An EPTAS for Minimizing the Total Weighted Completion Time of Jobs with Release Dates on Uniformly Related Machines

Leah Epstein
Department of Mathematics, Faculty of Natural Sciences, University of Haifa, Israel

Asaf Levin
Faculty of Data and Decision Sciences, Technion, Haifa, Israel

Abstract

Scheduling of independent jobs with release dates so as to minimize the total weighted completion time is a well-known scheduling problem. Here, we study it for the classic machine environment of uniformly related machines. An efficient polynomial time approximation scheme (an EPTAS) is a family of $(1+\varepsilon)$-approximation algorithms where the running time is bounded by a polynomial in the input size times a function of $\varepsilon > 0$. For problems that are NP-hard in the strong sense, as it is the case for the problem studied here, an EPTAS is the best possible approximation scheme. We design an EPTAS for the problem by employing known techniques and introducing a large collection of new methods.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms

Keywords and phrases Scheduling algorithms, Approximation schemes, Min-sum objectives

Digital Object Identifier 10.4230/LIPIcs.MFCS.2025.44

Funding *Asaf Levin*: Partially supported by ISF - Israel Science Foundation grant number 1467/22.

1 Introduction

In this work, we study scheduling on uniformly related machines with release dates, which is a standard multiprocessor scheduling problem. Here, the objective is to minimize the sum of weighted completion times of jobs, also called the total weighted completion time. In this problem, we are given a set of n jobs. Each job has a positive size and a positive weight, and a non-negative release date (also called release time) associated with it. There is a set of m parallel machines, such that each machine has a positive speed, and the jobs are to be assigned to the machines. We consider non-preemptive schedules, where every job is assigned to a machine and scheduled to a single (continuous) time slot (or interval) on that machine, such that the length of this interval with respect to time is equal to the processing time of j on the machine that received it. The processing time of j on machine i is the size of j divided by the speed of i.

The assignment has to fulfill the following requirements. First, as it is already explained above, the length of the time interval assigned to job j (on one of the machines) is the time required for processing j on that machine. Second, the time interval assigned to job j has a starting point that is no earlier than the release date of j. Finally, a machine cannot run more than one job at each time, and therefore we require that the time intervals allocated for two jobs on a common machine will intersect only at endpoints. We are interested in schedules of this structure, sometimes called valid schedules. The completion time of job j is equal to the right endpoint of the time interval of job j. The weighted completion time of j is defined by multiplying the weight of j by its completion time. In this algorithmic problem, the goal is to find a (valid) schedule for which the sum of the weighted completion times

over all jobs is minimized, that is, the goal is minimization of the total weighted completion time. We use the term density of a job which is defined as the ratio between its weight and its size. For previous work on this problem, see [26, 5, 28, 1, 8, 12, 29, 20].

In order to explain our result, we start with discussing concepts of approximation algorithms and the concepts of the main types of approximation schemes. We focus on minimization problems and define all concepts for them. An $\mathcal{R}$-approximation algorithm is a polynomial time algorithm that computes a solution whose cost never exceeds $\mathcal{R}$ multiplied by the cost of an optimal solution for the same input. The smallest value or the infimum value of $\mathcal{R}$ for which a given algorithm is an $\mathcal{R}$-approximation is called the approximation ratio of the algorithm. A polynomial time approximation scheme (a PTAS) is a class of approximation algorithms containing a $(1+\varepsilon)$-approximation algorithm for any value $\varepsilon > 0$. In the running time of a PTAS the parameter ε is seen as a constant. An efficient polynomial time approximation scheme (an EPTAS) is a PTAS whose running time complexity has the form $f(\frac{1}{\varepsilon})$ multiplied by a polynomial function of the length of the (binary) encoding of the input. The difference with a PTAS is that in an EPTAS ε (or $\frac{1}{\varepsilon}$) cannot appear in the exponent of n. A fully polynomial time approximation scheme (an FPTAS) is an EPTAS for which the function f is polynomial in $\frac{1}{\varepsilon}$. In this work, we are interested in EPTAS's. This intermediate concept of an EPTAS has relation to the FPT (fixed parameter tractable) literature (see for example [7, 20, 22, 21, 23, 25]). It can be argued that FPTAS's can be used in practice while PTAS's (with small values of ε) are too slow. On the other hand, many scheduling problems cannot have FPTAS's unless P=NP, which gave rise to the design of EPTAS's for such problems. In this paper, we design an EPTAS for the scheduling problem defined above. This is one of the well-motivated scheduling problems for which an FPTAS cannot be found (unless P=NP).

For a single machine without release dates, the problem of scheduling jobs so as to minimize the total weighted completion time is well understood. Smith [31] showed that sorting jobs by non-increasing densities and scheduling them in this order without any idle time starting from time zero gives an optimal solution. This sorted order is sometimes called Smith's ratio, and a trivial exchange argument implies its optimality. We can use this argument for a specific machine that already received its jobs, and it is required to schedule jobs to time slots after the last release time. Even the problem without release dates and $m \geq 2$ machines is NP-hard. Specifically, for constant m, it is NP-hard in the ordinary sense, and the variant where the number of machines is a part of the input is strongly NP-hard [14]. Interestingly, there are polynomially solvable special cases, where one example is the case of equal weights without release dates [18]. Problems with release dates are known to be much harder than those where all jobs are available at time zero, and they are strongly NP-hard for any constant number of machines, even with equal weights. Already the problem with release dates on a single machine and equal weights is strongly NP-hard [27]. The property that our problem is strongly NP-hard excludes the possibility to design an FPTAS for the problem that we consider in this work. Thus, as explained above, since the assumption P $\neq$ NP is standard, EPTAS's are probably the best possible results.

Approximation algorithms for the problem studied here and its special cases were found in a relatively late stage. Till twenty-five year ago there were only approximation algorithms for which the approximation ratio was a constant for min-sum scheduling problems as the one that we study (for information on these results see [30, 1, 8]). We discuss approximation schemes for our problem and its special cases in more detail. A PTAS which is also an EPTAS for identical machines without release dates was provided by Skutella and Woeginger [30]. Afrati et al. [1] designed PTAS's for identical machines with release dates, for the

problem with a constant number of unrelated machines (also with release dates), and for uniformly related machines (without release dates). Chekuri and Khanna [8] developed a PTAS for the problem studied here. That is, they studied the case of uniformly related machines with release dates, where the number of machines is not seen as a constant (see also [2]). The schemes of [1, 8] for non-constant numbers of machines are not EPTAS's, and it is not possible to modify them into EPTAS's (using known methods). Recently, we developed an EPTAS for the model of uniformly related machines without release dates [13].

Approximation schemes for load balancing problems (like the makespan objective) on identical machines and uniformly related machines were studied as well in the past. Load balancing problems deal with minimizing functions of completion times vectors of machines (and not on job completion times, which we study here). Hochbaum and Shmoys presented the dual approximation framework and used it to show that the makespan minimization problem (the maximum of the machine completion times vector) has a PTAS for identical machines and for uniformly related machines [16, 17, 15] (for identical machines the PTAS is in fact an EPTAS). Jansen [19] designed an EPTAS for the makespan minimization problem on uniformly related machines (see [6] for an alternative EPTAS for this problem). The ℓ_p norm minimization problem (of the vector of machine completion times) has an EPTAS for identical machines [3, 4], and a PTAS and an EPTAS for uniformly related machines [9, 10, 24]. This last objective for the case $p = 2$ is equivalent to our objective in the special case where the weight of each job is simply its size, which was noted by Skutella and Woeginger [30].

Previous directions. The equivalence for identical machines noted in [30] (which holds also for the case with release dates) follows from the next property. An optimal solution to one problem is also an optimal solution to the other problem, since the two values of the objective functions differ by an additive constant which is common to all feasible solutions, where this constant depends on the input. Using this idea, they proved that in the case where the ratio between the maximum density and the minimum density of input jobs is not larger than a constant, it is possible to convert methods of Alon et al. [3, 4] to obtain an EPTAS for this restricted setting. In [13], we have used the approach of [30] and we developed new approaches (used later by [24] for the ℓ_p norm minimization problem on uniformly related machines) in order to present an EPTAS for the problem without release dates. These ideas fail to work when release dates are present. Afrati et al. [1] were successful in overcoming difficulties in the design of approximation schemes by rounding the input parameters and then structuring the input in a way that a job is processed promptly after its release date. Using this structure, each machine has a constant number of states (where a state of a machine is the next time in which it becomes available for processing another job), and one can apply dynamic programming on the time-horizon to schedule the jobs while keeping track of the number of machines in each state, and the number of unscheduled jobs of each large size that were released at any given time in the last few release dates (and similarly, for the total volume of unscheduled jobs seen as small that were released in those last few release dates). Chekuri and Khanna [8] extended these techniques of [1] to the setting of uniformly related machines (with release dates). The time complexity of the time-horizon dynamic programming is too large if one seeks for an EPTAS due to the following. The number of possible states of machines is clearly a constant dependent on ε, and we need to recall the number of machines in each state, and thus the degree of m in the polynomial of the running time depends on ε, which violates the conditions on the running time of an EPTAS. Thus, the methods of [1, 8] fail completely when one tries to obtain an EPTAS for these problems with release dates (even for identical machines).

