS WIESE,
Technical University of Berlin

We propose a new approach to competitive analysis in online scheduling by introducing the novel concept of competitive-ratio approximation schemes. Such a 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 that are almost best possible among all online algorithms of the respective settings. We also generalize our techniques to arbitrary monomial cost functions and apply them to the makespan objective. 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 with nearly all previous manually obtained competitiveness results, and, most importantly, it reduces the search for the optimal competitive ratio to a question that a computer can answer. We believe that our concept can also be applied to many other problems and yields a new perspective on online algorithms in general.

Categories and Subject Descriptors: F.1.2 [**Computation by Abstract Devices**]: Modes of Computation—*Online computation*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Computations on discrete structures*; G.2.2 [**Discrete Mathematics**]: Combinatorics—*Combinatorial algorithms*; G.2.3 [**Discrete Mathematics**]: Applications; I.2.8 [**Artifical Intelligence**]: Problem Solving, Control Methods and Search—*Scheduling*

General Terms: Algorithms, Design, Theory

Additional Key Words and Phrases: Competitive analysis, jobs arrive over time, min-sum objective, makespan, online scheduling

ACM Reference Format:
Elisabeth Lübbecke, Olaf Maurer, Nicole Megow, and Andreas Wiese. 2016. A new approach to online scheduling: Approximating the optimal competitive ratio. ACM Trans. Algorithms 13, 1, Article 15 (December 2016), 34 pages.
DOI: http://dx.doi.org/10.1145/2996800

Elisabeth Lübbecke and Olaf Maurer were supported by the DFG Research center MATHEON *Mathematics for key technologies* in Berlin.
Nicole Megow was supported by the German Science Foundation (DFG) under contract ME 3825/1. Andreas Wiese was supported by the DFG Focus Program 1307 and by a fellowship within the Postdoc-Programme of the German Academic Exchange Service (DAAD).
Authors' addresses: E. Lübbecke, Workforce Management Division, INFORM GmbH, Pascalstr. 23, 52076 Aachen, Germany; email: elisabeth.luebbecke@inform-software.com; O. Maurer, Faculty of Mathematics and Natural Sciences, Kassel University, Heinrich-Plett-Strasse 40, 34132 Kassel, Germany; email: olaf.maurer@gmail.com; N. Megow, Zentrum für Mathematik, Technische Universität München, Boltzmannstr. 3, 85747 Garching bei München, Germany; email: nmegow@ma.tum.de; A. Wiese, Max Planck Institute for Informatics, Campus E1.4, 66123 Saarbrücken, Germany; email: awiese@mpi-inf.mpg.de.

1. INTRODUCTION

Competitive analysis [Sleator and Tarjan 1985; Karlin et al. 1988] 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 many 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 a lot 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. 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 *competitive-ratio 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 scheduling algorithms that allows us to reduce the infinite-size set of all 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. This is clearly in strong contrast to all previously given (lower) bounds that stem from *manually* designed input instances. We are aware of only very few online problems for which a competitive ratio, or even the optimal competitive ratio, are known to be *computable* by some algorithm (for a not inherently finite problem). 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 simply due to difficulties like the halting problem. We overcome these issues and pave the way for computer-assisted design of online algorithms.

We believe that our concept of abstraction for online algorithms can be applied successfully to other problems. We show this for other scheduling problems with jobs arriving online over time. We hope that our 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. Indeed, as we describe below, soon after the first publication of our results there has been follow-up work on competitive-ratio approximation schemes.

1.1. Problem Definition

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 $\mathsf{A}(\mathcal{I}) \leq \rho \cdot \mathsf{Opt}(\mathcal{I})$, 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 $\mathbb{E}[\mathsf{A}(\mathcal{I})] \leq \rho \cdot \mathsf{Opt}(\mathcal{I})$ 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 a competitive-ratio 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. A *competitive-ratio approximation scheme* computes for a given $\varepsilon > 0$ an online algorithm A with a competitive ratio $\rho_{\mathsf{A}} \leq (1+\varepsilon)\rho^*$. Moreover, it determines a value ρ' such that $\rho' \leq \rho^* \leq (1+\varepsilon)\rho'$.

Online scheduling. A scheduling instance consists of a fixed set of m machines and a set of jobs J, where 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 main problem considered in this article is to schedule the jobs on the given set of machines to minimize $\sum_{j\in J} w_j C_j$, where C_j denotes the completion time of job j. We consider the problem with and without preemption. Using standard scheduling notation [Graham et al. 1979], we denote the non-preemptive (preemptive) problems that we consider in this article by $\mathrm{Pm}|\,r_j, (\mathit{pmtn})\,|\sum w_j C_j$, $\mathrm{Qm}|\,r_j, (\mathit{pmtn})\,|\sum w_j C_j$, and $\mathrm{Rm}|\,r_j, \mathit{pmtn}\,|\sum w_j C_j$.

We also briefly consider more general min-sum objectives $\sum_{j\in J} w_j f(C_j)$, where f is an arbitrary monomial function $f(x) = k \cdot x^\alpha$, with constant $\alpha \geq 1, k > 0$, and the classical makespan $C_{\max} := \max_{j\in J} C_j$.

1.2. Previous Work

The offline variants of nearly all scheduling problems under consideration are NP-hard, but in most cases polynomial-time approximation schemes (PTAS) have been developed. The corresponding online settings have been a highly active field of research in the past 15 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. Interestingly, despite the considerable effort, *optimal* competitive ratios are hardly known while generally unsatisfactory, even quite significant gaps remain. We give more details on the state of the art for the different problem classes under consideration in the subsections below. Before we do so, we discuss other previous work on computing techniques for competitive ratios.

To the best of our knowledge, there are only very few problems in online optimization for which an optimal competitive ratio can be determined, bounded, or approximated by computational means. Lund and Reingold [1994] present a framework for upper-bounding the optimal competitive ratio of randomized algorithms by a linear program. For certain cases, for example, the two-server problem in a space of three points, this yields a provably optimal competitive ratio. Ebenlendr et al. [2009] and Ebenlendr and Sgall [2011] study various online and semi-online variants of scheduling preemptive jobs on uniformly related machines to minimize the makespan. In contrast to our model, they assume the jobs to be given one by one (rather than over time). They prove that the optimal competitive ratio can be computed by a linear program for any given set of speeds. In terms of approximating the best possible performance guarantee, the work by Augustine et al. [2008] is closest to ours. They show how to compute a nearly optimal power-down strategy for a processor with a finite number of power states.

Table I. Lower and Upper Bounds on the Competitive Ratio for Deterministic and Randomized Online Algorithms

	deterministic		randomized	
problem	lower bounds	upper bounds	lower bounds	upper bounds
$1\|r_j, pmtn\|\sum C_j$	1	1 [Schrage 1968]	1	1 [Schrage 1968]
$1\|r_j, pmtn\|\sum w_jC_j$	1.073 [Epstein and van Stee 2003]	1.566 [Sitters 2010a]	1.038 [Epstein and van Stee 2003]	4/3 [Schulz and Skutella 2002a]
$1\|r_j\|\sum C_j$	2 [Hoogeveen and Vestjens 1996]	2 [Hoogeveen and Vestjens 1996]	$\frac{e}{e-1} \approx 1.58$ [Stougie and Vestjens 2002]	$\frac{e}{e-1}$ [Chekuri et al. 2001]
$1\|r_j\|\sum w_jC_j$	2 [Hoogeveen and Vestjens 1996]	2 [Anderson and Potts 2004]	$\frac{e}{e-1}$ [Stougie and Vestjens 2002]	1.686 [Goemans et al. 2002]
$\mathrm{P}\|r_j, pmtn\|\sum C_j$	1.047 [Vestjens 1997]	5/4 [Sitters 2010b]	1	5/4 [Sitters 2010b]
$\mathrm{P}\|r_j, pmtn\|\sum w_jC_j$	1.047 [Vestjens 1997]	1.791 [Sitters 2010b]	1	1.791 [Sitters 2010b]
$\mathrm{P}\|r_j\|\sum w_jC_j$	1.309[1] [Vestjens 1997]	1.791 [Sitters 2010b]	1.157 [Seiden 2000]	1.791 [Sitters 2010b]
$\mathrm{R}\|r_j\|\sum w_jC_j$	1.309 [Vestjens 1997]	8 [Hall et al. 1997]	1.157 [Seiden 2000]	8 [Hall et al. 1997]

Sum of Weighted Completion Times. The offline variants of scheduling to minimize the total weighted completion time is NP-hard, even for the special case of a single machine [Labetoulle et al. 1984; Lenstra et al. 1977]. Two restricted variants can be solved optimally in polynomial time. *Smith's Rule* solves the problem $1| \,|\sum w_jC_j$ to optimality by scheduling jobs in non-increasing order of weight-to-processing-time ratios [Smith 1956]. Furthermore, scheduling by shortest remaining processing times yields an optimal schedule for $1|r_j, pmtn|\sum w_jC_j$ [Schrage 1968]. For other settings, polynomial-time approximation schemes have been developed, like for an arbitrary number of identical or a constant number of unrelated machines [Afrati et al. 1999]. For an arbitrary number of unrelated machines in the setting without release dates, very recently Bansal et al. [2016] gave a $(3/2-\delta)$-approximation algorithm for a small constant $\delta > 0$, improving on the previously known 3/2-approximation algorithms [Skutella 2001; Sethuraman and Squillante 1999]. If the jobs have release dates, then the best results for an arbitrary number of unrelated machines are a 2-approximation algorithm if preemption is not allowed [Skutella 2001] and a 3-approximation if preemption and migration are allowed [Skutella 2001].

The online setting has been a highly active field of research over the past 15 years [Goemans et al. 2002; Schulz and Skutella 2002a; Hall et al. 1997; Sitters 2010a, 2010b; Anderson and Potts 2004; Chekuri et al. 2001; Hoogeveen and Vestjens 1996; Correa and Wagner 2009; Goemans 1997; Megow and Schulz 2004; Megow 2007; Chung et al. 2010; Schulz and Skutella 2002b; Liu and Lu 2009; Lu et al. 2003; Stougie and Vestjens 2002; Phillips et al. 1998; Chakrabarti et al. 1996; Seiden 2000; Epstein and van Stee 2003]. We do not intend to give a detailed history of developments; instead, we refer the reader to overviews, for example, in Megow [2007] and Correa and Wagner [2009]. Table I summarizes the current state of the art on best-known lower and upper bounds on the optimal competitive ratios. Interestingly, despite the considerable effort, optimal competitive ratios are known only for $1|r_j, pmtn|\sum C_j$ [Schrage 1968] and for non-preemptive single-machine scheduling [Anderson and Potts 2004; Stougie and Vestjens 2002; Hoogeveen and Vestjens 1996; Chekuri et al. 2001].

More General Min-Sum (Completion Time) Objectives. Recently, there has been an increasing interest in studying generalized cost functions. So far, this research has focused on offline problems. The most general case is when each job may have its

[1]For $m = 1, 2, 3, 4, 5, \ldots, 100$ the lower bound is $LB = 2, 1.520, 1.414, 1.373, 1.364, \ldots, 1.312$.

individual non-decreasing cost function f_j. For scheduling on a single machine with release dates and preemption, $1|r_j, pmtn|\sum f_j$, Bansal and Pruhs [2010] gave a randomized $\mathcal{O}(\log\log(nP))$-approximation, where $P = \max_{j\in J} p_j$. In the case that all jobs have identical release dates, the approximation factor reduces to 16. Cheung and Shmoys [2011] improved this latter result. They give a deterministic primal-dual algorithm with an approximation factor of $4+\varepsilon$ [Cheung and Shmoys 2011; Mestre and Verschae 2014]. This result applies also on a machine of varying speed.

The more restricted problem with a global cost function $1|r_j, pmtn|\sum w_j f(C_j)$ has been studied by Epstein et al. [2012] in the context of universal solutions. They gave an algorithm that produces for any job instance one scheduling solution that is a $(4+\varepsilon)$ approximation for any cost function and even under unreliable machine behavior. Höhn and Jacobs [2012] studied the same problem without release dates. They analyzed the performance of Smith's Rule [Smith 1956] and gave tight approximation guarantees for all convex and all concave functions f.

Makespan. The online makespan minimization problem has been extensively studied in a different online paradigm where jobs arrive one by one (see Fleischer and Wahl [2000] and Rudin III and Chandrasekaran [2003] and references therein). Our model, in which jobs arrive online over time, is much less studied. In the identical parallel machine environment, Chen and Vestjens [1997] give nearly tight bounds on the optimal competitive ratio, $1.347 \leq \rho^* \leq 3/2$, using a natural online variant of the well-known largest processing time first algorithm.

In the offline setting, polynomial time approximation schemes are known for identical [Hochbaum and Shmoys 1987] and uniform machines [Hochbaum and Shmoys 1988]. For unrelated machines, the problem is NP-hard to approximate with a better ratio than 3/2 and a 2-approximation is known [Lenstra et al. 1990]. If the number of machines is bounded by a constant, then there is a PTAS [Lenstra et al. 1990].

1.3. New Results and Methodology

In this article, we introduce the concept of competitive-ratio approximation schemes and present such schemes for various scheduling problems with jobs arriving online over time. We present our technique focusing on the problems $\mathrm{Pm}|\,r_j, (pmtn)\,|\sum w_jC_j$, $\mathrm{Qm}|\,r_j, (pmtn)\,|\sum w_jC_j$ (assuming a constant range of machine speeds without preemption), and $\mathrm{Rm}|\,r_j, pmtn\,|\sum w_jC_j$, and we comment on how it applies to other cost functions such as the makespan, $C_{\max}$, and $\sum_{j\in J} w_j f(C_j)$, where f is an arbitrary monomial function with fixed exponent. 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 the randomized setting for any number of machines m. Moreover, we give an algorithm that estimates the optimal competitive ratio for these problems to any desired accuracy. Thus, we reduce algorithmically the performance gaps for all considered problems to an arbitrarily small value. 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 new and unusual way of designing online scheduling algorithms. We present an 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 that 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, for example, Afrati et al. [1999]. 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 (see also Manasse et al. [1988] for an enumeration routine for online algorithms for a fixed task system with finitely many states). For randomized algorithms we even show that we can restrict ourselves to instances with only constantly many jobs. As all our structural insights also apply to offline algorithms for the same problems, 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 with 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, for example, a computer. In general, there are no computational means to determine the competitive ratio of an algorithm—even when it is a constant. 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 article, 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 value.

Although the enumeration scheme for identifying the (nearly) optimal online algorithm 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.

