A Unified Framework for Designing EPTAS for Load Balancing on Parallel Machines

Ishai Kones1 · Asaf Levin1

Received: 25 April 2018 / Accepted: 15 March 2019 / Published online: 27 March 2019
© Springer Science+Business Media, LLC, part of Springer Nature 2019

Abstract
We consider a general load balancing problem on parallel machines. Our machine environment in particular generalizes the standard models of identical machines, and the model of uniformly related machines, as well as machines with a constant number of types, and machines with activation costs. The objective functions that we consider contain in particular the makespan objective and the minimization of the $\ell_p$-norm of the vector of loads of the machines, and each case allows the possibility of job rejection. We consider this general model and design an efficient polynomial time approximation scheme (EPTAS) that applies for all its previously-studied special cases. This EPTAS improves the current best approximation scheme for some of these cases where only a polynomial time approximation scheme was known into an EPTAS.

Keywords Approximation schemes · Scheduling · Load balancing

1 Introduction

We consider a model that generalizes many previously-studied optimization problems in the framework of scheduling and (minimization) load balancing problems on parallel machines. We use this generalization in order to exhibit that there is a standard way to design efficient polynomial time approximation schemes for all these special cases and for new special cases as well. In the earlier works, approximation schemes for

This research was supported by a grant from the GIF, the German-Israeli Foundation for Scientific Research and Development (Grant Number I-1366-407.6/2016). An earlier extended abstract version of this work appears in Proc. CiE 2018.

Asaf Levin
levinas@ie.technion.ac.il
Ishai Kones
ishai.kones@gmail.com

1 Faculty of Industrial Engineering and Management, The Technion, 32000 Haifa, Israel
many special cases of our model were developed using ad-hoc tricks, we show that such ad-hoc methods are not necessary.

Before going into the details of the definition of our model, we define the types of approximation schemes. A $\rho$-approximation algorithm for a minimization problem is a polynomial time algorithm that always finds a feasible solution of cost at most $\rho$ times the cost of an optimal solution. The infimum value of $\rho$ for which an algorithm is a $\rho$-approximation is called the approximation ratio or the performance guarantee of the algorithm. A polynomial time approximation scheme (PTAS) for a given problem is a family of approximation algorithms such that the family has a $(1 + \varepsilon)$-approximation algorithm for any $\varepsilon > 0$. An efficient polynomial time approximation scheme (EPTAS) is a PTAS whose time complexity is upper bounded by the form $f(\frac{1}{\varepsilon}) \cdot \text{poly}(n)$ where $f$ is some computable (not necessarily polynomial) function and $\text{poly}(n)$ is a polynomial of the length of the (binary) encoding of the input. A fully polynomial time approximation scheme (FPTAS) is a stricter concept. It is defined like an EPTAS, with the added restriction that $f$ must be a polynomial in $\frac{1}{\varepsilon}$. Note that whereas a PTAS may have time complexity of the form $n^{g(\frac{1}{\varepsilon})}$, where $g$ is for example linear or even exponential, this cannot be the case for an EPTAS. The notion of an EPTAS is modern and finds its roots in the FPT (fixed parameter tractable) literature (see e.g. [5, 7, 11, 25]).

Since these problems are proven to be strongly NP-Hard [12] (as for example our model is an extension of the minimum makespan problem on identical machines), it is unlikely (impossible assuming $P \neq NP$) that an optimal polynomial time algorithm or an FPTAS will be found for them. In our research, we will focus on finding an EPTAS for this general model and as a bi-product, obtain improved results to many of its special cases. As usual, in order to present an EPTAS we can show that for a sufficiently small value of $\varepsilon$ there exists an algorithm of time complexity of the form $f(\frac{1}{\varepsilon}) \cdot \text{poly}(n)$ with an approximation ratio of $1 + \kappa \varepsilon$ for an arbitrary constant $\kappa$ (independent of $\varepsilon$).

**Our model** Being a scheduling problem, the definition of the problem can be partitioned into the characteristics of the machines, the properties of the jobs, and the objective function.

**Machine characteristics** We are given $m$ machines denoted as $1, 2, \ldots, m$ each of which can be activated to work in one of $\tau$ types denoted as $1, 2, \ldots, \tau$. The type of the machine will influence the processing time of a job assigned to that machine. The input defines for every machine $i$ a (positive rational) speed $s_i$ and an activation cost function $\alpha_i(t)$ that is a non-negative rational number denoting the cost of activating machine $i$ in type $t$. We are also given a budget $\hat{A}$ on the total activation cost of all machines. The meaning of this budget is that a feasible solution needs to specify for every machine $i$ its type $t_i$ such that the total activation cost is at most the budget, that is, the following constraint holds

\[\sum_{i=1}^{m} \alpha_i(t_i) \leq \hat{A}\]
\[
\sum_{i=1}^{m} \alpha_i(t_i) \leq \hat{A}.
\] (1)

In our work we assume that \(\tau\) is a constant while \(m\) is a part of the input. Furthermore, without loss of generality we assume that \(1 = s_1 \geq s_2 \geq \cdots \geq s_m > 0\).