Paper outline. In the main part of the paper we prove our result which is an EPTAS for the non-preemptive scheduling problem with release dates on uniformly related machines so as to minimize the total weighted completion time of the jobs. Before presenting the detailed description of the scheme, we present an overview. Omitted proofs can be found in the full version of this work.

2 An Overview of the EPTAS

In a simpler EPTAS that we designed for the special case without release dates [11, 13], the instance was split into separate job sets based on densities. Specifically, every instance consisted of all machines together with a subset of jobs of a bounded class of densities. This turned out to be a useful strategy for the purpose of obtaining an EPTAS. Here, we will additionally need to require that the number of distinct values of release dates in each such bounded instance is a constant, which complicates the partition and the solution to different parts of the partition have to be synchronized carefully. After applying the partition, it is fairly easy to obtain an EPTAS that can be used for such instances.

Thus, the approach that we will take is to transform and reduce the original instance multiple times, using a variety of methods and tricks, until we obtain sufficiently simpler instances for which we will be able to employ the configuration based MILP method. This method is only used for the final instances. That is, we are interested in reducing the problem into a polynomial set of subproblems, where each subproblem which we create can be solved using a mixed-integer linear program (MILP). The solutions of the MILP's need to be coordinated using global information that we are able to guess. We will round the solutions of these MILP's, and then we will be able to integrate the different solutions for the different instances without increasing the total cost of the solutions to these MILP's significantly.

We start the scheme with rounding the input parameters. We round job sizes, job weights, and speeds of machines to integer powers of $1+\delta$ (where δ depends on ε). For release dates there are two rounding steps. First, we slightly increase the release dates of some jobs so a job is released no earlier than δ times its minimum processing time (the processing time that it would have on the fastest machine), and later we round the (modified) release dates up to integer powers of $1+\delta$. In particular, for such a rounded instance, no release date will be equal to zero. Release dates are never rounded down, so that a job cannot start running before its release date even if the schedule is constructed based on the rounded release dates. This type of rounding of the input parameters was done also in [1, 8], and nowadays it is considered as a standard tool.

Next, we use an iterative procedure, which we call *job shifting*, where we delay the release dates of some of the jobs. More specifically, we consider situations where the total number of jobs (for relatively large jobs that have similar properties) or the total size of jobs (for sub-classes of relatively small jobs that have similar properties) that are being released at a given release date is too large compared to the amount that can run until the next possible release date (where possible release dates are powers of $1+\delta$ that are not smaller than the first such release date in the input). When it is impossible to schedule such a large number or such a large total size of those jobs to start running by the next possible release date, the release date of some jobs can be altered without any damage to the set of possible schedules, since large inputs contain many pairs and subsets of interchangeable jobs. We introduce a suitable step, and show that as a result the following property will hold for the modified input. This property is that the set of jobs of a common density whose release dates are

all equal to T (for some value of T) can be scheduled on the set of machines such that they will be completed no later than a constant multiplied by T. Such a step was designed for identical machines in [1], where the output of the corresponding step satisfies a stronger property. Namely, the bound on the maximum completion times of jobs released at time T is defined for all jobs with this release time and not only for jobs of a common density.

We could certainly use the procedure of [1] if we were dealing only with identical machines, but once speeds are present, the situation becomes fairly complicated. A similar step was claimed in [8] for uniformly related machines. Unfortunately, the case of uniformly related machines is significantly more difficult because in the exchange argument for uniformly related machines, a small job that is delayed to a later time interval need not remain small, because the machine that will receive it as a result of an exchange could be slower. This difficulty was not observed by Chekuri and Khanna [8], and thus the description and analysis of their job shifting procedure as it has appeared in the conference proceeding version has significant gaps. Therefore, in order to overcome this difficulty, we provide a new procedure that handles each density separately. Our procedure uses a fine partition of the jobs of each density, and we can prove the required bound (per density). This fine partition of the job set is according to their sizes: we consider a constant number of divisible sequences of sizes. For each such subsequence and for each speed and density we do as follows. We keep the largest jobs of this density out of the jobs of the subsequence that are still considered to be small for machines of this speed, and we also keep some jobs that are considered to be large on machines with that speed. Keeping jobs means that their release dates are unchanged in this phase of the algorithm. If a job is not "kept" (after we process all machines), we postpone its release date (we move its release date to a later time which is the next integer power of $1+\delta$). The property of divisible sequences that we use in the exchange argument is that if a job j is large during some future time interval, and we need to schedule it earlier because the jobs scheduled in the optimal solution are postponed, we can pick a subset of these postponed jobs whose total size is *exactly* the size of j (with one possible exception per machine, density, and subsequence), and swap the positions of the larger job and the set of selected smaller jobs. Applying such transformations on all densities require additional considerations. The reason that our bound on the job set with a given release date is per density is that we do not know in advance if we want to prioritize (among small jobs for the current speed) larger jobs with smaller density or smaller jobs with higher density. These two subsets of jobs are incomparable. The existence of those incomparable sets is exactly the point missed in [8].

While in these first two steps we used the approach of [1, 8], in the main part of our scheme we deviate dramatically from previous approaches by providing original methods to face our problem. These new methods are needed even if we were able to mimic the shifting claim of [8].

Shifting steps. We apply two different shifting steps. The first shifting step is on the release dates. In this step we increase release dates of a small portion of the jobs (the amount of jobs is selected based on contribution to cost), but the increase is by a very large (constant) multiplicative factor. The goal of this increase of some release dates is to create large time differences between groups of release dates, so that sub-inputs of very different release dates could be solved almost independently, though they are not really independent instances. Our bound on the total size of jobs with a common release date is per density, and it is often not really possible to schedule all jobs quickly after their release dates. Thus, it can happen that jobs of much smaller release dates should be combined into a solution for later

release dates. In these cases, the assignment of such jobs can be restricted to gaps of the schedule (a specific type of idle time), rather than treating these jobs similarly to jobs whose release dates are recent. By slightly stretching time [1], there are suitable gaps with sufficient frequency in any schedule. Since we have an upper bound on the total size of jobs with a common density, and because we will solve each sub-input (of all jobs with similar release dates after this shifting step) by solving a series of bounded instances (with bounded ratios of densities), we will maintain a property that after the last release date of a sub-input, the density of jobs assigned to a common machine deteriorates geometrically (along time). Thus, postponing jobs by some constant multiplicative factor can be charged to the total cost of the earlier jobs on that machine (in the solution to the sub-input) that have much higher densities than the postponed jobs, and it does not increase the cost significantly. Therefore, in time intervals where the solutions to different sub-inputs contradict (i.e., two or more solutions try to schedule jobs at the same time) the jobs corresponding to earlier sub-inputs will be delayed to the gaps of the latest sub-input (among the gaps of the sub-inputs with the contradicting solutions) and this delay is charged to the jobs of these earlier sub-inputs that are not delayed.

Later, we will apply *shifting on densities*. The goal is to create inputs with small numbers of possible parameters, such that there is a major difference between parameters of subproblems. Unlike the variant without release dates where solutions of different densities were concatenated, which was possible due to large differences between the sets of densities for subproblems (see [11, 13]), combining solutions of very different densities is extremely challenging in the case with release dates that we study here. Jobs cannot run before their release dates, and if one solution is sparse in some time interval (a large part of this interval consists of idle time), we would like another solution to take over, in order to exploit that time, and to avoid postponing all jobs (which could result in increasing their costs significantly). Yet, that solution is still not sufficiently good in some cases, as the schedule of jobs with very high densities must be done carefully, and they cannot be delayed (compared to their schedule in optimal solutions) in any solution (as such a delay may cause a major increase in the total cost). On the other hand, it harms the solution to delay multiple jobs (even if their densities are not very large) due to a sparse time slot (with high density jobs). Thus, the schedule of different subproblems needs to be coordinated, and these subproblems must be taken care of delicately. To overcome this notoriously difficult task, we apply a series of guessing steps and transformations on the solutions to ensure that these bad cases simply do not happen. Intuitively, we will ensure that for every pair of solutions, every relatively fast machine, and every time interval, the following will hold. If one solution is a sparse solution during a given time interval (in which case the solution cannot be charged for postponing starting times of other solutions till later), the second solution will be such that no job occupies (at least) the entire time interval. We employ gaps in schedules to combine unassigned jobs, and to allow the application of modifications of solutions.