1.4. 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 $\mathrm{Pm}|\,r_j, \mathit{pmtn}\,|\sum w_jC_j$. Based on this, we present our abstraction of online algorithms and develop a competitive-ratio approximation scheme in Section 3. Next, we sketch in Section 4 how to extend these techniques to the non-preemptive setting and more general machine environments such that the approximation scheme (Section 3) remains applicable. In Section 5, we present competitive-ratio approximation schemes for the randomized setting. Finally, in Section 6, we extend our results to other objective functions.

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 $\mathrm{Pm}|\,r_j, \mathit{pmtn}\,|\sum w_jC_j$. Later, we outline how to adapt these for more general settings. Our construction combines several transformation techniques known for offline PTASs (see Afrati et al. [1999] and the references therein) and a new technique to subdivide an instance online into parts that can be handled separately.

We will use the terminology that *at* $1 + O(\varepsilon)$ *loss we can restrict* ourselves 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, then any mentioned constant depends only on ε and m.

To begin with, we use the standard simplification technique of *geometric rounding* introduced in Afrati et al. [1999]. For the sake of completeness, we prove the required properties in the following lemma.

LEMMA 2.1. *At $1+O(\varepsilon)$ loss we can restrict ourselves 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.*

PROOF. We prove that any given schedule can be adapted at $1+O(\varepsilon)$ loss such that the required properties can be assumed for the corresponding instance. Obviously, weights can be rounded up to the next power of $1+\varepsilon$ by increasing the cost of a given schedule by at most a factor $1+\varepsilon$.

Within a given preemptive schedule, the job volume of each job j is assigned to a set of time intervals on machines (such that their lengths sum up to p_j). Multiplying the boundary values of each interval by $(1+\varepsilon)$ results in time intervals assigned to each j with a total length of $(1+\varepsilon)p_j$. Hence, we get a feasible schedule even when rounding up all processing times to the next powers of $(1+\varepsilon)$. The total completion time does not increase by more than a factor of $(1+\varepsilon)$.

Consider now a schedule with rounded processing times. We again multiply each boundary value of the intervals by $(1+\varepsilon)$ and shift the processing volume of each job j to the latest possible time intervals within its assigned intervals. Then, job j completes at time $(1+\varepsilon)C_j$, and the earliest point in time at which j is processed increases from S_j to $S_j+\varepsilon p_j$. Hence, by losing at most a factor $(1+\varepsilon)$, we may assume that each job j has a release date $r_j \geq \varepsilon p_j$. If necessary, then the parameters of all jobs can be scaled by some power of $(1+\varepsilon)$ such that the earliest release date is at least one (since jobs with $r_j = p_j = 0$ can be ignored).

With a similar reasoning, we can finally round at a loss of $(1+\varepsilon)$ each release date to the next power of $(1+\varepsilon)$. □

The geometric rounding procedure allows us to see intervals of the form $I_x := [R_x, R_{x+1})$ with $R_x := (1+\varepsilon)^x$ as atomic entities. Note that $|I_x| = \varepsilon \cdot R_x$. An online algorithm can define the corresponding schedule at the beginning of an interval since no further jobs are released until the next interval. Moreover, we make, at $1+\varepsilon$ loss, the simplifying assumption that each job j finishing within I_x contributes $w_j \cdot R_{x+1}$ to the objective function, that is, we pretend that j finishes exactly at time R_{x+1}.

2.1. Simplification Within Intervals

The goal of this section is to reduce the number of situations that can arise at the beginning of an interval. For each ε, we identify constantly many relevant inputs per interval. And for a given input, we reduce the number of relevant algorithmic actions within each interval to a constant.

To do so, we use the techniques of *time-stretching* and partitioning jobs released at time R_x into *large and small* jobs, see Afrati et al. [1999]. In an online interpretation of time-stretching, we shift the work assigned to any interval I_x to the interval I_{x+1}. When doing this operation once we speak of *one time-stretch*. This can be done at a loss of $1+\varepsilon$ and we obtain free space of size $\varepsilon \cdot |I_x|$ in each interval I_{x+1}. For each $x \geq 0$, we define $L_x = \{j \in J \mid r_j = R_x, p_j > \varepsilon^2|I_x|\}$ to be the set of *large* jobs released at R_x, and we let $S_x = \{j \in J \mid r_j = R_x, p_j \leq \varepsilon^2|I_x|\}$ be the set of *small* jobs released at R_x.

We first take care of the small jobs. Since they are small, there is a lot of flexibility in scheduling them. We show that we can, at a small loss, fix in advance the order in which the jobs in each S_x are processed. This enables us to group very small jobs to job *packs*. Treating each pack as a single job, we then get a lower bound on the processing time of the released jobs.

We denote w_j/p_j as the *Smith ratio* of a job j. A non-increasing ordering by the Smith ratios is an ordering according to *Smith's rule* [Smith 1956]. We say that a job is *partially processed* at some point in time if it has been processed but not yet completed. For a set of jobs J, we define $p(J) := \sum_{j\in J} p_j$ and $w(J) := \sum_{j\in J} w_j$.

LEMMA 2.2. *At $1+\varepsilon$ loss, we can restrict ourselves 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 $S_{\leq x} := \bigcup_{x'\leq x} S_{x'}$ and no small job is preempted or only partially processed at the end of an interval. Therefore, we can restrict ourselves to instances with $p(S_x) \leq m\cdot |I_x|$ for each interval I_x.*

PROOF. Consider some schedule S and apply one time-stretch. The resulting schedule S' has a free space of $\varepsilon|I_x|$ units in each interval I_{x+1}, and its total cost is within a factor $(1+\varepsilon)$ of the cost of S. Remove now all small jobs from each interval I_{x+1}. For each x and each job j we denote by $q_{x,j}$ the amount of processing time of j assigned by S' to interval I_x. Furthermore, let Q_x be the sum of $q_{x,j}$ over all small jobs and Q'_x the respective sum over all large jobs. By A_x we denote the set of all removed small jobs completed by S' in I_x. Now, we apply for each $x \geq 0$ the following procedure. First, we remove all large jobs from interval I_x and reschedule them machinewise on the first $|I_{x-1}|$ units of processing time; that is, we fill the machines consecutively and with preemption until an amount of $q_{x,j}$ is assigned for each large job j. Notice that, due to the time-stretch, each large job has at most size $q_{x,j} \leq |I_x|/(1+\varepsilon) = |I_{x-1}|$ and $Q'_x \leq m\cdot|I_{x-1}|$. Thus, the procedure provides a feasible schedule for all large jobs, and we do not increase the total cost. This scheduling policy is also known as McNaughton's wrap-around rule [McNaughton 1959]. We now have a number of $k \geq 0$ machines in I_x whose first $|I_{x-1}|$ units of processing time are completely filled and $m-k$ machines where this is not the case.

Now, we fill unscheduled small jobs non-preemptively into the idle time of each machine $i = k+1, \ldots, m$. To that end, we consider unscheduled jobs from $S_{\leq x}$ in the order of their Smith ratios and schedule them without preemption until an amount of Q_x is assigned or no further small job is available. We fill the machines one after the other and switch whenever a machine has received $|I_{x-1}|$ units of processing. Again, this is possible since $Q_x + Q'_x \leq m\cdot|I_{x-1}|$. The procedure increases the processing time on the latter machines by at most $\varepsilon^2|I_x|$, the maximum size of small jobs, which is smaller than the created extra space of $\varepsilon|I_{x-1}|$ for sufficiently small ε. Denote by B_x the set of small jobs assigned to I_x in this step.

Inductively, we can prove that for each I_x the total weight of jobs from $S_{\leq x}$ that are completed before the end of I_x has not decreased compared to S'. For the first interval I_{x_1} with $Q_{x_1} > 0$, it holds that $w(A_{x_1}) \leq w(B_{x_1})$ since the jobs of B_{x_1} are assigned according to Smith's rule and $p(B_{x_1}) \geq Q_{x_1} \geq p(A_{x_1})$. This proves the base case since each job of B_{x_1} is completed before the end of I_{x_1}. For the induction step we use that $w(A_{\leq x}) \leq w(B_{\leq x})$ with $A_{\leq x} := \bigcup_{x'\leq x} A_{x'}$ and $B_{\leq x}$, respectively, for some x. Assume by contradiction $w(A_{\leq\tilde{x}}) > w(B_{\leq\tilde{x}})$ for the next $\tilde{x} > x$ with $Q_{\tilde{x}} > 0$. This implies that

$$w(A_{\tilde{x}}) > w(B_{\leq x}) - w(A_{\leq x}) + w(B_{\tilde{x}}).$$

Intuitively, the total weight of jobs in $A_{\tilde{x}}$ is higher than the surplus weight gained in the earlier intervals (being non-negative) together with the total weight of $B_{\tilde{x}}$. This contradicts the fact that we assigned for each $x' \leq \tilde{x}$ jobs with a total processing time of $p(B_{\leq x'}) \geq p(A_{\leq x'})$ via Smith's rule. Hence, we get $w(A_{\leq\tilde{x}}) \leq w(B_{\leq\tilde{x}})$. The induction hypothesis follows since each job in $B_{\tilde{x}}$ is completed before the end of $I_{\tilde{x}}$. Recall that we modified the objective function by rounding up the completion times of jobs to the end of the respective interval. Consequently, the observation on the completed weights before each interval end yields that the objective value is not increased by the described reassignment. Furthermore, the procedure ensures that no small job is preempted and that any small job finishes in the same interval where it started.

Since we can now assume that the jobs of each S_x are chosen to be scheduled within I_x in non-increasing order of their Smith ratios (completely and without preemption), we can conclude the last claim of the lemma. Note that the total processing time in interval I_x is $m|I_x|$. Pick the jobs of S_x in the respective order until the total processing time of picked jobs just exceeds $m|I_x|$. The remaining jobs of S_x cannot be scheduled within I_x, and, hence, by the above argument, we can safely move their release dates to R_{x+1}. $\square$

LEMMA 2.3. *At $1+O(\varepsilon)$ loss, we can restrict ourselves to instances such that $p_j \geq \frac{\varepsilon^2}{4} \cdot |I_x|$ for each job $j \in S_x$. In these instances, the number of distinct processing times of each set S_x is bounded from above by* $\log_{1+\varepsilon} 4$.

PROOF. We call a job $j \in S_x$ *tiny* if $p_j \leq \frac{\varepsilon^2}{4} \cdot |I_x|$. Let $T_x = \{j_1, j_2, \ldots, j_{|T_x|}\}$ denote all tiny jobs released at R_x. W.l.o.g. assume that they are ordered non-increasingly by their Smith ratios w_j/p_j. Let ℓ be the largest integer such that $\sum_{i=1}^{\ell} p_i \leq \frac{\varepsilon^2}{2} \cdot |I_x|$. We define the pack $P_x^1 := \{j_1, \ldots, j_\ell\}$. We define the processing time of pack P_x^1 to be $\sum_{i=1}^{\ell} p_i$ and its weight to be $\sum_{i=1}^{\ell} w_i$. 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 $[\frac{\varepsilon^2}{4} \cdot |I_x|, \frac{\varepsilon^2}{2} \cdot |I_x|]$.

We apply one time-stretch. In any schedule that assigns small jobs according to Smith's rule to intervals there is for each I_x at most one partially processed job pack per machine from each of the previous release dates $R_{x'} < R_x$. Since $\sum_{x'<x} \varepsilon^2 |I_{x'}| \leq \varepsilon |I_x|$, we can schedule all of them in the newly created space. This also includes space to increase the processing time of the very last pack of each $S_{x'}$ to $\frac{\varepsilon^2}{4} \cdot |I_{x'}|$, if necessary. Therefore, we can enforce that at $1 + O(\varepsilon)$ loss all tiny jobs of the same pack are scheduled in the same interval on the same machine. Hence, we can treat each pack as a single job whose processing time and weight matches the respective values of the pack.

Finally, at $1 + O(\varepsilon)$ loss we can ensure that the processing times and weights of the new jobs (which still remain small) are powers of $1 + \varepsilon$. Consequently, the processing times of jobs in S_x are of the form $(1 + \varepsilon)^y$ within the following range:

$$\frac{\varepsilon^3}{4} \cdot (1+\varepsilon)^x \leq (1+\varepsilon)^y \leq \varepsilon^2 |I_x| = \varepsilon^3 (1+\varepsilon)^x.$$

The number of integers y satisfying these inequalities is bounded from above by $\log_{1+\varepsilon} 4$. $\square$

For the large jobs released at the beginning of an interval we obtain an upper bound on their length by their relation of release date and processing time (Lemma 2.1). This induces a constant upper bound on the number of occurring processing times.

LEMMA 2.4. *The number of distinct processing times of jobs in each set L_x is bounded from above by* $4 \log_{1+\varepsilon} \frac{1}{\varepsilon}$.

PROOF. Let $j \in L_x$ be a large job released at R_x with processing time $p_j = (1 + \varepsilon)^y > \varepsilon^2 |I_x|$ for some integer y. By Lemma 2.1, we know that $p_j \leq \frac{1}{\varepsilon} r_j = \frac{1}{\varepsilon}(1+\varepsilon)^x$ and, hence,

$$\varepsilon^3 (1+\varepsilon)^x = \varepsilon^2 |I_x| < (1+\varepsilon)^y \leq \frac{1}{\varepsilon} (1+\varepsilon)^x .$$

The number of integers y that satisfy the above inequalities is upper bounded by the constant claimed in the lemma. $\square$

We say that two large jobs are of the same *type* if they have the same processing time and the same release date. By an exchange argument, we can restrict ourselves without any loss to schedules in which at each point in time at most m large jobs of each type are partially scheduled. Since the amount of work that can be processed within each interval is bounded, the number of large jobs of the same type can also be bounded.

LEMMA 2.5. *Without loss, we can restrict ourselves 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, that is, they are of the same type. Since $p_j > \varepsilon^2|I_x|$ for each job $j \in L_x$, at most $m/\varepsilon^2 + m$ jobs in $L_{x,p}$ can be started before I_{x+1}. By an exchange argument, we can assume that they are among 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. By Lemma 2.4 there are at most $4\log_{1+\varepsilon}\frac{1}{\varepsilon}$ distinct processing times p of large jobs in L_x, and, thus, the claim follows. $\square$

As we simplified the objective function by pretending all jobs to complete at the end of an interval, the only information needed for computing the objective function value is the interval in which a job completes. The only part that has not been bounded yet is the amount that large jobs are processed within the single intervals.