**Job characteristics** There are \(n\) jobs denoted as \(J = \{1, 2, \ldots, n\}\). Job \(j\) is associated with a \((\tau\text{-dimensional})\) vector \(p_j\) that specifies the size \(p_j(t)\) of job \(j\) if it is assigned to a machine of type \(t\). That is, if job \(j\) is assigned to machine \(i\), and we activate machine \(i\) in type \(t\), then the processing time of job \(j\) (on this machine) is \(\frac{p_j(t)}{s_i}\). Furthermore, for every job \(j\) we are given a rejection penalty \(\pi_j\) that is a positive rational number denoting the cost of not assigning job \(j\) to any machine. A definition of a feasible solution specifies for every job \(j\) if \(j\) is rejected (and thus incurs a rejection penalty of \(\pi_j\)) or not and if it is not rejected (i.e., \(j\) is accepted), then the machine \(i\) that \(j\) is assigned to. Formally, we need to specify a job assignment function \(\sigma: J \rightarrow \{0, 1, 2, \ldots, m\}\), where \(\sigma(j) = 0\) means that \(j\) is rejected, and \(\sigma(j) = i\) for \(i \geq 1\) means that \(j\) is assigned to machine \(i\).

**Definition of the objective function** As stated above a feasible solution defines a type \(t_i\) for every machine \(i\), and a job assignment function \(\sigma\). The load of machine \(i\) in this solution is

\[
\Lambda_i = \frac{\sum_{j \in J: \sigma(j) = i} p_j(t_i)}{s_i}.
\]

Our objective function is specified using a function \(F\) defined over the vector of the loads of the machines \(F(\Lambda_1, \Lambda_2, \ldots, \Lambda_m)\) that is the assignment cost of the jobs to the machines. \(F\) is defined by two scalar parameters \(\phi > 1\) and \(1 \geq \psi \geq 0\) as follows:

\[
F(\Lambda_1, \Lambda_2, \ldots, \Lambda_m) = \psi \cdot \max_{i=1}^{m} \Lambda_i + (1 - \psi) \cdot \sum_{i=1}^{m} (\Lambda_i)^\phi.
\]

The value of \(\psi\) has the following meaning. For \(\psi = 1\), the value of \(F\) is the makespan of the schedule, i.e., the maximum load of any machine, while for \(\psi = 0\), the value of \(F\) is the sum of the \(\phi\) powers of the loads of the machines, an objective that is equivalent to the \(\ell_\phi\) norm of the vector of loads. For \(\psi\) that is strictly between 0 and 1, the value of \(F\) is a convex combination of these classical objectives in the load balancing literature. The common values of \(\phi\) that were motivated by various applications that were considered in the literature are \(\phi = 2\) and \(\phi = 3\) (see e.g., [2]).

Our problem is to find a type \(t_i\) for every machine \(i\) such that \(\sum_{i=1}^{m} \alpha_i(t_i) \leq \hat{A}\), and a job assignment function \(\sigma\) so that the following objective function (denoted as \(\text{obj}\)) will be minimized:

\[
\text{obj} = F(\Lambda_1, \Lambda_2, \ldots, \Lambda_m) + \sum_{j \in J: \sigma(j) = 0} \pi_j.
\]
Our result is an EPTAS for this load balancing problem. For ease of notation we denote this problem by LBP and let $\varepsilon > 0$ be such that $1/\varepsilon \geq 100$ is an integer. We will use the fact that $\phi$ is a constant and the following simple property throughout the analysis.

**Lemma 1** Given a value of $\rho > 1$ and two vectors $(\Lambda_1, \ldots, \Lambda_m)$ and $(\Lambda_1', \ldots, \Lambda_m')$ such that for every $i$ we have $\Lambda_i \leq \Lambda_i' \leq (1 + \varepsilon)^\rho \Lambda_i$, then

$$F(\Lambda_1, \Lambda_2, \ldots, \Lambda_m) \leq F(\Lambda_1', \Lambda_2', \ldots, \Lambda_m') \leq (1 + \varepsilon)^\rho \cdot F(\Lambda_1, \Lambda_2, \ldots, \Lambda_m).$$