More precisely, given a solution to a sub-input (resulting from the step of release dates shifting) we define a *color* of a machine to be the subset of intervals between consecutive times of release dates (of jobs of that sub-input) in which the machine has either some idle time or a starting time of at least one job. Using time stretching, we have small gaps in this time interval in each one of the two cases. A machine is a *pink machine* if its color is the set of all intervals between consecutive times that are release dates. A palette of a schedule is the vector of colors of its (constant number of) its fastest machines. We show that we can restrict ourselves to solutions that have (at least one) pink machine among their fastest machines, and we guess the palette of a near optimal solution with the property that it has

a pink machine (observe that the number of possible palettes is a constant and thus we can emulate this guessing step via exhaustive enumeration). In what follows we will restrict ourselves to solutions corresponding to this guessed palette. This guessed palette will allow us to coordinate the different subproblems resulting from the shifting on the densities step as we discuss next.

After the step of guessing of the palette, we apply the shifting method on the densities, in order to partition the sub-input of similar release dates into several subproblems with the property that in each such subproblem the ratio between the maximum density of a job and the minimum density of a job is bounded from above by a constant (and by the rounding step, this means that the number of distinct densities in such subproblem is a constant), and for two jobs belonging to distinct subproblems, the two densities are not close. Using the existence of a pink machine, we show that we can restrict ourselves further to solutions satisfying that for every machine (which is not one of the constant number of fastest machines) and every time interval between consecutive release dates, either the machine is completely idle, or it is busy for a large portion of the time interval (and thus can be charged for postponing jobs of subproblems with much lower density after the last release date of this sub-input), and we say that this machine has no *sparse interval*. Therefore, in such solutions for every machine and every time interval, either no solution of a subproblem has a sparse interval, or all of them have either gaps or starting times in this time interval (this can happen only on a constant number of fast machines). In this way, we solve the issue of sparse versus fully occupied time intervals.

Then, we can build a solution for a sub-input from the solutions to its subproblems by processing one machine at a time, and this is carried out by processing the solutions from higher densities subproblems to lower densities subproblems. The embedding of one solution in another solution is done by uniting the set of jobs starting at a given time interval greedily until there is no space for additional jobs (the remaining jobs are postponed to later time intervals). When a time interval has no space for additional jobs, it is not a sparse time interval, and it can be charged for postponing the jobs of lower densities to later starting time (due to the large difference in densities between subproblems, which was achieved by shifting).

The next step processes the job set of each such subproblem, and in cases where there is a release date for which the set of jobs with this release date can be processed on one of the (constant number of) fastest machines without consuming too much of its processing time (and without violating its color), then we do that and remove this set of scheduled jobs from the subproblem. This step has a minor impact on the cost of the resulting solution, and it will help us to bound the additional cost due to rounding of the solution for the MILP in the last step. Finally, we use the *mixed-integer linear program (MILP) paradigm* to approximate each subproblem. Here, we use time-dependent configurations. Each such time-dependent configuration encodes the speed of the machine (and its index if it is one of the machines whose color is encoded in the palette), and defines the set of large jobs (or an approximated total size of small jobs) of every given density that starts running in every time interval. Since we consider instances with a constant number of release dates and a constant number of densities, we get that for every speed, the number of configurations with this encoded speed is a constant. In the MILP, we will require the variables of configurations of the (constant number of) fastest machines to be integral as well as configurations with a constant number of speeds (of speeds close to those of the fastest machines). All other variables are fractional, and thus the number of integral variables will be a constant.

The rounding of the solution for the MILP (to obtain an actual schedule) is carried out in several steps. The first one is similar to the case without release dates [11, 13] (in which every machine is slightly overused by increasing the total length of small jobs assigned to it), and schedule all the remaining jobs (there are such jobs due to rounding down the variables of machine configurations corresponding to slow machines) to start in the same time interval on the pink machine. This will be possible by time stretching of the schedule on that machine using the large ratio between the speed of the pink machine and the machines whose configuration variables are allowed to be fractional.

We argue that the rounded solution of the MILP satisfies the property that the maximum load of any machine is bounded from above by a constant multiplied by the maximum release date. In turn, this means that the solution obtained for each sub-input (with a bounded number of release dates but with an unbounded number of densities) satisfies that the densities of jobs deteriorate geometrically. This is exactly the property that we use to bound the impact of uniting the solutions to the sub-inputs into one solution.

We provide further details regarding the approximation scheme in what follows. A complete version of the description of the approximation scheme explained here is given in the full version of this work.

3 An EPTAS

We design an EPTAS for our problem. Here, in the original instance A, a job j has a size $a_j > 0$ and a release time $\rho_j \geq 0$ associated with it, such that j cannot be executed before time $\rho_j > 0$. The (strictly positive) speed of machine i is denoted by v_i. A solution or a schedule is not only a partition of the jobs among the machines, but it also states the completion time of each job, which is sufficient, since the schedule is non-preemptive (an alternative way to define a schedule is by defining the starting times, but we will use the former option). A schedule (or a valid schedule) must comply with the property that no machine runs more than one job at a time, except for times when one job completes its processing and another one starts its processing. In order to avoid this special case, we will assume in what follows that the time slot that a job runs on a machine is half open, that is, if j runs on machine i, its time slot is $[C_j - \frac{a_j}{v_i}, C_j)$. Note that an optimal schedule may have idle times due to the presence of release times. Using scaling (sorting the machines and dividing all speeds and sizes by the largest machine speed), we assume that $1 = v_1 \geq v_2 \geq \cdots \geq v_m > 0$. Let $0 < \delta \leq \frac{1}{36}$ be an accuracy factor, which is a function of ε, where $\delta < \varepsilon$ and such that $\frac{1}{\delta}$ is an integer.

Standard rounding steps. As a first step, we convert input A (which consists of m machines with their speeds and n jobs with their attributes) into a rounded instance A'. Let $a_{\min} = \min_{1 \leq j \leq n} a_j$. Let $\beta = \delta^2 \cdot a_{\min}$. The sets of jobs and machines are the same as in A (possibly with different attributes). Let $r'_j = (1+\delta)^{\lceil \log_{1+\delta}(\rho_j + \beta) \rceil}$ be the release date of job j. Note that $\rho_j + \beta > 0$, so r'_j is well-defined for any j (even if $\rho_j = 0$) and $r'_j \geq \rho_j + \beta > \rho_j$. Let $s_i = (1+\delta)^{\lfloor \log_{1+\delta} v_i \rfloor} \leq v_i$ be the speed of machine i in instance A', where $s_i > 0$ since $v_i > 0$. Let $w_j = (1+\delta)^{\lceil \log_{1+\delta} \omega_j \rceil} \geq \omega_j$ and $p_j = (1+\delta)^{\lceil \log_{1+\delta} a_j \rceil} \geq a_j$ be the weight and size (respectively) of job j in instance A'. It will also hold that $r'_j \leq (1+\delta) \cdot (\rho_j + \beta)$, $s_i \geq \frac{v_i}{1+\delta}$, $w_j \leq (1+\delta) \cdot \omega_j$, and $p_j \leq (1+\delta) \cdot a_j$. Using the new notation, for a given schedule, the completion time of j is still denoted by C_j, and its weighted completion time is $w_j \cdot C_j$. Note that $1 = s_1 \geq s_2 > \cdots \geq s_m > 0$ (the sorted order of machines is preserved and the largest speed is unchanged). Let $\mathcal{O}$ and $\mathcal{O}'$ denote optimal solutions for A and A' respectively.

▶ **Lemma 1.** *Every solution SOL for A' is also a solution for A, and $SOL(A) \leq SOL(A')$.*

Given a schedule (a solution) SOL defined for a given input, let the *time-augmented solution* $TA(SOL, \upsilon)$ for some $\upsilon > 1$ and the same input (the same set of jobs and the same set of machines) be the solution where each job is assigned to the same machine as in SOL, and the *completion time* of each job is υ times its completion time in SOL. In the next lemma we show that if SOL is a solution for A, then $TA(SOL, \upsilon)$ is a solution for A as well.

A schedule SOL'' for A' is called *timely* if for every job j, the starting time of j is at least $(1+\delta)^{\lceil \log_{1+\delta}(\delta \cdot \frac{p_j}{s_i}) \rceil}$, where i is the machine that runs j in SOL''. In particular, the starting time of j in a timely schedule is at least $\delta \cdot \frac{p_j}{s_i}$. To prove that SOL'' is timely, it is sufficient to prove that for every job j, the starting time of j in SOL'' is at least $\delta \cdot (1+\delta) \cdot \frac{p_j}{s_i}$.