LEMMA 2.6. *There is a constant $\mu \in \mathbb{N}$ such that at $1 + O(\varepsilon)$ loss we can restrict ourselves to schedules such that at the end of each interval, each large job j is processed to an extent which is an integer multiple of p_j/μ.*

PROOF. We apply one time-stretch and choose $\mu \in \mathbb{N}$ to be a constant such that $1/\mu$ is smaller than $\varepsilon^4/(8\log_{1+\varepsilon}\frac{1}{\varepsilon})$. Consider now an interval I_x and the jobs that are scheduled in I_x before the time-stretch. For each job $j \in L_{x'}$, $x' \leq x$ that was partially processed at time R_{x+1} we must extend the amount of time j is processed within I_{x+1} by at most p_j/μ to achieve the stated property. Using Lemma 2.1, this value can be bounded by

$$\frac{p_j}{\mu} \leq \frac{R_{x'}}{\varepsilon} \cdot \frac{\varepsilon^4}{8\log_{1+\varepsilon}\frac{1}{\varepsilon}} = \frac{\varepsilon^2|I_{x'}|}{2 \cdot 4\log_{1+\varepsilon}\frac{1}{\varepsilon}}.$$

Recall that we can assume without any loss that at the end of each interval at most one large job per job type is partially processed on each machine. Since the number of job types of $L_{x'}$ is bounded by $4\log_{1+\varepsilon}\frac{1}{\varepsilon}$ (Lemma 2.4), the space of $\varepsilon^2|I_{x'}|/2$ is sufficient to handle each job type of $L_{x'}$. Applying $\sum_{x'<x}\varepsilon^2|I_{x'}| \leq \varepsilon|I_x|$, we see that this amount of free space was created by the time-stretch. $\square$

To summarize, we get the following simplifications within intervals concerning input and scheduling decisions. Recall that we can calculate the value of μ that depends only on ε. Furthermore, we define Δ to be a constant upper bound on the number of released jobs in each interval, that is, Δ is an integer of at most $\lceil \frac{4m}{\varepsilon^2} + 4\log_{1+\varepsilon}\frac{1}{\varepsilon}(\frac{m}{\varepsilon^2} + m)\rceil$.

COROLLARY 2.7. *At $1 + O(\varepsilon)$ loss we can assume that for each interval I_x*

(*a*) *each job j released at time R_x has a processing time $p_j = (1+\varepsilon)^k \in [\frac{\varepsilon^3}{4}R_x, \frac{1}{\varepsilon}R_x]$ for some integer k,*
(*b*) *there are at most $\log_{1+\varepsilon}(4/\varepsilon^4)$ distinct processing time values of jobs released at R_x,*

(*c*) *at most* Δ *jobs are released at* R_x,
(*d*) *each small job starting in* I_x *completes in* I_x *without preemption, and*
(*e*) *at the end of* I_x, *each job* j *is processed to an extent of* $\ell_{x,j} \cdot p_j/\mu$ *for some* $\ell_{x,j} \in \{0, \dots, \mu\}$.

2.2. Irrelevant History

The schedule for an interval returned by an online algorithm may depend on the set of currently unfinished jobs and possibly the entire schedule computed 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 when taking its decisions, namely, the jobs with relatively large weight released in the last constantly many intervals.

First, we show that we may assume that each job completes within constantly many intervals after its release.

LEMMA 2.8. *There is a constant* s *such that at* $1 + O(\varepsilon)$ *loss we can restrict ourselves to schedules such that for each interval* I_x *there is a subinterval of* I_{x+s-1} *that is large enough to process all jobs released at* R_x *and during which only those jobs 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 using Lemmas 2.1, 2.2, and 2.5 we get

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

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 $p(S_x) + p(L_x)$. □

Given the bound on the number of intervals between release and completion times of jobs, we partition the time horizon into periods such that no job is "alive" for more than two periods. For each integer $k \ge 0$, we define *period* Q_k to consist of the s consecutive intervals $I_{k \cdot s}, \dots, I_{(k+1)\cdot s-1}$. We add an artificial period Q_{-1} for the interval $[0, 1)$ in which no job is released. Hence, we can assume by Lemma 2.8 that each job released in period Q_k is completed by the end of period Q_{k+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 value in *any* schedule. Due to Lemma 2.8, we also obtain an upper bound of $(1+\varepsilon)^s \cdot rw(J)$ for the latter quantity.

We will now determine at the end of each period how important its released jobs are compared to the previous periods. If the job weights released in a series of periods grow large enough from period to period, then we will see that the overall objective value is dominated by the contribution of the constantly many last periods. If otherwise the job weights of a period are too low compared to the preceding ones and its jobs can be moved to their safety net with a small loss, then we will see that the following periods can be treated independently. To this end, we define that $p > 0$ consecutive periods $Q_k, \dots, Q_{k+p-1}$ are a *sequence of significant periods* if $rw(Q_{k+\ell}) > \frac{\varepsilon}{(1+\varepsilon)^s} \cdot \sum_{i=0}^{\ell-1} rw(Q_{k+i})$ for each $\ell = 1, \dots, p-1$. This implies exponential growth for the series of partial sums of release weights, and we can prove the dominance of a few of the last periods.

LEMMA 2.9. *There is a constant K such that for each sequence $Q_k, Q_{k+1}, \dots, Q_{k+p-1}$ of significant periods we have that*

$$\sum_{i=0}^{p-K-1} (1+\varepsilon)^s rw(Q_{k+i}) \leq \varepsilon \cdot \sum_{i=p-K}^{p-1} rw(Q_{k+i}).$$

PROOF. Let $\delta := \frac{\varepsilon}{(1+\varepsilon)^s}$. Since we consider a sequence of significant periods, we get

$$rw(Q_{k+\ell}) > \delta \cdot \sum_{i=0}^{\ell-1} rw(Q_{k+i}) \qquad \forall \ell = 1, \dots, p-1.$$

This implies that

$$(1+\delta) rw(Q_{k+\ell}) > \delta \cdot \sum_{i=0}^{\ell} rw(Q_{k+i}) \qquad \forall \ell = 1, \dots, p-1.$$

Hence, we get

$$\frac{\sum_{i=0}^{\ell-1} rw(Q_{k+i})}{\sum_{i=0}^{\ell} rw(Q_{k+i})} = 1 - \frac{rw(Q_{k+\ell})}{\sum_{i=0}^{\ell} rw(Q_{k+i})} < 1 - \frac{\delta}{1+\delta} = \frac{1}{1+\delta} < 1 \quad \forall \ell = 1, \dots, p-1,$$

which implies

$$\sum_{i=0}^{\ell-1} rw(Q_{k+i}) < \frac{1}{1+\delta} \sum_{i=0}^{\ell} rw(Q_{k+i}) \qquad \forall \ell = 1, \dots, p-1. \tag{2.1}$$

In other words, if we remove $Q_{k+\ell}$ from $\cup_{i=0}^{\ell} Q_{k+i}$, the total release weight of the set decreases by a factor of at least $1/(1+\delta) < 1$. Recursively applying Equation (2.1), we get for any K

$$\begin{aligned}\sum_{i=0}^{p-1-K} rw(Q_{k+i}) &< \left(\frac{1}{1+\delta}\right)^K \sum_{i=0}^{p-1} rw(Q_{k+i}) \\ &= \frac{1}{(1+\delta)^K} \left(\sum_{i=0}^{p-K-1} rw(Q_{k+i}) + \sum_{i=p-K}^{p-1} rw(Q_{k+i}) \right)\end{aligned}$$

and hence

$$\left(1 - \left(\frac{1}{(1+\delta)^K}\right)\right) \sum_{i=0}^{p-K-1} rw(Q_{k+i}) < \frac{1}{(1+\delta)^K} \sum_{i=p-K}^{p-1} rw(Q_{k+i}).$$

To ensure that $(1+\varepsilon)^s \sum_{i=0}^{p-K-1} rw(Q_{k+i}) < \varepsilon \cdot \sum_{i=p-K}^{p-1} rw(Q_{k+i})$ we choose K sufficiently large such that

$$\frac{1}{1 - \frac{1}{(1+\delta)^K}} \cdot \frac{1}{(1+\delta)^K} = \frac{1}{(1+\delta)^K - 1} < \delta = \frac{\varepsilon}{(1+\varepsilon)^s}.$$

Hence, K is some constant integer larger than $\log_{1+\delta}(1/\delta + 1)$ and depends only on ε. □

Using the safety net (Lemma 2.8) the above lemma implies that an ε-fraction of the weighted completion time of the last $K-1$ periods of a sequence of significant periods yields an upper bound on the weighted completion time of the previous periods of this sequence. Therefore, the objective value is essentially dominated by the contribution of the last $K-1$ periods. We will need this later to show that at $1+O(\varepsilon)$ loss we can assume that an online algorithm bases its decisions only on a constant amount of information.

To that end, in the following we consider a sequence of significant periods $Q_k, \ldots, Q_{k+p-1}$ and an interval I_x of period Q_{k+p} where the newly released jobs are just revealed to an online algorithm. Recall that the online algorithm does not know the release weight of the current period Q_{k+p} unless I_x is the last interval of Q_{k+p}. Nevertheless, we know by Lemma 2.9 that the costs of $Q_k, \ldots, Q_{k+p-1}$ are dominated by the last $K-1$ periods. Hence, the costs until interval I_x are dominated by the last important $\Gamma := Ks$ intervals including I_x. In addition to the jobs that have been released very early, also the jobs with very small weight in comparison to at least one other job can be almost neglected for the total costs. Therefore, we partition the jobs into relevant and irrelevant jobs using these two criteria.

Definition 2.10. Let J be a set of jobs. A job $j \in J$ with $r_j \leq R_x$ is called *recent at time* R_x if $R_{x-\Gamma} \leq r_j$. Otherwise, it is called *old at time* R_x. Job j is furthermore called *dominated at time* R_x if it is dominated at time R_{x-1} or if there is a job $j' \in J$ being recent and not dominated at time R_x such that $w_j < \frac{\varepsilon}{\Delta \cdot \Gamma \cdot (1+\varepsilon)^{\Gamma+s}} w_{j'}$. Now, job j is *irrelevant at time* R_x if it is old or dominated at time R_x and otherwise *relevant at time* R_x. Denote the respective subsets of a job set J by $\mathrm{Rec}_x(J)$, $\mathrm{Old}_x(J)$, $\mathrm{Dom}_x(J)$, $\mathrm{Ir}_x(J)$, and $\mathrm{Rel}_x(J)$.

The subsequent lemma states that the irrelevant jobs can almost be ignored for the objective value of a schedule even when scheduled in their safety nets. This implies that we can restrict ourselves at $1+O(\varepsilon)$ loss to online algorithms that schedule the remaining part of a job in its safety net, once it has become irrelevant.

LEMMA 2.11. *Let $Q_k, \ldots, Q_{k+p-1}$ be a sequence of significant periods and let J be the jobs of $Q_k, \ldots, Q_{k+p}$ released until the beginning of an interval $I_x \in Q_{k+p}$. Then*

$$(1+\varepsilon)^s rw(\mathrm{Ir}_x(J)) \leq 6\varepsilon \cdot rw(\mathrm{Rel}_x(J)).$$

PROOF. By definition, an irrelevant job at time R_x is either old or dominated. Hence, $\mathrm{Ir}_x(J)$ is the disjoint union of $\mathrm{Old}_x(J)$ and $\mathrm{Dom}_x(J) \cap \mathrm{Rec}_x(J)$.

We start by giving a bound on the recent but dominated jobs and define $rw_{\max} := \max\{rw(j) \mid j \in \mathrm{Rel}_x(J) \cup \mathrm{Old}_x(J)\}$. Consider a job $j \in J$ that is dominated and recent at time R_x. By definition, there is a time $r_j \leq R_{x'} \leq R_x$ at which another job j' that is relevant at time $R_{x'}$ dominates j. Choose $R_{x'}$ to be as late as possible. As j' is recent at time $R_{x'}$, we get by $R_{x'}(1+\varepsilon)^{-\Gamma} \leq r_{j'}$ that $r_j \leq R_{x'} \leq (1+\varepsilon)^{\Gamma} r_{j'}$. Because we chose $R_{x'}$ to be as late as possible, we can assume that j' is not dominated at time R_x. Hence, j' is either relevant or old at time R_x. Therefore, there is for each $j \in \mathrm{Dom}_x(J) \cap \mathrm{Rec}_x(J)$ a job j' in $\mathrm{Rel}_x(J) \cup \mathrm{Old}_x(J)$ such that

$$\begin{aligned} w_j r_j &\leq \frac{\varepsilon}{\Delta \cdot \Gamma \cdot (1+\varepsilon)^{\Gamma+s}} w_{j'} r_j \\ &\leq \frac{\varepsilon}{\Delta \cdot \Gamma \cdot (1+\varepsilon)^{\Gamma+s}} w_{j'} r_{j'} (1+\varepsilon)^{\Gamma} \\ &\leq \frac{\varepsilon}{\Delta \cdot \Gamma \cdot (1+\varepsilon)^{s}} rw_{\max}. \end{aligned}$$

Since at most Δ jobs are released at the beginning of each interval (Corollary 2.7) and $rw_{\max}$ is the release weight of a job in $\mathrm{Rel}_x(J) \cup \mathrm{Old}_x(J)$ we get:

$$\begin{aligned}(1+\varepsilon)^s rw(\mathrm{Dom}_x(J) \cap \mathrm{Rec}_x(J)) &= (1+\varepsilon)^s \sum_{j \in \mathrm{Dom}_x(J) \cap \mathrm{Rec}_x(J)} w_j r_j \\ &\le (1+\varepsilon)^s \Delta \cdot \Gamma \frac{\varepsilon}{\Delta \cdot \Gamma \cdot (1+\varepsilon)^s} rw_{\max} \\ &= \varepsilon \cdot rw_{\max} \\ &\le \varepsilon \left(rw(\mathrm{Rel}_x(J)) + rw(\mathrm{Old}_x(J)) \right).\end{aligned}$$

Moreover, Lemma 2.9 implies for the old jobs at time R_x that

$$\begin{aligned}(1+\varepsilon)^s rw(\mathrm{Old}_x(J)) &\le \varepsilon \cdot rw(\mathrm{Rec}_x(J)) \\ &= \varepsilon \cdot \left(rw(\mathrm{Rel}_x(J)) + rw(\mathrm{Dom}_x(J) \cap \mathrm{Rec}_x(J)) \right).\end{aligned}$$

Combining all these arguments yields

$$\begin{aligned}(1+\varepsilon)^s rw(\mathrm{Ir}_x(J)) &= (1+\varepsilon)^s \left(rw(\mathrm{Old}_x(J)) + rw(\mathrm{Dom}_x(J) \cap \mathrm{Rec}_x(J)) \right) \\ &\le \varepsilon \cdot rw(\mathrm{Rel}_x(J)) + 2\varepsilon \cdot \left(rw(\mathrm{Rel}_x(J)) + rw(\mathrm{Old}_x(J)) \right) \\ &\le 3\varepsilon \cdot rw(\mathrm{Rel}_x(J)) + 2\varepsilon \cdot rw(\mathrm{Ir}_x(J)).\end{aligned}$$

With $\varepsilon < \frac{1}{3}$ we obtain the bound $(1+\varepsilon)^s rw(\mathrm{Ir}_x(J)) \le 6\varepsilon \cdot rw(\mathrm{Rel}_x(J))$. □

With the preceding lemmas, we have identified for each point in time a subset of jobs that is relevant at this time—under the assumption that we consider a sequence of significant periods $Q_k, \dots, Q_{k+p-1}$. It now remains to handle the case that the following period Q_{k+p} is not significant. In this case, we know that $(1+\varepsilon)^s rw(Q_{k+p}) \le \varepsilon \cdot \sum_{i=0}^{p-1} rw(Q_{k+i})$. Hence, completing all unfinished jobs of Q_{k+p} in their safety nets costs only an ε-fraction of the costs caused by the preceding significant periods. And since an online algorithm is allowed to preempt these jobs, it can now schedule the succeeding periods independently. However, in the following we will state a more general result that will be useful for eventually proving that online algorithms can forget all irrelevant jobs.

We define the consecutive periods $Q_k, \dots, Q_{k+p}$, for some $p > 0$, to be a *part* if $Q_k, \dots, Q_{k+p-1}$ is a sequence of significant periods and $(1+\varepsilon)^s rw(Q_{k+p}) \le 8\varepsilon \cdot \sum_{i=0}^{p-1} rw(Q_{k+i})$. We then call Q_{k+p} to be *insignificant* (even though it is possible that $Q_k, \dots, Q_{k+p}$ is a sequence of significant periods). We now consider a partitioning of a given instance $\mathcal{I}$ into parts $P_i, i = 0, \dots, \ell$, and denote the insignificant period of each part P_i by $Q_{a_{(i+1)}}$. With $a_0 := -1$ each part P_i consists of the periods $Q_{a_i+1}, \dots, Q_{a_{(i+1)}}, i = 0, \dots, \ell$. Again, we identify with P_i all jobs released in this part.

With the possibility to preempt jobs, we now treat each part P_i of a partition 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. By the following lemma, it then suffices to bound $\mathsf{A}(P_i)/\mathsf{Opt}(P_i)$ for each part P_i.

LEMMA 2.12. *At $1 + O(\varepsilon)$ loss, we can restrict ourselves to instances that consist of only one part.*

PROOF. Consider an online algorithm A, some instance $\mathcal{I}$ and a partition of $\mathcal{I}$ into parts $P_i, i = 0, \dots, \ell - 1$. We define an adapted version A' that applies $\mathsf{A}(P_i)$ to each part P_i and moves all jobs of P_i that are unfinished at the end of P_i to their respective safety nets. Recall that due to Lemma 2.8 these unfinished jobs must be released within the last period $Q_{a_{(i+1)}}$ of part P_i and this period is insignificant. Hence, these jobs contribute at most

$$\sum_{i=1}^{\ell+1}(1+\varepsilon)^s rw(Q_{a_i}) \leq \sum_{i=1}^{\ell+1} 8\varepsilon \cdot \sum_{p=a_{i-1}+1}^{a_i-1} rw(Q_p) \leq \sum_{i=0}^{\ell} O(\varepsilon)\cdot \mathsf{Opt}(P_i)$$

to the objective value. Therefore, we get that $\mathsf{A}'(\mathcal{I}) \leq (1+O(\varepsilon))\sum_{i=0}^{\ell}\mathsf{A}(P_i)$. Applying $\sum_{i=0}^{\ell}\mathsf{Opt}(P_i) \leq \mathsf{Opt}(\mathcal{I})$ we can bound the competitive ratio of A' as follows:

$$\frac{\mathsf{A}'(\mathcal{I})}{\mathsf{Opt}(\mathcal{I})} \leq (1+O(\varepsilon))\frac{\sum_{i=0}^{\ell}\mathsf{A}(P_i)}{\sum_{i=0}^{\ell}\mathsf{Opt}(P_i)} \leq (1+O(\varepsilon))\max_{i=1,\dots,\ell}\frac{\mathsf{A}(P_i)}{\mathsf{Opt}(P_i)}. \quad \square$$

Note that there might be different partitions of one instance into parts since it is possible that a part is at the same time a sequence of significant periods. In the following section, we will consider online algorithms that work in principle only on relevant jobs while forgetting irrelevant jobs after they have been moved to their safety net. Hence, we need a partition into parts that can be determined regardless of the irrelevant jobs. To that end, we define $x(a) := (a+1)s-1$ for a period Q_a to be the index of the last interval of period Q_a.

LEMMA 2.13. *For a given instance $\mathcal{I}$, let $a_1 < \ldots < a_{\ell+1}$ be all indices such that*

$$(1+\varepsilon)^s rw(\mathrm{Rel}_{x(a_i)}(P_{i-1})\cap Q_{a_i}) \leq \varepsilon\cdot\left(1+\frac{6\varepsilon}{(1+\varepsilon)^s}\right)\sum_{p=a_{i-1}+1}^{a_i-1} rw(\mathrm{Rel}_{x(a_i)}(P_{i-1})\cap Q_p) \tag{2.2}$$

and $a_0 := -1$ where each P_{i-1} consists of all periods $Q_{a_{(i-1)}+1},\ldots,Q_{a_i}$ for $i=1,\ldots,\ell+1$. Then, each P_i is a part of $\mathcal{I}$ for $i=0,\ldots,\ell$.

PROOF. We first prove that the periods between Q_{a_i} and $Q_{a_{(i+1)}}$ build a sequence of significant periods. To that end, we can inductively apply Lemma 2.11 and get for each $a_{i-1} < a < a_i$:

$$\begin{aligned}(1+\varepsilon)^s rw(Q_a) &\geq (1+\varepsilon)^s rw(\mathrm{Rel}_{x(a)}(P_{i-1})\cap Q_a)\\ &\overset{(2.2)}{>} \varepsilon\cdot\left(1+\frac{6\varepsilon}{(1+\varepsilon)^s}\right)\sum_{p=a_{i-1}+1}^{a-1} rw(\mathrm{Rel}_{x(a)}(P_{i-1})\cap Q_p)\\ &= \varepsilon\cdot\left(1+\frac{6\varepsilon}{(1+\varepsilon)^s}\right) rw\left(\mathrm{Rel}_{x(a)}(P_{i-1})\cap \bigcup_{p=a_{i-1}+1}^{a-1} Q_p\right)\\ &\overset{2.11}{\geq} \varepsilon\cdot\sum_{p=a_{i-1}+1}^{a-1} rw(Q_p).\end{aligned}$$

That is why we can now apply Lemma 2.11 to each complete P_{i-1} and get that each Q_{a_i} is insignificant:

$$
\begin{aligned}
(1+\varepsilon)^s rw(Q_{a_i}) &= (1+\varepsilon)^s rw(\mathrm{Rel}_{x(a_i)}(P_{i-1}) \cap Q_{a_i}) + (1+\varepsilon)^s rw(\mathrm{Ir}_{x(a_i)}(P_{i-1}) \cap Q_{a_i}) \\
&\leq \varepsilon \cdot \left(1 + \frac{6\varepsilon}{(1+\varepsilon)^s}\right) \cdot \sum_{p=a_{i-1}+1}^{a_i-1} rw(\mathrm{Rel}_{x(a_i)}(P_{i-1}) \cap Q_p) + 6\varepsilon \cdot rw(\mathrm{Rel}_{x(a_i)}(P_{i-1})) \\
&\leq \varepsilon \cdot \left(1 + \frac{6\varepsilon}{(1+\varepsilon)^s} + 6\right) \cdot \sum_{p=a_{i-1}+1}^{a_i-1} rw(\mathrm{Rel}_{x(a_i)}(P_{i-1}) \cap Q_p) \\
&\quad + 6\varepsilon \cdot rw(\mathrm{Rel}_{x(a_i)}(P_{i-1}) \cap Q_{a_i}) \\
&\leq \varepsilon \cdot \left(7 + \frac{6\varepsilon}{(1+\varepsilon)^s}\right) \cdot \sum_{p=a_{i-1}+1}^{a_i-1} rw(\mathrm{Rel}_{x(a_i)}(P_{i-1}) \cap Q_p) \\
&\quad + \frac{6\varepsilon^2}{(1+\varepsilon)^s}\left(1 + \frac{6\varepsilon}{(1+\varepsilon)^s}\right) \cdot \sum_{p=a_{i-1}+1}^{a_i-1} rw(\mathrm{Rel}_{x(a_i)}(P_{i-1}) \cap Q_p) \\
&\leq 8\varepsilon \cdot \sum_{p=a_{i-1}+1}^{a_i-1} rw(Q_p). \quad \square
\end{aligned}
$$

We conclude this section by summarizing those consequences of our considerations that we need in the following section. To that end, we denote the maximum ratio between weights of relevant jobs at any point in time by $W = \frac{\Delta \cdot \Gamma \cdot (1+\varepsilon)^{\Gamma+s}}{\varepsilon}$. Recall that the values of $\mu, \Delta, s, K, \Gamma$, and, hence, W depend only on ε and m. Furthermore, for some given schedule we denote by $o_{x,j}$ the amount of a job j that is processed within an interval I_x.

COROLLARY 2.14. *At $1 + O(\varepsilon)$ loss we can assume for each instance $\mathcal{I}$ with job set J and each interval I_x that*

(*a*) $p_j \in \{(1+\varepsilon)^k \mid \frac{\varepsilon^3}{4}(1+\varepsilon)^{-\Gamma} R_x \leq (1+\varepsilon)^k \leq \frac{1}{\varepsilon} R_x\}$ *for each* $j \in \mathrm{Rel}_x(J)$,
(*b*) $r_j \in \{(1+\varepsilon)^k \mid (1+\varepsilon)^{-\Gamma} R_x \leq (1+\varepsilon)^k \leq R_x\}$ *for each* $j \in \mathrm{Rel}_x(J)$,
(*c*) $w_j \in \{(1+\varepsilon)^k \mid w_x \leq (1+\varepsilon)^k \leq W \cdot w_x\}$ *for some value* w_x *and each* $j \in \mathrm{Rel}_x(J)$,
(*d*) $o_{x,j} \in \{\ell \cdot p_j/\mu \mid \ell \in \{0, \ldots, \mu\}\}$ *for each* $j \in \mathrm{Rel}_x(J)$,
(*e*) *the cardinality of* $\mathrm{Rel}_x(J)$ *is bounded by* $\Gamma\Delta$, *and*
(*f*) $\sum_{j \in \mathrm{Ir}_x(J)} w_j C_j \leq O(\varepsilon) \cdot \mathit{Opt}(\mathrm{Rel}_x(J))$.

Note that the respective sets of relevant processing times, release dates, weights, and processing time fractions have constant size.

3. ABSTRACTION OF ONLINE ALGORITHMS

In this section, we show how to construct a competitive-ratio approximation scheme based on the simplifications of Section 2. To do so, we restrict ourselves to such simplified instances and schedules. 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 $o_j < p_j$ 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 all values for the jobs of $J(S)$ in the interval-schedule fit to the corresponding values of the schedule 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, o_j of each job $j \in J(C)$,
—an interval-schedule for each interval $I_{x'}$ with $x' < x$.

A configuration is called *feasible* if there is a feasible schedule in which all values for the jobs of $J(C)$ in the configuration (including each interval-schedule) fit to the corresponding values of the schedule. The set of all feasible configurations (respecting the adaptations of Section 2) is denoted by $\mathcal{C}$. An *end-configuration* is a feasible configuration C for an interval I_x such that at time R_x, and not earlier, all jobs have been released and all relevant jobs have completely finished processing.

We say that an interval-schedule S is *feasible for a configuration C* if the set of jobs in $J(C)$ that 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. Call these algorithm maps *feasible*. 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 that 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 $I_x, I_{x'}$. Let further $\sigma : \tilde{J} \to \tilde{J}'$ be a bijection from a subset $\tilde{J} \subseteq J(S)$ to a subset $\tilde{J}' \subseteq J(S')$ and y an integer. The interval-schedules S, S' are *(σ, y)-equivalent* if $r_{\sigma(j)} = r_j(1+\varepsilon)^{x'-x}$, $p_{\sigma(j)} = p_j(1+\varepsilon)^{x'-x}$, $o_{\sigma(j)} = o_j(1+\varepsilon)^{x'-x}$, $q_{i\sigma(j)} = q_{ij}(1+\varepsilon)^{x'-x}$ and $w_{\sigma(j)} = w_j\,(1+\varepsilon)^y$ for all $j \in \tilde{J}$ and $i = 1, \ldots, m$. Denote by $J_{\mathrm{Rel}}(S) \subseteq J(S)$ and $J_{\mathrm{Rel}}(S') \subseteq J(S')$ the jobs of $J(S)$ relevant at time R_x and of $J(S')$ relevant at time $R_{x'}$. The interval-schedules S, S' are *equivalent* (denoted by $S \sim S'$) if a bijection $\sigma : J_{\mathrm{Rel}}(S) \to J_{\mathrm{Rel}}(S')$ and an integer y exist such that they are (σ, y)-equivalent.

Definition 3.5. Let C, C' be two feasible configurations for two intervals $I_x, I_{x'}$. Denote by $J_{\mathrm{Rel}}(C)$, $J_{\mathrm{Rel}}(C')$ the jobs that are relevant at times $R_x, R_{x'}$ in C, C', respectively. The 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

—$r_{\sigma(j)} = r_j(1+\varepsilon)^{x'-x}$, $p_{\sigma(j)} = p_j(1+\varepsilon)^{x'-x}$, $o_{\sigma(j)} = o_j(1+\varepsilon)^{x'-x}$, 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 for each $k \in \mathbb{N}$.