Given a solution SOL for A, we will be able to use $TA(SOL, \upsilon)$ as a solution for the rounded instance A' for certain values of υ. The next lemma specifies cases such that this is indeed a valid schedule for A', and moreover it is a timely schedule. This condition on υ will be a sufficient one.

▶ **Lemma 2.** *Given a schedule SOL for A and $\upsilon > 1$, let $SOL' = TA(SOL, \upsilon)$. The schedule SOL' is a schedule for A such that $SOL'(A) = \upsilon \cdot SOL(A)$. If $\upsilon \geq (1+\delta)^4$, then SOL' is a schedule for A', and it is timely.*

In what follows, when we are given a solution SOL for A, we let $S\bar{O}L = TA(SOL, (1+\delta)^4)$. We use $S\bar{O}L$ as a solution for A' and not only for A.

▶ **Lemma 3.** *We have $(1+\delta)^4 \cdot SOL(A) \leq S\bar{O}L(A') \leq (1+\delta)^5 \cdot SOL(A)$, $\bar{\mathcal{O}}(A') \leq (1+\delta)^5 \cdot \mathcal{O}(A)$, and $\mathcal{O}(A) \leq \mathcal{O}'(A') \leq (1+\delta)^5 \cdot \mathcal{O}(A)$.*

In what follows, we will only consider timely (and valid) schedules for A'. Let $\mathcal{O}''$ denote an optimal timely schedule for A'.

▶ **Lemma 4.** *We have $\mathcal{O}''(A') \leq (1+\delta)^5 \cdot \mathcal{O}'(A')$.*

▶ **Corollary 5.** *If a solution SOL for A' satisfies $SOL(A') \leq (1+k\delta) \cdot \mathcal{O}''(A')$ for some $k \geq 1$, then $SOL(A) \leq (1+(3k/2+15)\delta) \cdot \mathcal{O}(A)$.*

In the analysis of algorithms for A', we will use *pseudo-costs* which we will define now, rather than actual costs. The completion time of any job is strictly positive, and we use this property in the definition. The pseudo-cost of job j whose completion time C_j satisfies $C_j \in [(1+\delta)^i, (1+\delta)^{i+1})$ is defined as $w_j(1+\delta)^{i+1}$ (that is, we round its completion time up to the next power of $1+\delta$ for the calculation of its pseudo-cost such that the completion time is increased even if it is already an integer power of $1+\delta$). Let $\mathcal{O}_{PC}$ denote an optimal timely solution for A' with respect to pseudo-cost (that is, the total pseudo-cost of all jobs is minimized). Let $PC(SOL(A'))$ denote the pseudo-cost of SOL for input A'.

▶ **Lemma 6.** *For a solution SOL for input A', we have $\frac{PC(SOL(A'))}{1+\delta} \leq SOL(A') \leq PC(SOL(A'))$, and $PC(\mathcal{O}_{PC}(A')) \leq PC(\mathcal{O}''(A')) \leq (1+\delta) \cdot \mathcal{O}''(A')$.*

If SOL satisfies $PC(SOL(A')) \leq (1+k' \cdot \delta) \cdot PC(\mathcal{O}_{PC}(A'))$ for some $k' \geq 1$, then $SOL(A) \leq (1+(2k'+17) \cdot \delta) \cdot \mathcal{O}(A)$.

Given the last lemma we find that in order to obtain an approximate (within a factor of $1+O(\varepsilon)$) schedule for A with respect to the total weighted completion time, it is sufficient to consider timely schedules and pseudo-costs, and find an approximate schedule (within a factor of $1+O(\varepsilon)$) for the latter problem.

Preliminaries. Before presenting the next steps of our scheme, we define some procedures for modifying a solution. These procedures that we develop here will be invoked throughout the further steps of our algorithm.

A simple representation of schedules. We consider input A' and timely schedules. In what follows we only discuss pseudo-costs. As we saw, a resulting approximate solution can be used as an approximate solution for A.

We slightly abuse notation and use $SOL(A')$ to denote the *pseudo-cost* of SOL for A'. Let $\mathcal{J}_{i,\ell}$ denote the time interval $[(1+\delta)^i, (1+\delta)^{i+1})$ on machine ℓ. The total size of jobs for which both the starting time and completion time are within this interval (on machine ℓ) is at most $s_\ell \cdot \delta \cdot (1+\delta)^i$, which is called the *length* of this interval. In what follows we let θ denote the smallest value of i such that $(1+\delta)^\theta$ is a release date of some job of A'.

▶ **Lemma 7.** *Consider a timely schedule. If job j is assigned to run on machine ℓ and has a starting time in $\mathcal{J}_{i,\ell}$, then $p_j < \frac{s_\ell}{\delta} \cdot (1+\delta)^{i+1}$. If the completion time of j is in the same interval, then $p_j < s_\ell \cdot \delta \cdot (1+\delta)^i$.*

Since we use pseudo-costs, we will represent schedules by specifying for each job j the machine that runs it, the interval where j starts, and the interval that contains the completion time of the job (both being time intervals of the same machine where for a time interval, both the start point and the endpoint are integer powers of $1+\delta$). Note that if the completion time of j is $(1+\delta)^i$ (on some machine ℓ), which is the right endpoint of the previous interval and the start endpoint of $\mathcal{J}_{i,\ell}$, then we say that the completion time of j belongs to $J_{i,\ell}$ even though the time slot that j runs in is $[(1+\delta)^i - \frac{p_j}{s_i}, (1+\delta)^i)$ (so j is completed just before $\mathcal{J}_{i,\ell}$ starts, but the completion time of j is $(1+\delta)^i$, and this is a point of $\mathcal{J}_{i,\ell}$). Alternatively, it is possible to state, for each interval, the list of jobs whose starting times are in this interval, and the list of jobs whose completion times are in this interval. The cost of the schedule can be computed by computing the total cost of all intervals. The cost of $\mathcal{J}_{i,\ell}$ is $(1+\delta)^{i+1}$ times the total weight of jobs whose completion times are in $\mathcal{J}_{i,\ell}$. Obviously, additional conditions are required for such lists to ensure that a list originates in a valid schedule (where an exact completion time is assigned to each job), and a complete schedule of the same cost (i.e., pseudo-cost) can be constructed. For example, if job j has a starting time in $\mathcal{J}_{i_1,\ell}$ and completion time in $\mathcal{J}_{i_2,\ell}$, where $i_2 - i_1 \geq 2$, then for any $i_1 < i' < i_2$, no job can have a starting time or a completion time in $\mathcal{J}_{i',\ell}$. Moreover, the total size of jobs that have to run in a sequence of consecutive intervals cannot exceed the total length of the intervals. We say that a schedule defined by such a list is timely if for every job j with starting time in $\mathcal{J}_{i,\ell}$ it holds that $\delta \cdot \frac{p_j}{s_\ell} \leq (1+\delta)^i$, that is, if the minimum starting time that it may receive satisfies the condition for timely schedules for this job. This is equivalent to the original definition, since the thresholds for starting times in that definition are integer powers of $1+\delta$. In the next lemma we formulate necessary and sufficient conditions for a list to represent a valid timely schedule.

▶ **Lemma 8.** *Consider the following conditions on a list.*

1. *For every job j, that has a starting time in $\mathcal{J}_{i,\ell}$ and a completion time in $\mathcal{J}_{i',\ell'}$, it holds that $\ell' = \ell$, $i' \geq i$, and $r'_j \leq (1+\delta)^i$.*
2. *For any $1 \leq \ell \leq m$, $i \geq \theta$ (where $(1+\delta)^\theta$ was defined as the minimum release date in A'), and θ' such that $\theta \leq \theta' \leq i$, the total size of jobs for which both the completion times and starting times are in $\mathcal{J}_{\theta',\ell} \cup \mathcal{J}_{\theta'+1,\ell} \cup \cdots \cup \mathcal{J}_{i,\ell}$ is strictly smaller than the total length of the intervals $\mathcal{J}_{\theta',\ell}, \mathcal{J}_{\theta'+1,\ell}, \ldots, \mathcal{J}_{i,\ell}$.*

3. *For any j whose starting time and completion time are in the intervals $\mathcal{J}_{i_1,\ell}$ and $\mathcal{J}_{i_2,\ell}$ respectively, such that $i_2 > i_1$, any interval $\mathcal{J}_{i',\ell}$ with $i_1 < i' < i_2$ has no starting times of jobs and no completion times of jobs.*
4. *For any $\mathcal{J}_{i,\ell}$, there is at most one job whose starting time is in this interval and its completion time is not, and at most one job whose completion time is in this interval, but its starting time is not.*

Every schedule has these properties, and for every list that has these properties, there exists a schedule of the same cost that obeys the requirements on starting times and completion times of jobs of this list in the sense that the schedule is valid, starting times are in the required intervals and completion times are no later than the required intervals (and every job is assigned to the machine that should receive it).