The restriction of equivalence to relevant jobs allows a reasonable measurement of the performance of end-configurations contained in the same equivalence class. On the one hand, we get equal performance ratios for equivalent configurations. On the other hand, we can approximate the actual competitive ratio of the complete solution since relevant jobs dominate the objective value. Consider therefore an end-configuration C. We denote the objective value of a subset $\tilde{J} \subseteq J(C)$ in the history of C by $val_C(\tilde{J})$. We further define $\rho(C) := val_C(J_{\mathrm{Rel}}(C))/\mathsf{Opt}(J_{\mathrm{Rel}}(C))$ to be the achieved competitive ratio of C when restricted to the relevant jobs.

LEMMA 3.6. *For each end-configuration $C \in \mathcal{C}$ it holds that*

(1) $(1+O(\varepsilon))^{-1}\rho(C) \le val_C(J(C))/\mathsf{Opt}(J(C)) \le (1+O(\varepsilon)) \cdot \rho(C)$ *and*
(2) $\rho(C) = \rho(C')$ *for any $C' \in \mathcal{C}$ with $C \sim C'$.*

PROOF. The first property follows by Lemma 2.11 via

$$\frac{val_C(J(C))}{\mathsf{Opt}(J(C))} \le \frac{val_C(J(C))}{\mathsf{Opt}(J_{\mathrm{Rel}}(C))} \le (1+O(\varepsilon))\frac{val_C(J_{\mathrm{Rel}}(C))}{\mathsf{Opt}(J_{\mathrm{Rel}}(C))} = (1+O(\varepsilon))\rho(C).$$

Similarly, we get

$$\rho(C) = \frac{val_C(J_{\mathrm{Rel}}(C))}{\mathsf{Opt}(J_{\mathrm{Rel}}(C))} \le \frac{val_C(J(C))}{\mathsf{Opt}(J_{\mathrm{Rel}}(C))} \le (1+O(\varepsilon))\frac{val_C(J(C))}{\mathsf{Opt}(J(C))}.$$

For the second property, denote the job weights and the resulting completion times of C (C') by w_j (w'_j) and C_j (C'_j). Since $C \sim C'$ there is an integer y and a bijection $\sigma : J_{\mathrm{Rel}}(C) \to J_{\mathrm{Rel}}(C')$ such that

$$\begin{aligned} val_{C'}(J_{\mathrm{Rel}}(C')) &= \sum_{j \in J_{\mathrm{Rel}}(C')} w'_j C'_j = \sum_{j \in J_{\mathrm{Rel}}(C)} w'_{\sigma(j)} C'_{\sigma(j)} \\ &= \sum_{j \in J_{\mathrm{Rel}}(C)} (1+\varepsilon)^y\, w_j\,(1+\varepsilon)^{x'-x}\, C_j \\ &= (1+\varepsilon)^{y+x'-x}\, val_C(J_{\mathrm{Rel}}(C)). \end{aligned}$$

If we transform the solution $\mathsf{Opt}(J_{\mathrm{Rel}}(C))$ by scaling weights by a factor of $(1+\varepsilon)^y$ and all time values by a factor of $(1+\varepsilon)^{x'-x}$ via σ, then we get a solution for $J_{\mathrm{Rel}}(C')$ with value $(1+\varepsilon)^{y+x'-x}\,\mathsf{Opt}(J_{\mathrm{Rel}}(C))$. Vice versa, applying the inverse transformation to the solution $\mathsf{Opt}(J_{\mathrm{Rel}}(C'))$ yields a solution for $J_{\mathrm{Rel}}(C)$ with value $(1+\varepsilon)^{x-x'-y}\,\mathsf{Opt}(J_{\mathrm{Rel}}(C'))$. By

$$\begin{aligned} \mathsf{Opt}(J_{\mathrm{Rel}}(C')) &\le (1+\varepsilon)^{y+x'-x}\,\mathsf{Opt}(J_{\mathrm{Rel}}(C)) \\ &\le (1+\varepsilon)^{y+x'-x}\,(1+\varepsilon)^{x-x'-y}\,\mathsf{Opt}(J_{\mathrm{Rel}}(C')) = \mathsf{Opt}(J_{\mathrm{Rel}}(C')), \end{aligned}$$

we can conclude equality. This finally yields

$$\rho(C') = \frac{val_{C'}(J_{\mathrm{Rel}}(C'))}{\mathsf{Opt}(J_{\mathrm{Rel}}(C'))} = \frac{(1+\varepsilon)^{y+x'-x}\, val_{C}(J_{\mathrm{Rel}}(C))}{(1+\varepsilon)^{y+x'-x}\, \mathsf{Opt}(J_{\mathrm{Rel}}(C))} = \rho(C). \quad \square$$

The following lemma shows that we can restrict the set of algorithm maps under consideration to those that treat equivalent configurations equivalently. We call algorithm maps obeying this condition (in addition to the restrictions of Section 2) *simplified algorithm maps*. A configuration C is called *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.

LEMMA 3.7. *At $1 + O(\varepsilon)$ loss we can restrict ourselves 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. We now construct a new algorithm map $\bar{f}$ that is simplified and has competitive ratio $\rho_{\bar{f}} \leq (1 + O(\varepsilon))\rho_f$ almost as good as f. Therefore, we pick for each equivalence class $\mathcal{C}_e \in \mathcal{C}/_\sim$ of the set of configurations a representative C_e (i.e., $[C_e] = \mathcal{C}_e$) that is realistic for f. For each configuration $C \in [C_e]$ that is equivalent to C_e with bijection σ and integer y, we define $\bar{f}$ by setting $\bar{f}(C)$ to be the interval-schedule for C that is (σ, y)-equivalent to $f(C_e)$. 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.

Consider now an instance $\bar{\mathcal{I}}$. We show that there is an instance $\mathcal{I}$ such that $\bar{f}(\bar{\mathcal{I}})/\mathsf{Opt}(\bar{\mathcal{I}}) \leq (1 + O(\varepsilon)) f(\mathcal{I})/\mathsf{Opt}(\mathcal{I})$ that implies the claimed competitive ratio for $\bar{f}$. Let $\bar{C}$ for interval $I_{\bar{x}}$ be the end-configuration obtained when $\bar{f}$ is applied iteratively on $\bar{\mathcal{I}}$. Let C_e be the representative of the equivalence class of $\bar{C}$, which was chosen above and which is realistic for f (C_e is also an end-configuration). Therefore, there is an instance $\mathcal{I}$ such that C_e is reached at time R_x when f is applied on $\mathcal{I}$. Hence, by Lemma 3.6 and $C_e \sim \bar{C}$ we get that $\mathcal{I}$ is the required instance. $\square$

LEMMA 3.8. *There are only constantly many simplified algorithm maps. Each simplified algorithm map can be described using finite information.*

PROOF. Assuming the simplifications introduced in Section 2, we can apply Corollary 2.14. Therefore, 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 that obeys the restrictions of Section 2, the interval-schedule $f(C)$ is fully specified when knowing only C and the equivalence class that 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. $\square$

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 3.9. *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 $\tilde{x} \leq E$ that 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, that is, the equivalence classes that 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 $k \in \mathbb{N}$, that is, 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 $x \leq E$, then 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 $\tilde{C}$ that is equivalent to C for the interval $I_{\tilde{x}}$ with $\tilde{x} := \bar{x} + k \leq E$. □

LEMMA 3.10. *Let f be a simplified algorithm map. There is an algorithm that approximates ρ_f up to a factor $1+\varepsilon$, that is, it computes a value ρ' with $\rho' \leq \rho_f \leq (1+O(\varepsilon))\rho'$.*

PROOF. According to Lemma 3.6 and Lemma 3.9, it suffices to construct the sets $\mathcal{C}_0^f, \dots, \mathcal{C}_E^f$ in order to approximate the competitive ratio of all end-configurations in these sets. Due to Corollary 2.14, we know all possible values for all parameters of jobs explicitly with lower and upper bounds. Due to Lemma 2.13, we can additionally ensure that each equivalence class of configurations corresponds to at most one part. So the enumeration can be done in a finite amount of time. We start with $\mathcal{C}_0^f$ and determine $f(C_e)$ for one representative C_e of each equivalence class $[C_e] \in \mathcal{C}_0^f$. Based on this, we determine the set $\mathcal{C}_1^f$. We continue inductively to construct all sets $\mathcal{C}_x^f$ with $x \leq E$.

We define $\rho_{\max}$ to be the maximum ratio $\rho(C)$ for an end-configuration $C \in \cup_{0 \leq x \leq E} \mathcal{C}_x^f$. Due to Lemma 3.9 and Lemma 3.6, the value $\rho_{\max}$ 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 3.10. 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 3.11. *$Pm|r_j, pmtn|\sum w_j C_j$ admits a competitive-ratio approximation scheme for any $m \in \mathbb{N}$.*

4. EXTENSIONS TO OTHER SETTINGS

Certain arguments in Section 2 do not transfer directly to more complex scheduling settings. In this section, we argue how to overcome the increased complexity. If we are able to obtain similar statements as for Corollary 2.7 and Corollary 2.14 for the extended settings, then competitive-ratio approximation schemes are also possible.

4.1. Non-Preemptive Scheduling

In this section, we review which statements or proofs of Section 2 make use of the possibility to preempt jobs and explain how to proceed in the non-preemptive case. One difference is that a schedule can be defined by a start time S_j for each job j with a resulting completion time of $C_j = S_j + p_j$ on its assigned machine i_j. Nevertheless, Lemma 2.1 can be proven similarly. We adapt the definition of time-stretch: When applying a time-stretch, we now shift each completion time $C_j \in I_x$ to the next interval while keeping the offset w.r.t. the beginning of the interval, that is, $C'_j = R_{x+1} + (C_j - R_x)$. For two completion time values $C_1 < C_2$ with $C_1 \in I_{x_1}$ and $C_2 \in I_{x_2}$ we observe that the difference after a time-stretch is $(C'_2 - C'_1) = (C_2 - C_1) + \sum_{x_1 \le x < x_2} \varepsilon |I_x|$. Hence, idle time is inserted right before each job that is partially processed at some interval bound. Therefore, we adapt the proof of Lemma 2.2 as follows: For each interval I_x the large jobs with start and completion time in I_x are shifted on each machine to the left such that on each machine the necessary amount of processing of small jobs can be assigned without preemption via Smith's rule. Lemmas 2.3, 2.4, and 2.5 of Section 2.1 are not affected and remain valid as they are stated. Since splitting of jobs into small atoms as in Lemma 2.6 is not applicable here we give a variant concerning start times. For each job j, we define $s(j)$ and $c(j)$ to be interval indices such that $S_j \in I_{s(j)}, C_j \in I_{c(j)}$. Recall that with our adapted objective function each ordering of jobs with $s(j) = c(j)$ assigned to one machine yields the same objective value. Hence, we only have to take care of the exact start times of those jobs with $s(j) < c(j)$ since they determine the amount of processing time assigned to each interval.

LEMMA 4.1. *There is a constant $\mu \in \mathbb{N}$ such that at $1+\varepsilon$ loss we can restrict ourselves to schedules where each large job j with $s(j) < c(j)$ has a start time of the form $S_j = R_{s(j)} + \ell_j \cdot |I_{s(j)}|/\mu$ with $\ell_j \in \{0, \dots, \mu - 1\}$.*

PROOF. We apply one time-stretch on a considered schedule yielding a new start time S_j for each job $j \in J$. Choose $\mu \in \mathbb{N}$ to be a constant integer such that $1/\mu$ is smaller than ε^2.

Consider now each interval I_x. For each machine i there is at most one job j with $x = s(j) < c(j)$ running on i. Due to the time-stretch, there is idle time of at least $(1+\varepsilon)|I_{x-1}|$ before the start of j. We now set $\ell_j = \lfloor (S_j - R_x)\mu/|I_x| \rfloor$ and decrease the start time of j to $S'_j = R_x + \ell_j |I_x|/\mu$. This yields

$$S'_j \ge R_x + \left((S_j - R_x)\frac{\mu}{|I_x|} - 1 \right) \frac{|I_x|}{\mu} = S_j - \frac{|I_x|}{\mu} \ge S_j - \varepsilon^2 |I_x| \ge S_j - \varepsilon |I_{x-1}|.$$

Hence, we get a feasible schedule of the required form with costs increased by at most a factor of $(1+\varepsilon)$. □

Therefore, a similar variant of Corollary 2.7 with a restatement of item (e) also holds in the non-preemptive setting.

The conclusion that also in the non-preemptive case only a constant amount of history is relevant demands a bit more work. First, the definition of the safety net (Lemma 2.8) needs to be adjusted, since it might be that all machines are executing jobs during the entire interval I_{x+s-1}. However, with the adapted definition of time-stretching, we know that there is reserved space on one machine in $[R_x, R_{x+s})$ to process all jobs released at time R_x. It remains to verify that an online algorithm (that does not know the future) can determine the beginning of this reserved space before it actually happens. Let j be the job scheduled to cover R_{x+s} on the last empty machine at this time. Denote its start and completion times before the time-stretch by $S_j \in I_{s(j)}, C_j \in I_{c(j)}$. From the above observation on time-stretches, we can additionally conclude that $C'_j - p_j - \sum_{s(j) \le x < c(j)} \varepsilon |I_x| \ge R_{s(j)+1}$. Hence, the safety net can be scheduled within interval $I_{s(j)+1}$ by our online algorithm since it decides about the complete schedule of an interval at its beginning.

With the existence of the safety net, we can consider similar periods and the proof of Lemma 2.9 that only counts release weights of periods remains valid. For the following, we need the small adaptation of $\Gamma := (K+1) \cdot s$. Definition 2.10, defining that a released job is recent, old, dominated, irrelevant, and relevant at time R_x, then uses this adapted Γ value. Nevertheless, the proof of Lemma 2.11 still holds. This is unfortunately not true for Lemma 2.12. 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. Hence, parts cannot be treated independently. Therefore, we switch back to consider complete instances. If we can no longer assume that an instance is mainly a sequence of significant periods, then we lose the premise of Lemma 2.9. We cannot simply use that only a constant amount of history is relevant.

To solve this problem, we will consider only special instances, where the weights after each insignificant period are sufficiently high such that they dominate the complete past until this period. The following lemma states the exact condition for these instances and proves again for each point in time that the relevant history of an instance with constant length dominates the irrelevant past even in the presence of insignificant periods. In a second step, we will reason why it is sufficient to consider only those instances satisfying the given condition.