We will have that for every timely schedule that is given as a list, every schedule that is created from the list (i.e., choosing a permutation of the jobs that start and complete in a common interval) results in a timely schedule.

Shifted schedules. For a job j of size $(1+\delta)^i$, let its stretched size be $(1+\delta)^{i+1}$. We define a stretched input $\bar{A}'$ where the size of each job is stretched and its processing time is exactly $1+\delta$ times larger. The other properties of the stretched input are unchanged compared to the original input in the sense that it consists of the same machines and jobs as A', each machine has the same speed in both inputs, and each job has the same release date in both inputs.

Given a schedule SOL for A', we define a schedule $S(SOL)$, called the *shifted* schedule of the schedule SOL, which is defined for $\bar{A}'$. If job j has a starting time in $\mathcal{J}_{i_1,\ell}$ and completion time in $\mathcal{J}_{i_2,\ell}$ (where $i_2 \geq i_1$) in SOL, we define its starting time and its completion time in $S(SOL)$ to be in the intervals $\mathcal{J}_{i_1+1,\ell}$ and $\mathcal{J}_{i_2+1,\ell}$, respectively.

▶ **Lemma 9.** *If SOL is a valid timely schedule (for A'), then so is $S(SOL)$ (for $\bar{A}'$), and the costs satisfy $S(SOL)(\bar{A}') = (1+\delta)SOL(A')$.*

Schedule $S(SOL)$ can obviously be used also as a schedule for an input where the size of each job in A' is stretched by some factor in $[1, 1+\delta]$ (the need to use this property is one of the reasons that we consider $S(SOL)$ as an assignment of a machine and completion time for each job), and it remains timely since jobs can only start later while their sizes can only decrease. This argument is valid even if the stretch factors of different jobs may be different, and also if all stretch factors are equal to 1 (that is, the input is simply A').

Time stretched schedules. Given a timely schedule $\hat{S}$ for an input $\bar{A}'$, we do not modify the input, but we define a new schedule called the *schedule obtained from $\hat{S}$ by time stretching by a factor of* $1+\delta$, as follows. If j is assigned to run (in $\hat{S}$) on machine ℓ during $[t, t')$, we obviously have $p_j = s_\ell(t'-t)$. We reserve the time period $[(1+\delta)t, (1+\delta)t')$ on machine ℓ for job j (where $(1+\delta)t$ is the reserved starting time of j, and $(1+\delta)t'$ is the reserved completion time of j). This interval is too long for the job, and we place the job in the middle. Specifically, we assign j to start at time $(1+\delta)\cdot t + \frac{\delta \cdot p_j}{2s_\ell}$, where this time will be called the basic starting time of j. Scheduling the job in the middle of the interval allows us to keep the time where exactly half of the job is completed unchanged (but stretched together with the schedule). Obviously, no job will start running before its release date, and the schedule remains timely. The basic completion time of j will be $(1+\delta)\cdot t + \frac{\delta \cdot p_j}{2s_\ell} + \frac{p_j}{s_\ell}$,

that is,

$$(1+\delta)\cdot t + \frac{p_j}{s_\ell}(1+\delta) - \frac{\delta p_j}{2s_\ell} = (1+\delta)\cdot t + (t'-t)(1+\delta) - \frac{\delta p_j}{2s_\ell} = (1+\delta)\cdot t' - \frac{\delta p_j}{2s_\ell} \ .$$

If j originally completed during $\mathcal{J}_{i,\ell}$ in $\hat{S}(\bar{A}')$, then both its reserved and basic completion times will be no later than in the interval $\mathcal{J}_{i+1,\ell}$, so the cost increases by at most a multiplicative factor of $1+\delta$.

Next, for any interval $\mathcal{J}_{i,\ell}$, such that there is no job whose reserved time interval contains $\mathcal{J}_{i,\ell}$, shift all jobs that start and end during this time interval such that they run continuously as early as possible, that is, either starting at the beginning of $\mathcal{J}_{i,\ell}$, or just at the basic completion time of a job that starts in an earlier time interval and completes in $\mathcal{J}_{i,\ell}$ (jobs are reassigned to run without idle time, ignoring the time that was reserved before jobs are started or after they are completed). Moreover, if there is a job whose reserved starting time is during $\mathcal{J}_{i,\ell}$, its reserved completion time is in another time interval, but it is *small* for $\mathcal{J}_{i,\ell}$, where we define small for $\mathcal{J}_{i,\ell}$ such that its size is smaller than δ^{10} times the length of the interval, it is also shifted to start as early as possible. The set of these jobs, that we call the *jobs of* $\mathcal{J}_{i,\ell}$, can be processed in any order, as long as the length of the interval is sufficient to accommodate all of them such that their completion times are within this time interval. The new starting times and completion times will be called (with a slight abuse of notation) actual starting times and actual completion times, respectively. For jobs whose time slots are not modified, the actual starting times and actual completion times are equal to the basic starting times and basic completion times, respectively.

Analysis. Let U denote the total size of the jobs whose reserved starting times and reserved completion times are in $\mathcal{J}_{i,\ell}$. If there is a job whose reserved starting time is in this time interval but its reserved completion time is not, consider this job and denote it by x. The basic starting time of x is in $\mathcal{J}_{i,\ell}$ or in a later interval (this may happen because the basic starting time is larger than the reserved starting time), and the basic completion time is larger than its basic starting time. Let U_2 denote the total size of x that can run during $\mathcal{J}_{i,\ell}$ between its basic starting time and the minimum between its basic completion time and the end of the interval (this value can be zero even if x exists). Let U_2' be the size that can be run between the reserved starting time of x and the minimum between the basic starting time of x and the end of the interval (letting $U_2 = U_2' = 0$ if no such job exists). Similarly, if there is a job with a reserved starting time in an earlier time interval on the same machine and a reserved completion time in $\mathcal{J}_{i,\ell}$, let U_1 denote the total size of this job that can run during $\mathcal{J}_{i,\ell}$ until its basic completion time, and let U_1' the size that can run in $\mathcal{J}_{i,\ell}$ after the basic completion time and until the reserved completion time (where $U_1 = U_1' = 0$ if no such job exists).

For a fixed interval $\mathcal{J}_{i,\ell}$ let $Z = s_\ell \cdot \delta \cdot (1+\delta)^i$ denote its length.

▶ **Lemma 10.** *Assume that there is no job with a reserved time interval that contains $\mathcal{J}_{i,\ell}$. Then, the total size that is available for jobs of $\mathcal{J}_{i,\ell}$ (in the time stretched schedule) is at least $U + \delta^2 \cdot Z$, and the current total size of the jobs of $\mathcal{J}_{i,\ell}$ (after the modifications concerning shifting jobs, as described above) is at most $U + \delta^{10} \cdot Z$.*

Consider a (timely) schedule $\hat{S}$ for $\bar{A}'$ and the solution $\bar{S}$ obtained from $\hat{S}$ by time stretching by a factor of $1+\delta$. Let $\mathcal{J}_{i,\ell}$ be an interval that is not contained in a reserved time interval of any job. By the last claim, this interval contains an idle time of length at least δ^3 times the length of the interval, and such idle time in an interval $\mathcal{J}_{i,\ell}$ is called *a gap* in $\mathcal{J}_{i,\ell}$. The lower bound on the length is found by $(U+\delta^2\cdot Z) - (U+\delta^{10}\cdot Z) = \delta^2(1-\delta^8)\cdot Z > \delta^3 \cdot Z$ for $\delta \leq \frac{1}{36}$.

Observe that there might be other time intervals containing idle time which are not gaps. This happens in the case that the time interval is contained in a reserved period of some job and at least one of the basic starting time and the basic completion time is an inner point of that interval.