To state the first lemma, we need the following definitions. For a given instance $\mathcal{I}$, let $a_1 < \ldots < a_{\ell+1}$ be again all indices satisfying Condition (2.2) of Lemma 2.13 (and $a_0 := -1$). Recall that the periods between $Q_{a_{i-1}}$ and Q_{a_i} build a sequence of significant periods for each $i = 1, \ldots, \ell$. Let first$(i), i = 1, \ldots, \ell$, denote the job that is released first after period Q_{a_i} at release time R_{x_i}. W.l.o.g. we can assume that $I_{x_i} \in Q_{a_i+1}$ since we can otherwise split the instance again after an empty period. For a given algorithm A, we denote by $\mathsf{A}(\mathcal{I}|i)$ the contribution of the jobs in $P_i = \bigcup_{p=a_i+1}^{a_{(i+1)}} Q_p$ to the objective value of the solution $\mathsf{A}(\mathcal{I})$. We denote by $\mathcal{I}(i) := \cup_{k\leq i} P_k$ the instance up to period $Q_{a_{(i+1)}}$. ($I_{x_{(\ell+1)}}$ denotes the first interval after $Q_{a_{(\ell+1)}}$.)

Since we increased Γ by the length of one period and $I_{x_i} \in Q_{a_i+1}$, we can use Lemma 2.9 to extend Lemma 2.11 such that $(1+\varepsilon)^s \cdot rw(\mathrm{Ir}_x(P_{i-1})) \leq 6\varepsilon \cdot rw(\mathrm{Rel}_x(P_{i-1}))$ actually holds for each $x_{(i-1)} \leq x < x_i, i = 1, \ldots, \ell+1$.

LEMMA 4.2. *For an instance $\mathcal{I}$ with*

$$rw(\mathrm{Rel}_{x_i-1}(\mathcal{I}(i-1))) \leq \frac{\varepsilon}{(1+\varepsilon)^s(1+7\varepsilon)} \cdot rw(\mathrm{first}(i)) \quad \textit{and} \tag{4.1a}$$

$$\max\{w_j \mid j \in \mathrm{Rel}_{x_i-1}(\mathcal{I}(i-1))\} \leq w_{\mathrm{first}(i)} \tag{4.1b}$$

for each $i = 1, \ldots, \ell$, it holds that

$$(1+\varepsilon)^s rw(\mathrm{Ir}_x(\mathcal{I}(i-1))) \leq 7\varepsilon \cdot rw(\mathrm{Rel}_x(\mathcal{I}(i-1))) \tag{4.2}$$

for each $i = 1, \ldots, \ell+1$ and each $x_{i-1} \leq x < x_i$.

PROOF. We proof this by induction and start with the base case $i = 1$. Consider some $x_0 \leq x < x_1$. By definition, all periods before Q_{a_1} are significant. By Lemma 2.11 we get

$$(1+\varepsilon)^s rw(\mathrm{Ir}_x(\mathcal{I}(0))) \leq 7\varepsilon \cdot rw(\mathrm{Rel}_x(\mathcal{I}(0))).$$

For the inductive step consider some $i = 2, \ldots, \ell+1$ and assume that Equation (4.2) holds for $i-1$ and each $x_{(i-2)} \leq x < x_{(i-1)}$. Observe that $\mathcal{I}(i-1) = \mathcal{I}(i-2) \cup P_{i-1}$. Due to Condition (4.1b), we know that $\mathrm{Dom}_x(\mathcal{I}(i-1)) \cap P_{i-1} = \mathrm{Dom}_x(P_{i-1})$ for each $x_{i-1} \leq x < x_i$, which yields:

$$\begin{aligned}(1+\varepsilon)^s rw(\mathrm{Ir}_x(\mathcal{I}(i-1))) &= (1+\varepsilon)^s \left[rw(\mathrm{Ir}_x(\mathcal{I}(i-1)) \cap \mathcal{I}(i-2)) + rw(\mathrm{Ir}_x(\mathcal{I}(i-1)) \cap P_{i-1})\right] \\ &\leq (1+\varepsilon)^s \left[rw(\mathcal{I}(i-2)) + rw(\mathrm{Ir}_x(P_{i-1}))\right]\end{aligned}$$

$$\begin{aligned}
&\overset{(4.2)}{\leq} (1+\varepsilon)^s \left[(1+7\varepsilon) rw(\mathrm{Rel}_{x_{(i-1)}-1}(\mathcal{I}(i-2))) + rw(\mathrm{Ir}_x(P_{i-1}))\right] \\
&\overset{(4.1)}{\leq} \varepsilon \cdot rw(\mathit{first}(i-1)) + (1+\varepsilon)^s \cdot rw(\mathrm{Ir}_x(P_{i-1}))) \\
&\overset{2.11}{\leq} \varepsilon \cdot rw(\mathit{first}(i-1)) + 6\varepsilon \cdot rw(\mathrm{Rel}_x(P_{i-1}))) \\
&\leq 7\varepsilon \cdot rw(\mathrm{Rel}_x(\mathcal{I}(i-1))).
\end{aligned}$$

This proves the hypothesis. □

Consider now some online algorithm A with competitive ratio ρ_{A} that performs with ratio $\overline{\rho_{\mathsf{A}}} := \max\{\mathsf{A}(\mathcal{I})/\mathsf{Opt}(\mathcal{I}) \mid \mathcal{I} \text{ satisfies } (4.1)\} \leq \rho_{\mathsf{A}}$ on our desired instances where the first released job after each insignificant period dominates the complete instance up to this period. We now want to find a new online algorithm A' that modifies the weights of each given arbitrary instance $\mathcal{I}$ to an instance $\mathcal{I}'$ satisfying (4.1) and creates a solution $\mathsf{A}(\mathcal{I}')$ that yields a schedule $\mathsf{A}'(\mathcal{I})$ for the original instance with

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

Therefore, an online algorithm that proves to be good on only these instances yields an online algorithm for general instances with almost the same competitive ratio. Hence, it is sufficient to consider only these instances for the enumeration of simplified algorithm maps.

Lemma 4.3. *At* $(1+O(\varepsilon))$ *loss, we can restrict ourselves to instances satisfying* (4.1).

Proof. For a considered online algorithm A, we define a new algorithm A' that creates for a given instance $\mathcal{I}$ the new instance $\mathcal{I}'$ and applies A on $\mathcal{I}'$ as follows:

At the beginning, we set $\mathcal{I}'(0) = \mathcal{I}(0)$. After period Q'_{a_i} we add for each further $i = 1, \ldots, \ell$ and for each job $j \in P_i$ of instance $\mathcal{I}$ a new job j' to $\mathcal{I}'$ with equal processing time and release date but new weight $w_{j'} := v_i \cdot w_j$ such that $w_{j'} \leq w_{\mathrm{first}(i)'}$ for each $j' \in \mathrm{Rel}_{x_i-1}(\mathcal{I}'(i-1))$ and

$$rw(\mathrm{Rel}_{x_i-1}(\mathcal{I}'(i-1))) \leq \frac{\varepsilon}{(1+\varepsilon)^s(1+7\varepsilon)} \cdot rw(\mathrm{first}(i)').$$

Hence, $\mathcal{I}'$ satisfies Equation (4.1). The schedules for $\mathsf{A}'(\mathcal{I}(i))$ are then defined by applying $\mathsf{A}(\mathcal{I}'(i))$.

We then can easily observe that $\mathsf{A}'(\mathcal{I}|i) \cdot v_i = \mathsf{A}(I'|i) \leq \mathsf{A}(I'(i))$ and get, by Lemma 4.2,

$$\begin{aligned}
\mathsf{Opt}(\mathcal{I}'(i)) &\leq \mathsf{Opt}(\mathcal{I}|i) \cdot v_i + (1+\varepsilon)^s rw(\mathcal{I}'(i-1)) \\
&\overset{(4.2)}{\leq} \mathsf{Opt}(\mathcal{I}|i) \cdot v_i + (1+\varepsilon)^s (1+7\varepsilon) rw(\mathrm{Rel}_{x_i-1}(\mathcal{I}'(i-1))) \\
&\overset{(4.1)}{\leq} \mathsf{Opt}(\mathcal{I}|i) \cdot v_i + \varepsilon \cdot rw(\mathrm{first}(i)') \leq (1+\varepsilon) v_i \cdot \mathsf{Opt}(\mathcal{I}|i).
\end{aligned}$$

The combination finally yields

$$\frac{\mathsf{A}'(\mathcal{I})}{\mathsf{Opt}(\mathcal{I})} \leq \max_i \frac{\mathsf{A}'(\mathcal{I}|i)}{\mathsf{Opt}(\mathcal{I}|i)} \leq \max_i \frac{(1+\varepsilon) v_i \cdot \mathsf{A}(\mathcal{I}'(i))}{\mathsf{Opt}(\mathcal{I}'(i)) \cdot v_i} \leq (1+\varepsilon) \max_i \frac{\mathsf{A}(\mathcal{I}'(i))}{\mathsf{Opt}(\mathcal{I}'(i))}.$$

Hence, we get $\rho_{\mathsf{A}'} \leq (1+O(\varepsilon))\overline{\rho_{\mathsf{A}}} \leq (1+O(\varepsilon))\rho_{\mathsf{A}}$. □

To conclude, even without treating parts independently, we can restrict ourselves to instances where only a constant amount of history is relevant. Therefore, we can state an adapted version of Corollary 2.14 for the non-preemptive setting:

COROLLARY 4.4. *At $1 + O(\varepsilon)$ loss we can assume for each instance $\mathcal{I}$ with job set J and each interval I_x that*

(*a*) $p_j \in \{(1+\varepsilon)^k \mid \frac{\varepsilon^3}{4}(1+\varepsilon)^{-\Gamma} R_x \le (1+\varepsilon)^k \le \frac{1}{\varepsilon} R_x\}$ *for each* $j \in \mathrm{Rec}_x(J)$,
(*b*) $r_j \in \{(1+\varepsilon)^k \mid (1+\varepsilon)^{-\Gamma} R_x \le (1+\varepsilon)^k \le R_x\}$ *for each* $j \in \mathrm{Rec}_x(J)$,
(*c*) $w_j \in \{(1+\varepsilon)^k \mid w_x \le (1+\varepsilon)^k \le W \cdot w_x\}$ *for some value* w_x *and each* $j \in \mathrm{Rel}_x(J)$,
(*d*) $s(j), c(j) \in \{x - \Gamma, \dots, x + s\}$ *for each* $j \in \mathrm{Rec}_x(J)$ *and*
$S_j \in \{R_{s(j)} + \ell_j \cdot |I_{s(j)}|/\mu \mid \ell_j \in \{0, \dots, \mu - 1\}\}$ *if* $s(j) < c(j)$,
(*e*) *the cardinality of* $\mathrm{Rec}_x(J) \supseteq \mathrm{Rel}_x(J)$ *is bounded by* $\Gamma\Delta$, *and*
(*f*) $\sum_{j \in \mathrm{Ir}_x(J)} w_j C_j \le O(\varepsilon) \cdot \mathit{Opt}(\mathrm{Rel}_x(J))$.

We now continue with adaptations for Section 3. Since schedules are defined in the non-preemptive case via assigned machines and start times, we discard the values of o_j and q_{ij} from the definitions of interval-schedules and configurations (cf. Definitions 3.1 and 3.2). Instead, each interval-schedule S for interval I_x knows in addition to all properties of $J(S)$ for each already running job j with $s(j) < x \le c(j)$ the values S_j and i_j and assigns these two values to each job j scheduled to start within I_x (i.e., $s(j) = x$). Also, each configuration C for interval I_x contains additionally the values S_j and i_j for each job $j \in P(C) = \{j \in J(C) \mid s(j) < x \le c(j)\}$ that is partially processed at time R_x.

In Definition 3.4 of two (σ, y)-equivalent interval-schedules S, S', we replace the o_j- and q_{ij}-conditions by $i_{\sigma(j)} = i_j$ and $x - x' = s(j) - s(\sigma(j)) = c(j) - c(\sigma(j))$ for each $j \in \tilde{J}$ and $S_{\sigma(j)} = S_j(1+\varepsilon)^{x'-x}$ for each $j \in \tilde{J}$ with $s(j) < c(j)$.

In the non-preemptive setting, it is possible that jobs running at the beginning of an interval are also dominated and hence irrelevant at that time. Nevertheless, configurations with different dominated jobs partially processed at time R_x must be treated differently. To deal with this circumstance, we have to extend the definition of equivalent configurations appropriately:

Definition 4.5. Two feasible configurations C, C' for the intervals $I_x, I_{x'}$ are called equivalent

—if they are equivalent in the sense of Definition 3.5 without the o_j-conditions
—and if there is a bijection $\psi : P(C) \to P(C')$ such that $r_{\psi(j)} = r_j(1+\varepsilon)^{x'-x}$, $p_{\psi(j)} = p_j(1+\varepsilon)^{x'-x}$, $i_{\sigma(j)} = i_j$, and $S_{\sigma(j)} = S_j(1+\varepsilon)^{x'-x}$ for each $j \in P(C)$.

This extension allows a similar proof of Lemma 3.7 since for any feasible algorithm-map f a (σ, y)-equivalent interval schedule of $f(C_e)$ for any configuration $C \in [C_e]$ equivalent to the representative C_e has enough idle time where the running dominated jobs of $P(C)$ can be feasibly continued (and since Lemma 3.6 still holds).

Note that each job of $P(C)$ is recent at time R_x since no job remains unfinished for more than one period. Corollary 4.4 explicitly takes care on which statement is valid not only for the set of relevant but also for the set of recent jobs at any point in time. Therefore, we can use Corollary 4.4 in the proof of Lemma 3.8 instead of Corollary 2.14 to conclude again that there are only constantly many simplified algorithm maps. With this, the implications for Lemmas 3.9 and 3.10 apply similarly, and we can construct a competitive-ratio approximation scheme as in Section 3.

THEOREM 4.6. *$Pm|r_j|\sum w_j C_j$ admits a competitive-ratio approximation scheme for any $m \in \mathbb{N}$.*

4.2. Scheduling on Related Machines

We now consider the setting of scheduling related machines, where each machine i has a certain speed s_i associated to it. Processing job j on machine i takes p_j/s_i time units.