▶ **Corollary 11.** *Given a timely schedule $\hat{S}$, the schedule $\tilde{S}$ obtained from $\hat{S}$ by time stretching by a factor of $1+\delta$ can be constructed in polynomial time and the cost (that is, pseudo-cost) of this solution is at most $1+\delta$ times the cost (pseudo-cost) of $\hat{S}$. Consider $\tilde{S}$ and an interval $\mathcal{J}_{i,\ell}$ for which one of the following holds for $\tilde{S}$: the entire time interval is idle, or there is a job with a reserved starting time in this time interval, or there is a job with a reserved completion time in this interval. For this interval, $\tilde{S}$ satisfies that the interval contains idle time of length at least $\delta^3 \cdot Z$ where $Z = s_\ell \cdot \delta \cdot (1+\delta)^i$ is the length of $\mathcal{J}_{i,\ell}$. Moreover, for any job that is small for the interval that contains its basic starting time, its basic completion time is in the same interval.*

Consider the solution $\bar{S}$ obtained from a timely schedule $\hat{S}$ by time stretching by a factor of $1+\delta$, and let j be a job whose starting time in $\hat{S}$ is in the interval $\mathcal{J}_{i,\ell}$. The reserved starting and completion time of j in $\bar{S}$ are not larger than $1+\delta$ multiplied by the starting time and completion time of j in $\hat{S}$. The processing time of j on machine ℓ is at most $\frac{(1+\delta)^i}{\delta}$ since $\hat{S}$ is timely, and since the starting time of j in $\hat{S}$ is $\mathcal{J}_{i,\ell}$. Thus, the completion time of j in $\hat{S}$ is at most $(1+\delta)^i + \frac{(1+\delta)^i}{\delta} = \frac{(1+\delta)^{i+1}}{\delta}$. In $\bar{S}$, the reserved completion time of j is at most $\frac{(1+\delta)^{i+2}}{\delta}$, and the end of the time interval containing the reserved completion time of this job is at most $\frac{(1+\delta)^{i+3}}{\delta}$. Therefore, using $\frac{(1+\delta)^3}{\delta} \leq \frac{1}{\delta^2}$ we conclude the following.

▶ **Lemma 12.** *If $\bar{S}$ is obtained from a timely schedule $\hat{S}$ by time stretching by a factor of $1+\delta$, and $\mathcal{J}_{i,\ell}$ has a gap in $\bar{S}$, then $\bar{S}$ has another gap on machine ℓ during the time interval $[(1+\delta)^{i+1}, (1+\delta)^i \cdot \frac{1}{\delta^2})$.*

In what follows, given a time-stretched schedule, we refer to basic starting times and basic completion times simply as starting times and completion times, respectively.

A job shifting procedure. We now return to focusing on the design of our EPTAS, and we consider the rounded input A' resulting from the standard rounding steps. Recall that the density of job j in A' is the ratio $\frac{w_j}{p_j}$.

Our next goal is to create a new input from A' by increasing the release date of some of the jobs while keeping all other parts of the input unchanged, such that the following holds. There is a positive constant z, such that for every integer value of t, the set of jobs released at time $(1+\delta)^t$ of a common density can be scheduled on the m machines to complete no later than time $z(1+\delta)^t$. This goal was achieved for identical machines by Afrati et al. [1] using a fairly straightforward exchange argument. The case of uniformly related machines is significantly more difficult because in the exchange argument for uniformly related machines (as opposed to the situation for identical machines), a small job that is delayed to a later time interval need not stay small since it may be assigned to a different (slower) machine. This difficulty was not observed by Chekuri and Khanna [8], and thus we cannot use their procedure. Therefore, in order to overcome this difficulty, we provide a new procedure that handles each density separately, and uses a fine partition of the jobs of each density for which we are able to show such a bound.

First, we define *divisions* of job sizes as follows. Let $\Delta = \lceil \log_{1+\delta} 2 \rceil$. We have $\Delta \leq \frac{1}{\delta}$ as $(1+\delta)^{\frac{1}{\delta}} > 2$ (which can be verified since $\frac{1}{\delta}$ is an integer). For a job size $(1+\delta)^i$, let k_i be an integer such that $2^{k_i} \cdot (1+\delta)^i \in (1,2]$. Let $k'_i = \lceil \log_{1+\delta} 2^{k_i} \rceil + i$, where $k'_i \leq \Delta$ as $2^{k_i} \cdot (1+\delta)^i \leq 2$, and therefore $\lceil \log_{1+\delta}(2^{k_i} \cdot (1+\delta)^i) \rceil \leq \lceil \log_{1+\delta} 2 \rceil$. Additionally, $k'_i \geq 1$, as $2^{k_i} \cdot (1+\delta)^i > 1$.

The division of a job of size $(1+\delta)^i$ is defined to be k'_i, and its subdivision is k_i. The pseudo-size of such a job is defined to be $\pi_i = \frac{(1+\delta)^{k'_i}}{2^{k_i}}$. Since $\log_{1+\delta} 2^{k_i} \leq k'_i - i < \log_{1+\delta} 2^{k_i} + 1$, we have $(1+\delta)^{k'_i - i} \geq 2^{k_i}$ and $(1+\delta)^{k'_i - i - 1} < 2^{k_i}$. Thus, we have $(1+\delta)^i \leq \pi_i < (1+\delta)^{i+1}$ (so the pseudo-size is not smaller than the size, but it is not much larger). Since every job has a value k'_i, and $1 \leq k'_i \leq \Delta$, the divisions $1, 2, \ldots, \Delta$ form a partition of $[1, 2)$.

More intuitively, the division of a size of a job (a power of $1+\delta$) is the part of $[1, 2)$ that it belongs to when it is multiplied by an appropriate integer power of 2. The subdivision is this last power of 2. For example, a job of size $(1+\delta)^{-1}$ has the subdivision 1, as $1 < \frac{2}{1+\delta} \leq 2$. Its division is $\lceil \log_{1+\delta} 2 \rceil - 1 = \Delta - 1$, which means that once the size is multiplied by the appropriate power of 2 (which is 2^1 for this example), it becomes relatively close to 2 (and indeed $\frac{1}{1+\delta}$ is not much smaller than 1). Two other examples are jobs of sizes 1 and $1+\delta$. The subdivisions of these jobs are 1 and 0, respectively, their divisions are Δ and 1 (and indeed the second size is not much larger than 1).

Let A'' be A' such that the size of a job of size $(1+\delta)^i$ is replaced with the pseudo-size π_i. The values π_i of one division form a divisible sequence, and more specifically, given two such distinct values of one division, the larger one divided by the smaller one is a positive integral power of 2. In what follows we will schedule the jobs of A'' in some cases. Let $\tilde{\mathcal{O}}$ be an optimal timely solution for A'' and recall that $\mathcal{O}''$ is an optimal timely schedule for A'.

▶ **Corollary 13.** *We have* $\mathcal{O}''(A') \leq \tilde{\mathcal{O}}(A'') \leq (1+\delta)\mathcal{O}''(A')$.

Recall that any job of size below $\delta^{10} \cdot s_\ell \cdot \delta(1+\delta)^i = \delta^{11} \cdot s_\ell \cdot (1+\delta)^i$ is defined to be *small* for interval $\mathcal{J}_{i,\ell}$. We also say that any job of size in $[\delta^{11} \cdot s_\ell \cdot (1+\delta)^i, s_\ell \cdot \delta \cdot (1+\delta)^i)$ is called *medium* for this time interval, any job of size in $[s_\ell \cdot \delta \cdot (1+\delta)^i, \frac{s_\ell}{\delta} \cdot (1+\delta)^{i+1})$ is called *large* for this time interval, and larger jobs (of size at least $\frac{s_\ell}{\delta} \cdot (1+\delta)^{i+1}$) are called *huge* for the interval. Recall that a job that is huge for a given time interval cannot start during this time interval in a timely schedule, and a large job cannot be assigned to run only in the interval (it is not possible that both its starting time and completion time are in the interval due to the length of the interval). We say that a job is *big for an interval* if it is either medium or large for it. The following definition uses the pseudo-sizes of jobs that are their sizes in A''.

▶ **Definition 14.** *A schedule for A'' is called organized if it is timely, and it has the following properties. 1. Let j be a job whose starting time is in $\mathcal{J}_{i,\ell}$. If j is small for this interval, then its completion time is in the same interval.*

2. Consider two jobs, j_1, j_2, of a given division k and the same density, such that the starting time of j_1 is in interval $\mathcal{J}_{i_1,\ell_1}$, and the starting time of j_2 is in interval $\mathcal{J}_{i_2,\ell_2}$, where $i_2 > i_1$ or both $i_2 = i_1$ and $\ell_2 > \ell_1$. Moreover, assume that $r'_{j_2} \leq (1+\delta)^{i_1}$.

a. If $\pi_{j_1} < \pi_{j_2}$, then j_2 is not small for $\mathcal{J}_{i_1,\ell_1}$. b. If $\pi_{j_1} = \pi_{j_2}$, then $r'_{j_2} \geq r'_{j_1}$. c. If both $\pi_{j_1} = \pi_{j_2}$ and $r'_{j_2} = r'_{j_1}$ hold, then the index of the second job is smaller, i.e., $j_2 < j_1$.