We review the statements and proofs of Section 2 and discuss where adjustments need to be made. Without loss of generality, we can assume that the slowest machine has unit speed. Let $s_{\max}$ denote the maximum speed in an instance, and denote by $s_{\sum}$ the sum of the processing speeds of all machines. We use the following adjusted version of Lemma 2.1:

LEMMA 4.7. *At $1+O(\varepsilon)$ loss we can restrict ourselves to instances where all processing times, release dates, and weights are powers of $1+\varepsilon$, no job is released before time $t = 1$, and $r_j \geq \varepsilon \cdot p_j/s_{\max}$ for all jobs j.*

PROOF. Everything but the last statement of the lemma follows exactly as in the proof of Lemma 2.1. For the last claim, let $A_1 = [\ell_1, u_1), \ldots, A_k = [\ell_k, u_k)$ be the time intervals during which job j is to processed in an optimal schedule that satisfies the other properties. Let t_ε denote the time where exactly an $\frac{\varepsilon}{1+\varepsilon}$-fraction of j has been processed. We multiply every time event by a factor of $(1+\varepsilon)$. Then we have enough processing time reserved that we can shift the starting time of j to $r'_j := (1+\varepsilon)t_\varepsilon$. If started j at time $\ell_1(1+\varepsilon)$, then at time $(1+\varepsilon)t_\varepsilon$ we would already have processed an ε-fraction of j. Therefore, $r'_j \geq \frac{\varepsilon p_j}{s_{\max}}$ is true even if j was processed on the fastest machine up to time r'_j. □

We define $L_x = \{j \in J \mid r_j = R_x, p_j > \frac{\varepsilon^2 |I_x|}{s_{\max}}\}$ to be the set of *large* jobs released at R_x and $S_x = \{j \in J \mid r_j = R_x, p_j \leq \frac{\varepsilon^2 |I_x|}{s_{\max}}\}$ to be the set of *small* jobs released at R_x.

LEMMA 4.8. *At $1+\varepsilon$ loss we can restrict ourselves 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 $\bigcup_{x' \leq x} S_{x'}$ and no small job is preempted or only partially processed at the end of an interval. Therefore, we can restrict ourselves to instances with $p(S_x) \leq m \cdot |I_x| \cdot s_{\sum}$ for each interval I_x.*

The proof of Lemma 4.8 works nearly the same way as the proof of Lemma 2.2. When scheduling the large jobs preemptively in the beginning of an interval, we may use the linear programming–based algorithm for minimimizing the makespan by Lawler and Labetoulle [1978]. Then, when assigning the small jobs, we fill the machines one after the other in nonincreasing order of their speeds. Also, the bound on processing times now includes a factor of $s_{\sum}$.

For Lemma 2.3, we have a slight modification:

LEMMA 4.9. *At $1+\mathcal{O}(\varepsilon)$ loss we can restrict ourselves to instances such that $p_j \geq \frac{\varepsilon^2}{4s_{\max}} \cdot |I_x|$ for each job $j \in S_x$. In these instances, the number of distinct processing times of each set S_x is bounded from above by $\log_{1+\varepsilon} 4$.*

PROOF. We call a job $j \in S_x$ *tiny* if $p_j \leq \frac{\varepsilon^2}{4s_{\max}} \cdot |I_x|$. We let ℓ be the largest integer such that $\sum_{i=1}^{\ell} p_i \leq \frac{\varepsilon^2}{2s_{\max}} \cdot |I_x|$. We construct the packs the same way as in Lemma 2.3 and get the that the processing time of all but possibly the last pack released at time R_x is in the interval $[\frac{\varepsilon^2}{4s_{\max}} \cdot |I_x|, \frac{\varepsilon^2}{2s_{\max}} \cdot |I_x|]$. We apply one time-stretch and the rest of the argumentation holds. Each processing times of a job in S_x then equals $(1+\varepsilon)^y$ for some integer y such that

$$\frac{\varepsilon^3}{4s_{\max}} \cdot (1+\varepsilon)^x \leq (1+\varepsilon)^y \leq \frac{\varepsilon^2 |I_x|}{s_{\max}} = \frac{\varepsilon^3}{s_{\max}} \cdot (1+\varepsilon)^x.$$

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

Possible processing times of large jobs fall in the range $\left[\frac{\varepsilon^2}{s_{\max}}\cdot|I_x|, \frac{r_j s_{\max}}{\varepsilon}\right]$, and, therefore, we get an upper bound on the number of distinct processing times of large jobs of $8\log_{1+\varepsilon}\varepsilon\cdot s_{\max}$.

LEMMA 4.10. *Without loss, we can restrict ourselves to instances with* $|L_x| \leq (m/\varepsilon^2 + m)8s_{\max}^2\log_{1+\varepsilon}\frac{1}{\varepsilon}$ *for each set* L_x.

PROOF. Same calculation as in the proof of Lemma 2.4. □

LEMMA 4.11. *There is a constant* $\mu \in \mathbb{N}$ *such that at* $1 + O(\varepsilon)$ *loss we can restrict ourselves to schedules such that at the end of each interval, each large job* j *is processed to an extent that is an integer multiple of* p_j/μ.

PROOF. We choose $\mu \in \mathbb{N}$ such that $1/\mu$ is smaller than $\varepsilon^4/(s_{\max}^2\cdot 8\log_{1+\varepsilon}\frac{1}{\varepsilon})$. The remainder of the proof is identical to the argumenation in the proof of Lemma 2.6. □

The following corollary summarizes the simplifications that we need for the setting of related machines.

COROLLARY 4.12. *At* $1 + O(\varepsilon)$ *loss we can assume that for each interval* I_x

(*a*) *each job* j *released at time* R_x *has a processing time* $p_j = (1+\varepsilon)^k \in \left[\frac{\varepsilon^3 R_x}{4s_{\max}}, \frac{R_x\cdot s_{\max}}{\varepsilon}\right]$ *for some integer* k,
(*b*) *there are at most* $8s_{\max}^2\log_{1+\varepsilon}\frac{1}{\varepsilon} + \log_{1+\varepsilon}4$ *distinct processing time values of jobs released at* R_x,
(*c*) *at most* Δ *jobs are released at* R_x,
(*d*) *each small job started in* I_x *is completed in* I_x *without preemption, and*
(*e*) *at the end of* I_x, *each job* j *is processed to an extent of* $\ell_{x,j}\cdot p_j/\mu$ *for some* $\ell_{x,j} \in \{0,\ldots,\mu\}$.

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 2.8 in the related machine setting. For the non-preemptive setting, we incorporate the adjustments introduced in Section 4.1. At $1+\varepsilon$ loss we can round the speeds of the machines to powers of $1+\varepsilon$. Thus, for a constant number of machines m, there are only constantly many possible vectors s characterizing the speeds of the machines if the machine speeds lie in a constant range. Assuming the latter, we apply our enumeration scheme to each of these speed vectors of the machines and obtain a competitive-ratio approximation scheme for this case.

THEOREM 4.13. *For any* $m \in \mathbb{N}$, *we obtain competitive-ratio approximation schemes for* $Qm|r_j, pmtn|\sum w_jC_j$, *and* $Qm|r_j|\sum w_jC_j$, *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 a competitive-ratio 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 4.14. *For* $Qm|r_j, pmtn|\sum w_jC_j$, *at* $1 + O(\varepsilon)$ *loss, we can restrict ourselves to instances in which* $s_{\max}$ *is bounded by* m/ε.

PROOF. Given a schedule on related machines with speed values $s_1, \dots, s_{\max}$, 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 ignored. The remaining machines have speeds in $[\frac{\varepsilon}{m} s_{\max}, s_{\max}]$. Assuming the slowest machines has unit speed gives the desired bound. □

As speeds are geometrically rounded, we have for each value m only finitely many speed vectors.

THEOREM 4.15. *For any $m \in \mathbb{N}$ we obtain a competitive-ratio approximation scheme for $Qm \mid r_j, pmtn \mid \sum w_j C_j$.*

4.3. Preemptive Scheduling on Unrelated Machines

In the unrelated machine setting, where each job j has its individual processing time p_{ij} on machine i, the problem complexity increases significantly. We restrict ourselves to preemptive scheduling and show how to decrease the problem complexity in order to obtain competitive-ratio approximation schemes. 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 Afrati et al. [1999] for a counterexample).

LEMMA 4.16. *At $1+\varepsilon$ loss, we can restrict ourselves to instances in which, for each job j, the ratio of any two of its finite processing times is bounded by m/ε.*

PROOF. Consider a schedule for an instance that does not satisfy the property. We stretch time twice and thus we gain a free space of $\varepsilon|I_x|$ in each interval I_x. Consider some interval I_x and a job j that 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 $p_{ij} \le \frac{\varepsilon}{m} p_{i'j}$. □

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 j with $r_j = R_x$ is *small* if $\tilde{p}_j \le \varepsilon^2 |I_x|/W$ and *large* otherwise. For each job class separately, we perform the adjustments of Section 2. This yields the following lemma.

LEMMA 4.17. *At $1 + O(\varepsilon)$ loss, we can restrict ourselves to instances and schedules such that*

—*for each job j released at time R_x and each machine i, it holds that $p_{i,j}$ is a power of $1+\varepsilon$ and $p_{i,j} \in \left[\frac{\varepsilon^4 R_x}{4Wm}, \frac{R_x \cdot m}{\varepsilon^2}\right] \cup \{\infty\}$,*
—*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 $\tilde{\Delta}$,*
—*for each interval I_x, the processing time of each large job j during I_x on each machine equals an integer multiple of $\tilde{p}_j \cdot \tilde{\mu}$ for some constant $\tilde{\mu}$, and*
—*small jobs are never preempted and finish in the same interval in which 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 4.18. *For any $m \in \mathbb{N}$, we obtain a competitive-ratio approximation scheme for $Rm|\, r_j,\, pmtn\, |\sum w_j C_j$.*

5. RANDOMIZED ALGORITHMS

When algorithms are allowed to make random choices and we consider expected values of schedules, we can restrict ourselves to instances that 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 ourselves to instances in which all jobs are released in at most $(1+\varepsilon)^s/\varepsilon$ consecutive periods.*

PROOF. Let A be a randomized online algorithm with a competitive ratio of ρ_{A} on instances that span at most $(1+\varepsilon)^s/\varepsilon$ periods. We construct a new randomized online algorithm A' that works on arbitrary instances $\mathcal{I}$ such that $\rho_{\mathsf{A}'} \le \rho_{\mathsf{A}}(1+\varepsilon)$. At the beginning of A', we choose an offset $o \in \{0, \dots, M-1\}$ uniformly at random with $M := \lceil (1+\varepsilon)^s/\varepsilon \rceil$. In instance $\mathcal{I}$, we move all jobs to their safety net that are released in periods $Q \in \mathcal{Q} := \{Q_i | i \equiv o \bmod M\}$. This splits the instance into parts $P_0, \dots, P_k$, where each part P_ℓ consists of the periods $Q_{o+(\ell-1)\cdot M}, \dots, Q_{o+\ell\cdot M-1}$. 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

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

Thus, the total expected cost of the computed schedule is

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

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

A randomized online algorithm can be viewed as a function that maps every possible configuration C to a probability distribution of interval-schedules that 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 $x \leq \bar{\Gamma}$, let $\bar{\mathcal{S}}$ be the set of interval-schedules for intervals I_x with $x \leq \bar{\Gamma}$, 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 $C \in \bar{\mathcal{C}}$.

By restricting ourselves to δ-discretized algorithm maps, we only slightly increase 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 $\rho_g \leq \rho_f (1 + \varepsilon)$.*

Proof. 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 $\lfloor \frac{f(C,S)}{\delta} \rfloor \cdot \delta \leq g(C, S) \leq \lceil \frac{f(C,S)}{\delta} \rceil \cdot \delta$ 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 $S(\mathcal{I})$ that 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 $C_0, \ldots, C_{\bar{\Gamma}-1}$ the configurations that algorithms are faced with when computing $S(\mathcal{I})$, that is, each configuration C_x contains the jobs that 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 $I_0, \ldots, I_{x-1}$. 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 $\mathcal{S}_L^g(\mathcal{I}) := \mathcal{S}(\mathcal{I}) \backslash \mathcal{S}_H(\mathcal{I})$.

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 $(\frac{k+1}{k})^{\bar{\Gamma}} \leq 1 + \varepsilon/2$ 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 $\rho_g \leq (1+\varepsilon)\,\rho_f$. □

Like in the deterministic case, we can now show that at $1+\varepsilon$ loss it suffices to restrict ourselves to *simplified δ-discretized algorithm maps* that treat equivalent configurations equivalently, similarly to Lemma 3.7 (replacing Γ by $\bar{\Gamma}$ in Definition 2.10 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 competitive-ratio approximation schemes for $Pm|r_j,(pmtn)|\sum w_jC_j$, $Qm|r_j, pmtn|\sum w_jC_j$, and $Rm|r_j, pmtn|\sum w_jC_j$ and for $Qm|r_j|\sum w_jC_j$ with a bounded range of speeds for any fixed $m \in \mathbb{N}$.*

6. GENERAL MIN-SUM OBJECTIVES AND MAKESPAN

In this section, we show how to adjust our techniques presented in the previous sections to obtain competitive-ratio approximation schemes for minimizing monomial job cost functions and for minimizing the makespan when jobs arrive online over time.

Monomial Cost Functions. Instead of minimizing the weighted sum of completion time $\sum_j w_jC_j$ as in the previous sections, we assume now that we are given a global function f with $f(x) = k\cdot x^\alpha$ and for constants $\alpha \geq 1$ and $k > 0$, and our objective is to minimize $\sum_{j\in J} w_j f(C_j)$. Since such monomial functions f have the property that $f((1+\varepsilon)C_j) \leq (1+O(\varepsilon))f(C_j)$, the arguments in previous sections apply almost directly to this more general min-sum objective. In each step of simplification and abstraction, we have an increased loss in the performance guarantee, but it is covered by the $O(\varepsilon)$-term. Thus, we obtain competitive-ratio approximation schemes also for this setting.

THEOREM 6.1. *For any $m \in \mathbb{N}$ there are deterministic and randomized competitive-ratio approximation schemes for preemptive and non-preemptive scheduling, on m identical, related (with bounded speed ratio when non-preemptive), and unrelated machines (only preemptive) for the objective of minimizing $\sum_{j\in J} w_j f(C_j)$, with $f(x) = k\cdot x^\alpha$ and constant $\alpha \geq 1, k > 0$.*

Makespan Over Time. We consider now the objective of minimizing the makespan $C_{\max}$. We use the same online-time model as before in which jobs become known at their release date. Note that our online model differs from the setting where jobs arrive online one by one and one has to assign a job irrevocably to a machine once it has been presented. The latter model is particularly common for online makespan minimization, see for example, Fleischer and Wahl [2000] and Rudin III and Chandrasekaran [2003].

The *simplifications within intervals* of Section 2 are based on arguing on completion times of individual jobs and clearly hold also for the last job. Thus, they directly apply to minimizing the makespan. Furthermore, we simplify the definition of *irrelevant history* in Section 2 by omitting the partition of the instance into parts. We observe that when the last job is released at time R_{x^*}, then all jobs j with $r_j \leq R_{x^*-s}$ are irrelevant for the objective value: Such a job j finishes in any schedule no later than time R_{x^*} (due to the safety net), whereas $OPT \geq R_{x^*}$. Therefore, we define a job j to be *irrelevant at time R_x* if $r_j \leq R_{x-s}$. We keep Definition 3.4 for the equivalence relation of schedules as it is except for the notion of job weights that are irrelevant for the makespan. Based on the above definition for relevant jobs, we define equivalence classes of configurations. With this definition, we can still restrict ourselves to algorithm maps f with $f(C) \sim f(C')$ for any two equivalent configurations C, C' (Lemma 3.7). Lemmas 3.8 to 3.10 then hold accordingly. Finally, note that since we do not split the instance into parts, we do not need (an adjusted version of) Lemma 2.12.

THEOREM 6.2. *For any $m \in \mathbb{N}$, we obtain competitive-ratio approximation schemes for* $\mathrm{Pm}|rj, (pmtn)|C_{\max}$, $\mathrm{Qm}|r_j, pmtn|C_{\max}$, *and* $\mathrm{Rm}|r_j, pmtn|C_{\max}$, *and for* $\mathrm{Qm}|r_j|C_{\max}$ *assuming a constant range of machine speeds.*

For constructing randomized competitive-ratio approximation schemes for minimizing the makespan, similarly to Lemma 5.1, we can show that we can restrict our attention to instances that span only a constant number of periods.

LEMMA 6.3. *For randomized algorithms for minimizing the makespan, at $1+O(\varepsilon)$ loss we can restrict ourselves to instances in which all jobs are released in at most $(1+\varepsilon)^s/\varepsilon$ consecutive periods.*

PROOF. Let A be a randomized online algorithm for minimizing the makespan over time with a competitive ratio of ρ_{A} on instances that span at most $(1+\varepsilon)^s/\varepsilon$ periods. We construct a new randomized online algorithm A' that works on arbitrary instances such that $\rho_{\mathsf{A}'} \leq \rho_{\mathsf{A}}(1+O(\varepsilon))$. Our reasoning is similar to the proof of Lemma 5.1.

At the beginning of A', we choose an offset $o \in \{0, \dots, M-1\}$ uniformly at random with $M := \lceil (1+\varepsilon)^s/\varepsilon \rceil$. Given an 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 $P_0, \dots, P_k$ where each part P_ℓ consists of the periods $Q_{o+(\ell-1)\cdot M}, \dots, Q_{o+\ell\cdot M-1}$. Note that at the end of each part no job remains. We present each part separately to A.

We bound the competitive ratio $\rho_{\mathsf{A}'}$ of the resulting algorithm. Let $R_{x^*} := \max_{j\in\mathcal{I}} r_j$. Moving the jobs from periods $\mathcal{Q}$ has an effect on the optimal makespan only if o is chosen such that at least one job j with $r_j > R_{x^*-s}$ is moved. There are at most two offsets o such that this happens. In that case, the algorithm still achieves a competitive ratio of at most $(1+\varepsilon)^s$. In all other cases, A' achieves a competitive ratio of at most ρ_{A}. Thus, we can bound $\rho_{\mathsf{A}'}$ by

$$\rho_{\mathsf{A}'} \leq \frac{2}{M}(1+\varepsilon)^s + \frac{M-2}{M}\rho_{\mathsf{A}} \leq 2\varepsilon + \rho_{\mathsf{A}} \leq (1+O(\varepsilon))\rho_{\mathsf{A}}. \quad \square$$

We can prove, similarly as in Lemma 5.3, that any randomized algorithm map f can be well approximated by a discretized randomized algorithm map g. Hence, we obtain the following theorem.

THEOREM 6.4. *For any $m \in \mathbb{N}$, we obtain randomized competitive-ratio approximation schemes for* $\mathrm{Pm}|r_j, (pmtn)|C_{\max}$, $\mathrm{Qm}|r_j, pmtn|C_{\max}$, *and* $\mathrm{Rm}|r_j, pmtn|C_{\max}$ *and for* $\mathrm{Qm}|r_j|C_{\max}$ *assuming a constant range of machine speeds.*

7. CONCLUSION

We introduce the concept of competitive-ratio approximation schemes that compute online algorithms with a competitive ratio arbitrarily close to the best possible competitive ratio. We provide such schemes for various problem variants of scheduling jobs online to minimize the weighted sum of completion times, arbitrary monomial cost functions, and the makespan.

The techniques derived in this article provide a new and interesting view on the behavior of online algorithms. We believe that they contribute to the understanding of such algorithms, possibly open a new viewpoint for understanding them, and open a new line of research that yields even further insights. In particular, recent results building on our article show that our methods can also be applied to other scheduling objective settings [Kurpisz et al. 2012], other online models [Chen et al. 2015; Megow and Wiese 2013], and also to different online problems such as the *k*-server problem [Mömke 2013].

REFERENCES

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. 1999. 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'99)*. 32–43.

E. J. Anderson and C. N. Potts. 2004. On-line scheduling of a single machine to minimize total weighted completion time. *Math. Operat. Res.* 29 (2004), 686–697.

J. Augustine, S. Irani, and C. Swamy. 2008. Optimal power-down strategies. *SIAM J. Comput.* 37, 5 (Jan. 2008), 1499–1516. DOI:http://dx.doi.org/10.1137/05063787X

N. Bansal and K. Pruhs. 2010. The geometry of scheduling. In *Proceedings of the 51th Annual IEEE Symposium on Foundations of Computer Science (FOCS'10)*. IEEE Computer Society, 407–414.

N. Bansal, A. Srinivasan, and O. Svensson. 2016. Lift-and-round to improve weighted completion time on unrelated machines. In *Proceedings of the 48th Annual Symposium on Theory of Computing (STOC'16)*. ACM, 156–167. DOI:http://dx.doi.org/10.1145/2897518.2897572

S. Chakrabarti, C. A. Phillips, A. S. Schulz, D. B. Shmoys, C. Stein, and J. Wein. 1996. Improved algorithms for minsum criteria. In *Proceedings of the 23rd International Conference on Automata, Languages and Programming (ICALP'96)*, Vol. 1099. 646–657.

C. Chekuri, R. Motwani, B. Natarajan, and C. Stein. 2001. Approximation techniques for average completion time scheduling. *SIAM J. Comput.* 31 (2001), 146–166.

B. Chen and A. P. A. Vestjens. 1997. Scheduling on identical machines: How good is LPT in an on-line setting? *Operat. Res. Lett.* 21 (1997), 165–169.

L. Chen, D. Ye, and G. Zhang. 2015. Approximating the optimal algorithm for online scheduling problems via dynamic programming. *Asia-Pacif. J. Operat. Res.* 32, 01 (2015), 1540011. DOI:http://dx.doi.org/10.1142/S0217595915400114

M. Cheung and D. Shmoys. 2011. A primal-dual approximation algorithm for min-sum single-machine scheduling problems. In *Proceedings of the 14th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX'11)*, Vol. 6845. Springer, 135–146.

C. Chung, T. Nonner, and A. Souza. 2010. SRPT is 1.86-competitive for completion time scheduling. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'10)*. SIAM, 1373–1388.

J. Correa and M. Wagner. 2009. LP-based online scheduling: From single to parallel machines. *Math. Program.* 119 (2009), 109–136.

T. Ebenlendr, W. Jawor, and J. Sgall. 2009. Preemptive online scheduling: Optimal algorithms for all speeds. *Algorithmica* 53 (2009), 504–522.

T. Ebenlendr and J. Sgall. 2011. Semi-online preemptive scheduling: One algorithm for all variants. *Theory Comput. Syst.* 48, 3 (2011), 577–613.

L. Epstein, A. Levin, A. Marchetti-Spaccamela, N. Megow, J. Mestre, M. Skutella, and L. Stougie. 2012. Universal sequencing on an unreliable machine. *SIAM J. Comput.* 41, 3 (2012), 565–586.

L. Epstein and R. van Stee. 2003. Lower bounds for on-line single-machine scheduling. *Theoret. Comput. Sci.* 299 (2003), 439–450.

R. Fleischer and M. Wahl. 2000. Online scheduling revisited. In *Proceedings of the 8th Annual European Symposium on Algorithms (ESA'00)*, Mike Paterson (Ed.). Lecture Notes in Computer Science, Vol. 1879. Springer. Berlin, 202–210.

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

M. X. Goemans, M. Queyranne, A. S. Schulz, M. Skutella, and Y. Wang. 2002. Single machine scheduling with release dates. *SIAM J. Discr. Math.* 15 (2002), 165–192.

R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. 1979. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Ann. Discr. Math.* 5 (1979), 287–326.

L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. 1997. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Math. Operat. Res.* 22 (1997), 513–544.

D. S. Hochbaum and D. B. Shmoys. 1987. Using dual approximation algorithms for scheduling problems theoretical and practical results. *J. ACM* 34 (1987), 144–162.

D. S. Hochbaum and D. B. Shmoys. 1988. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM J. Comput.* 17 (1988), 539–551. DOI:http://dx.doi.org/10.1137/0217033

W. Höhn and T. Jacobs. 2012. On the performance of Smith's rule in single-machine scheduling with nonlinear cost. In *Proceedings of the 10th Latin American Symposium on Theoretical Informatics (LATIN)*. To appear.

H. Hoogeveen and A. P. A. Vestjens. 1996. Optimal on-line algorithms for single-machine scheduling. In *Proceedings of the 5th International Conference on Integer Programming and Combinatorial Optimization (IPCO) (LNCS)*, William H. Cunningham, S. Thomas McCormick, and Maurice Queyranne (Eds.), Vol. 1084. 404–414.

A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator. 1988. Competitive snoopy caching. *Algorithmica* 3 (1988), 79–119.

A. Kurpisz, M. Mastrolilli, and G. Stamoulis. 2012. Competitive ratio approximation schemes for makespan scheduling problems. In *Proceedings of the 10th Workshop on Approximation and Online Algorithms (WAOA 2012)*. Springer, 159–172.

J. Labetoulle, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. 1984. Preemptive scheduling of uniform machines subject to release dates. In *Progress in Combinatorial Optimization*. Academic Press Canada, 245–261.

E. L. Lawler and J. Labetoulle. 1978. On preemptive scheduling of unrelated parallel processors by linear programming. *J. ACM* 25, 4 (1978), 612–619.

J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker. 1977. Complexity of machine scheduling problems. *Ann. Discr. Math.* 1 (1977), 243–362.

J. K. Lenstra, D. B. Shmoys, and É. Tardos. 1990. Approximation algorithms for scheduling unrelated parallel machines. *Math. Program.* 46 (1990), 259–271. Issue 1. http://dx.doi.org/10.1007/BF01585745

P. Liu and X. Lu. 2009. On-line scheduling of parallel machines to minimize total completion times. *Comput. Operat. Res.* 36 (2009), 2647–2652.

X. Lu, R. A. Sitters, and L. Stougie. 2003. A class of on-line scheduling algorithms to minimize total completion time. *Operat. Res. Lett.* 31 (2003), 232–236.

C. Lund and N. Reingold. 1994. Linear programs for randomized on-line algorithms. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'94)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 382–391.

M. Manasse, L. McGeoch, and D. Sleator. 1988. Competitive algorithms for on-line problems. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC'88)*. ACM, 322–333.

R. McNaughton. 1959. Scheduling with deadlines and loss functions. *Manag. Sci.* 6, 1 (1959), 1–12.

N. Megow. 2007. *Coping with Incomplete Information in Scheduling—Stochastic and Online Models*. Ph.D. Dissertation. Technische Universität Berlin, Germany.

N. Megow and A. S. Schulz. 2004. On-line scheduling to minimize average completion time revisited. *Operat. Res. Lett.* 32 (2004), 485–490.

N. Megow and A. Wiese. 2013. Competitive-ratio approximation schemes for minimizing the makespan in the online-list model. *CoRR* abs/1303.1912.

J. Mestre and J. Verschae. 2014. A 4-approximation for scheduling on a single machine with general cost function. *CoRR* abs/1403.0298.

T. Mömke. 2013. A competitive ratio approximation scheme for the k-server problem in fixed finite metrics. *CoRR* abs/1303.2963.

C. A. Phillips, C. Stein, and J. Wein. 1998. Minimizing average completion time in the presence of release dates. *Math. Program.* 82 (1998), 199–223.

J. F. Rudin III and R. Chandrasekaran. 2003. Improved bounds for the online scheduling problem. *SIAM J. Comput.* 32 (2003), 717–735. DOI:http://dx.doi.org/10.1137/S0097539702403438

L. Schrage. 1968. A proof of the optimality of the shortest remaining processing time discipline. *Operat. Res.* 16 (1968), 687–690.

A. S. Schulz and M. Skutella. 2002a. The power of α-points in preemptive single machine scheduling. *J. Sched.* 5 (2002), 121–133.

A. S. Schulz and M. Skutella. 2002b. Scheduling unrelated machines by randomized rounding. *SIAM J. Discr. Math.* 15 (2002), 450–469.

S. S. Seiden. 2000. A guessing game and randomized online algorithms. In *Proceedings of the 32nd ACM Symposium on the Theory of Computing (STOC'00)*. 592–601.

J. Sethuraman and M. S. Squillante. 1999. Optimal scheduling of multiclass parallel machines. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'99)*. SIAM, 963–964.

R. Sitters. 2010a. Competitive analysis of preemptive single-machine scheduling. *Operat. Res. Lett.* 38 (2010), 585–588.

R. Sitters. 2010b. Efficient algorithms for average completion time scheduling. In *Proceedings of the 14th International Conference on Integer Programming and Combinatorial Optimization (IPCO'10)*, Vol. 6080. Springer, 411–423.

M. Skutella. 2001. Convex quadratic and semidefinite programming relaxations in scheduling. *J. ACM* 48 (2001), 206–242.

D. D. Sleator and R. E. Tarjan. 1985. Amortized efficiency of list update and paging rules. *Commun. ACM* 28 (1985), 202–208.

W. E. Smith. 1956. Various optimizers for single-stage production. *Nav. Res. Logist. Quart.* 3 (1956), 59–66.

L. Stougie and A. P. A. Vestjens. 2002. Randomized algorithms for on-line scheduling problems: How low can't you go? *Operat. Res. Lett.* 30 (2002), 89–96.

A. P. A. Vestjens. 1997. *On-line Machine Scheduling*. Ph.D. Dissertation. Eindhoven University of Technology, Netherlands.

Received July 2015; revised August 2016; accepted September 2016