Since the considered schedule A'' is timely, jobs j_1, j_2 are not huge for $\mathcal{J}_{i_1,\ell_1}$ and $\mathcal{J}_{i_2,\ell_2}$, respectively. This means that for j_ψ ($\psi = 1, 2$), if it is not small for $\mathcal{J}_{i_\psi,\ell_\psi}$, it has to be big for this interval. Note that in the case $\pi_{j_1} < \pi_{j_2}$, if $i_1 = i_2 = i$, the case where j_1 is big for $\mathcal{J}_{i,\ell_1}$ while j_2 is small for $\mathcal{J}_{i,\ell_2}$, cannot occur since speeds are sorted (so machine ℓ_2 is not faster than ℓ_1).

This definition specifies fixed priorities on jobs. For jobs of equal sizes and densities, a job of a smaller release date has higher priority, and out of two such jobs with the same release date, the one of the higher index has a higher priority. For a fixed time interval, there are also priorities between jobs of one division that are small for it in the sense that the interval will contain jobs that are prioritized while jobs with a lower priority are assigned to start

during a later interval or to a parallel interval (during the exact same time) on a machine of a higher index. The priority between two jobs of one size remains the same, and additionally, a larger job has a higher priority than a smaller job (this is defined per time interval, and only for jobs of the instance that are small for it). Obviously, given a time interval $\mathcal{J}_{i,\ell}$, all jobs that are released at time $(1+\delta)^{i+1}$ or later are irrelevant for it and have no priority. Let $\bar{\mathcal{O}}$ denote an optimal organized schedule for A'' (in particular, $\bar{\mathcal{O}}$ must be timely).

▶ **Lemma 15.** $\bar{\mathcal{O}}(A'') \leq (1+2\delta) \cdot \tilde{\mathcal{O}}(A'')$.

We apply an additional transformation on release dates to obtain the instance $\tilde{A}$ from A' where the release date of j is denoted by r_j (and satisfies $r_j \geq r'_j$). Other than modified release dates for some of the jobs, $\tilde{A}$ and A' have the same machines and the same jobs. The instance $\tilde{A}$ may contain release dates that do not exist in A', but all release dates are integer powers of $1+\delta$ and the smallest release date will remain $(1+\delta)^{\theta}$. Thus, any (timely) solution for $\tilde{A}$ is a (timely) solution for A' as well. In order to use a solution of A' for $\tilde{A}$, one has to show that each job j that has a starting time in $\mathcal{J}_{i,\ell}$ has $r_i \leq (1+\delta)^i$ (while other aspects of feasibility follow from the feasibility for A'). In particular, we can use an organized schedule for A'' as a schedule for A' and thus for $\tilde{A}$. In $\tilde{A}$, for every type of jobs, if too many (in terms of the number or the total size) pending jobs exist at time $(1+\delta)^i$, the release date of some of them is increased. This can be done when it is impossible to schedule all these jobs due to the lengths of intervals. We need to take into account jobs that have starting times in an interval $\mathcal{J}_{i,\ell}$ even if their completion times are in later time intervals. Jobs with this property are called special. The total size of jobs that are not special will be at most the total lengths of time intervals. The process is applied separately and independently on every possible density.

We will define $\tilde{A}$ using a process that acts on increasing values of i and at each step applies a modification on release dates of jobs whose current release date is $(1+\delta)^i$. For a given density d, initialize $r_j = r'_j$ for every job j of density d. For each possible size, create a list of preferences. In this list, jobs are ordered by non-decreasing release dates in A', and within every release date, by decreasing indices. In what follows, for any job j we will refer to r'_j as the initial release date of j, and to the values r_j as modified release dates. As these values will be changed during a modification process that we define (they can possibly be changed a number of times for each job), when we discuss such a value, we will always refer to the current value even if it will be changed later. We apply the process for $i = \theta, \theta+1, \ldots$. The stopping condition will be the situation where for some i there are no jobs whose modified release dates are at least $(1+\delta)^i$. If during the process for some i there are no jobs with modified release date $(1+\delta)^i$, but there exist jobs with larger release dates (in this case these are both initial and modified release dates of those jobs), we skip this value of i and move to the next value.

Recall that our goal is to increase some (modified) release dates of jobs where these jobs will not be started before the next possible release date, and to break ties consistently, we do this for jobs that would not be started before the next release date in an organized schedule. Since we are interested in organized schedules that are, in particular, timely, jobs that are huge for a given interval should not be scheduled a starting time in this interval.

For a given value of i, and a fixed density d, we consider jobs whose modified release date is $(1+\delta)^i$. Out of these jobs we will select a subset, and for every such j in the subset, the current value of r_j will remain $(1+\delta)^i$ and will not change later. For every job j' that is not selected (after applying the process to all machines), we modify its modified release date into $r_{j'} = (1+\delta)^{i+1}$. Initially, no job is selected. For every time interval $\mathcal{J}_{i,\ell}$ (for some $1 \leq \ell \leq m$), for every size that is big (large or medium) for this interval and every density,

select the unselected highest priority job j such that $r_j = (1+\delta)^i$. For sizes that no such job exists obviously no job is selected. So far we selected at most one job for every size and density among sizes that are not huge for the interval (for which we do not choose jobs) and sizes that are not small for the interval. We proceed to choosing additional medium jobs and to choosing small jobs.

For every size that is medium for this interval and every density, select an additional unselected $\frac{1}{\delta^{10}}$ highest priority jobs of this size (selecting all such jobs if less than $\frac{1}{\delta^{10}}$ such jobs exist). For every division k and every density, consider the job sizes of this division that are small for this time interval in non-increasing order. For each size (that is small for the interval), unselected jobs of one size are selected according to priorities. Keep selecting unselected jobs according to priorities, moving to the next size if all jobs of one size have been selected, until the total size of selected jobs of division k (and density d, that are small for $\mathcal{J}_{i,\ell}$) is at least $s_\ell\delta(1+\delta)^i$ (and thus their total size is at most $s_\ell\delta(1+\delta)^i(1+\delta^{10})$, since any such job is small for $\mathcal{J}_{i,\ell}$). If the total size of all these jobs is smaller than $s_\ell \cdot \delta(1+\delta)^i$, all of them are selected. For a given triple d, i, ℓ, letting $Z = s_\ell \cdot \delta(1+\delta)^i$, the total size of selected jobs is as follows. There are at most $\log_{1+\delta}\frac{1+\delta}{\delta^2}+1 \leq \frac{1}{\delta^3}+2$ sizes of large jobs, each having a size of at most $\frac{1+\delta}{\delta^2}\cdot Z$. There are at most $\log_{1+\delta}\frac{1}{\delta^{10}}+1 \leq \frac{1}{\delta^{11}}+1$ sizes of medium jobs, each having a size of at most Z. To find an upper bound on the total size of small jobs that are selected, recall that there are at most $\frac{1}{\delta}$ divisions. The total size of selected large jobs is at most $Z(\frac{1}{\delta^3}+2)(\frac{1+\delta}{\delta^2}) < \frac{Z}{\delta^6}$. The total size of selected medium jobs is at most $(1+\frac{1}{\delta^{10}})(\frac{1}{\delta^{11}}+1)Z < \frac{Z}{\delta^{22}}$. The total size of small jobs is at most $Z\cdot\frac{1}{\delta}(1+\delta^{10})$. The total size is therefore at most $\frac{Z}{\delta^{23}}$.

▶ **Lemma 16.** *In an organized schedule for A'', every job j is assigned a starting time in a time interval that is no earlier than r_j. Thus, any organized solution for A'' is a valid timely schedule for $\tilde{A}$ with the same cost.*

▶ **Lemma 17.** *If the input belongs to at most $\hat{y}$ densities (where $\hat{y}$ is a function of δ), then the jobs of modified release date r can be all scheduled during $[t,t')$, where $t \geq r$, and $t' \leq t + \frac{r\hat{y}}{\delta^{22}}$.*

Given a set of $\hat{y}$ densities, the set of jobs with modified release date at most r and these densities, can be all scheduled during a time interval $[t,t')$ where $t \geq r$ and $t' \leq t + \frac{r\hat{y}(1+\delta)}{\delta^{23}}$.

Proof. Let $t \geq r$, and schedule the jobs whose modified release date is r starting at time t, such that the jobs scheduled on machine ℓ are those that were selected for $\mathcal{J}_{i,\ell}$ (where $r = (1+\delta)^i$). The total size of the jobs of one density is at most $\frac{Z}{\delta^{23}} = \frac{s_\ell(1+\delta)^i}{\delta^{22}} = \frac{s_\ell\cdot r}{\delta^{22}}$ so the jobs of $\hat{y}$ densities will be completed (on a machine of speed s_ℓ) within a time interval not longer than $\frac{r\cdot\hat{y}}{\delta^{22}}$. To prove the second claim, note that the total size of jobs of one density and modified release date at most r that will run on machine ℓ is at most $\frac{s_\ell\cdot r}{\delta^{22}}\cdot\sum_{i=0}^{\log_{1+\delta} r-\theta}\frac{1}{(1+\delta)^i} \leq \frac{s_\ell r}{\delta^{22}}\cdot\frac{1+\delta}{\delta}$, since previous interval lengths are smaller by a multiplicative factor which is a positive integral power of $\frac{1}{1+\delta}$ dependent on the distance to r. ◀

▶ **Corollary 18.** *Consider an instance $\tilde{A}_s$ obtained from $\tilde{A}$ by keeping jobs belonging to only $\hat{y}$ given densities, and that are released by time R (including R, for a fixed $R > 0$). There exists an optimal timely schedule $\mathcal{O}_s$ for $\tilde{A}_s$ where no job is completed after time $\frac{\hat{y}R}{\delta^{25}}$.*

Proof. Consider an optimal schedule $\mathcal{O}'_s$, and let $R' = \frac{\hat{y}R}{\delta^{24}}$. We define T as follows. Let $R' < T \leq (1+\delta)R'$ be such that $T = (1+\delta)^\iota$ for an integer ι. Consider the jobs released by time R whose completion times are above T (and thus in their pseudo-costs, their weights are multiplied by at least $T(1+\delta)$). By Lemma 17, it is possible to schedule these jobs during $[T, T+\frac{R\hat{y}}{\delta^{23}})$. We have $\frac{R\hat{y}}{\delta^{23}} = \delta R' < \delta T$, and therefore it is possible to remove

all these jobs from $\mathcal{O}'_s$, and schedule them within the time interval $[T, (1+\delta)T)$. As all jobs that are still running at time T (jobs whose starting times are smaller than T but their completion times are larger than T) are also removed, before the jobs are assigned again, the time interval $[T, (1+\delta)T)$ is idle on all machines. No reassigned job has an increased cost, and therefore the resulting schedule, $\mathcal{O}_s$ is optimal as well. We are done since $(1+\delta)T \leq (1+\delta)^2 R' = (1+\delta)^2 \frac{\hat{y}R}{\delta^{24}} < \frac{\hat{y}\cdot R}{\delta^{25}}$. ◀

This concludes the initial steps of our scheme. Together with later steps (see the full version of this work) we derive an EPTAS for our problem.

References

1 Foto N. Afrati, Evripidis Bampis, Chandra Chekuri, David R. Karger, Claire Kenyon, Sanjeev Khanna, Ioannis Milis, Maurice Queyranne, Martin Skutella, Clifford Stein, and Maxim Sviridenko. Approximation schemes for minimizing average weighted completion time with release dates. In *Proc. 40th Symp. Foundations of Computer Science (FOCS)*, pages 32–44, 1999. doi:10.1109/SFFCS.1999.814574.

2 Foto N. Afrati and Ioannis Milis. Designing PTASs for MIN-SUM scheduling problems. *Discrete Applied Mathematics*, 154(4):622–639, 2006. doi:10.1016/J.DAM.2005.05.014.

3 N. Alon, Y. Azar, G. J. Woeginger, and T. Yadid. Approximation schemes for scheduling. In *Proc. 8th Symp. on Discrete Algorithms (SODA)*, pages 493–500, 1997.

4 N. Alon, Y. Azar, G. J. Woeginger, and T. Yadid. Approximation schemes for scheduling on parallel machines. *Journal of Scheduling*, 1(1):55–66, 1998.

5 Hocine Belouadah, Marc E. Posner, and Chris N. Potts. Scheduling with release dates on a single machine to minimize total weighted completion time. *Discrete applied mathematics*, 36(3):213–231, 1992. doi:10.1016/0166-218X(92)90255-9.

6 Sebastian Berndt, Hauke Brinkop, Klaus Jansen, Matthias Mnich, and Tobias Stamm. New support size bounds for integer programming, applied to makespan minimization on uniformly related machines. In *Proc. 34th International Symp. on Algorithms and Computation (ISAAC)*, 2023.

7 M. Cesati and L. Trevisan. On the efficiency of polynomial time approximation schemes. *Information Processing Letters*, 64(4):165–171, 1997. doi:10.1016/S0020-0190(97)00164-6.

8 Chandra Chekuri and Sanjeev Khanna. A PTAS for minimizing weighted completion time on uniformly related machines. In *Proc. 28th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 848–861, 2001. doi:10.1007/3-540-48224-5_69.

9 L. Epstein and J. Sgall. Approximation schemes for scheduling on uniformly related and identical parallel machines. *Algorithmica*, 39(1):43–57, 2004. doi:10.1007/S00453-003-1077-7.

10 Leah Epstein and Asaf Levin. An efficient polynomial time approximation scheme for load balancing on uniformly related machines. *Mathematical Programming*, 147(1-2):1–23, 2014. doi:10.1007/S10107-013-0706-4.

11 Leah Epstein and Asaf Levin. Minimum total weighted completion time: Faster approximation schemes. *CoRR*, abs/1404.1059, 2014. arXiv:1404.1059.

12 Leah Epstein and Asaf Levin. The benefit of preemption for single machine scheduling so as to minimize total weighted completion time. *Operations Research Letters*, 44(6):772–774, 2016. doi:10.1016/J.ORL.2016.09.013.

13 Leah Epstein and Asaf Levin. An efficient polynomial time approximation scheme for minimizing the total weighted completion time on uniformly related machines. In *Proc. 19th International Symposium on Algorithms and Data Structures (WADS)*, 2025. To appear.

14 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

15 D. S. Hochbaum. Various notions of approximations: Good, better, best, and more. In D. S. Hochbaum, editor, *Approximation algorithms*. PWS Publishing Company, 1997.

16 D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: theoretical and practical results. *Journal of the ACM*, 34(1):144–162, 1987. doi:10.1145/7531.7535.

17 D. S. Hochbaum and D. B. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM Journal on Computing*, 17(3):539–551, 1988. doi:10.1137/0217033.

18 W. A. Horn. Minimizing average flow time with parallel machines. *Operations Research*, 21(3):846–847, 1973.

19 K. Jansen. An EPTAS for scheduling jobs on uniform processors: Using an MILP relaxation with a constant number of integral variables. *SIAM Journal on Discrete Mathematics*, 24(2):457–485, 2010. doi:10.1137/090749451.

20 Dušan Knop and Martin Koutecký. Scheduling meets n-fold integer programming. *Journal of Scheduling*, 21:493–503, 2018. doi:10.1007/S10951-017-0550-0.

21 Dusan Knop and Martin Koutecký. Scheduling kernels via configuration LP. In *Proc. 30th Annual European Symposium on Algorithms (ESA)*, pages 73:1–73:15, 2022. doi:10.4230/LIPICS.ESA.2022.73.

22 Dusan Knop, Martin Koutecký, Asaf Levin, Matthias Mnich, and Shmuel Onn. Parameterized complexity of configuration integer programs. *Operations Research Letters*, 49(6):908–913, 2021. doi:10.1016/J.ORL.2021.11.005.

23 Dusan Knop, Martin Koutecký, Asaf Levin, Matthias Mnich, and Shmuel Onn. High-multiplicity N-fold IP via configuration LP. *Mathematical Programming*, 200(1):199–227, 2023. doi:10.1007/S10107-022-01882-9.

24 Ishai Kones and Asaf Levin. A unified framework for designing eptas for load balancing on parallel machines. *Algorithmica*, 81(7):3025–3046, 2019. doi:10.1007/S00453-019-00566-9.

25 Martin Koutecký and Johannes Zink. Complexity of scheduling few types of jobs on related and unrelated machines. *Journal of Scheduling*, 28(1):139–156, 2025. doi:10.1007/S10951-024-00827-8.

26 Eugene L. Lawler. Sequencing jobs to minimize total weighted completion time subject to precedence constraints. In *Algorithmic Aspects of Combinatorics*, volume 2, pages 75–90. Elsevier, 1978.

27 J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.

28 Andreas S. Schulz. Scheduling to minimize total weighted completion time: Performance guarantees of LP-based heuristics and lower bounds. In *Proc. 5th International conference on integer programming and combinatorial optimization (IPCO)*, pages 301–315, 1996. doi:10.1007/3-540-61310-2_23.

29 Martin Skutella. A 2.542-approximation for precedence constrained single machine scheduling with release dates and total weighted completion time objective. *Operations Research Letters*, 44(5):676–679, 2016. doi:10.1016/J.ORL.2016.07.016.

30 Martin Skutella and Gerhard J. Woeginger. A PTAS for minimizing the total weighted completion time on identical parallel machines. *Mathematics of Operations Research*, 25(1):63–75, 2000. doi:10.1287/MOOR.25.1.63.15212.

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