**Proof** Using the definition of $F$ we have

$$F(\Lambda_1, \Lambda_2, \ldots, \Lambda_m) = \psi \cdot \max_{i=1}^m \Lambda_i + (1 - \psi) \cdot \sum_{i=1}^m (\Lambda_i)^\phi,$$

$$\leq \psi \cdot \max_{i=1}^m \Lambda_i' + (1 - \psi) \cdot \sum_{i=1}^m (\Lambda_i')^\phi = F(\Lambda_1', \Lambda_2', \ldots, \Lambda_m'),$$

$$\leq \psi \cdot \max_{i=1}^m (1 + \varepsilon)^\rho \Lambda_i + (1 - \psi) \cdot \sum_{i=1}^m ((1 + \varepsilon)^\rho \Lambda_i)^\phi,$$

$$\leq (1 + \varepsilon)^\rho \cdot \left( \psi \cdot \max_{i=1}^m \Lambda_i + (1 - \psi) \cdot \sum_{i=1}^m (\Lambda_i)^\phi \right) = (1 + \varepsilon)^\rho \cdot F(\Lambda_1, \Lambda_2, \ldots, \Lambda_m). \quad \square$$

**Special cases of our model and related literature on these cases** The objective function we consider here generalizes the makespan minimization objective (the special case with all $\pi_j = \infty$ and $\psi = 1$), the sum of the $\phi$ powers of the machines loads (the special case with all $\pi_j = \infty$ and $\psi = 0$), as well as these two objectives with job rejections (i.e., finite $\pi_j$ for some $j \in J$).

As for the machine model that we consider, next we state some of the earlier studied special cases of this model. We say that machines have pre-specified types if $\hat{A} = 0$ and for every $i$ we have a value $t_i$ such that $\alpha_i(t_i) = 0$ and $\alpha_i(t) = 1$ if $t \neq t_i$. This special case of the machine environment is the case of unrelated machines with a constant number of types, whose special case where machines have a common speed was studied in [19] who presented an EPTAS for the makespan objective (the extension of this scheme to machines of different speeds was explored in [20]). This EPTAS of [19] improves earlier PTAS’s for that special case [4,13,27]. The $\ell_p$-norm minimization objective for the case where machines have pre-specified types and all speeds are 1 admits a PTAS [4].

The case where machines have pre-specified types generalizes its special case of uniformly related machines that is the case where $\tau = 1$. For this machine model, Jansen [17] presented an EPTAS for the makespan objective improving the earlier
Table 1  Summary of previous studies of special cases of problem LBP

| Definition of the special case using our notation | PTAS/EPTAS | References |
|-------------------------------------------------|------------|------------|
| \(\tau = 1, \psi = 1, \text{ and } \pi_j = \infty \forall j\) | EPTAS | [17] |
| \(\tau = 1, \psi = 0, \text{ and } \pi_j = \infty \forall j\) | EPTAS | [8] |
| \(\tau = 1, \psi = 1, \text{ and } s_i = 1 \forall i\) | PTAS | [3] |
| \(\tau = 1, \psi = 1 \text{ or } \psi = 0\) | PTAS | [10] |
| Machines with pre-specified types, \(\psi = 1 \text{ and } \pi_j = \infty \forall j\) | EPTAS | [20] |
| Machines with pre-specified types, \(\psi = 0, s_i = 1 \forall i, \text{ and } \pi_j = \infty \forall j\) | PTAS | [4] |
| \(\tau = 2, \alpha_i(1) = 0 \forall i, p_j(1) = \pi_j = \infty \forall j, \psi = 1, \text{ and } \pi_j = \infty \forall j\) | PTAS | [23] |

For every row for which the second column is a PTAS, our EPTAS is the first efficient polynomial time approximation scheme for this special case.

PTAS established in the seminal work of Hochbaum and Shmoys [16], while Epstein and Levin [8] presented an EPTAS for the minimization of the \(\ell_p\)-norm of the vector of machines loads improving an earlier PTAS by Epstein and Sgall [10]. Later on, Epstein and Levin [9] presented an EPTAS for another scheduling objective, namely total weighted completion times, and their scheme for the case where all jobs are released at time 0, implies a different EPTAS for the minimization of the sum of squares of the loads of the machines on uniformly related machines. As far as we know the two schemes of [8,9] are the only examples for EPTAS’s for load balancing objectives on uniformly related machines where one cannot use the dual approximation method of [15,16]. Our approach here is based on [9].

The case of identical machines is the special case of uniformly related machines where all speeds are equal. See [1,14,15] for earlier approximation schemes for this case. The time complexity of the schemes for makespan minimization on identical machines and on uniformly related machines, as well as the time complexity of the scheme for \(\ell_p\)-norm of the vector of machines loads on identical machines have been improved recently [18] almost matching an earlier lower bound on the time complexity of a scheme to the makespan minimization on identical machines [6].

The next special objective we consider here is scheduling with rejection. This is the special case of our objective function where \(\pi_j\) is finite (at least for some jobs). In [3,10] there is a PTAS for this variant (for \(\psi \in \{0,1\}\)) on identical machines and on uniformly related machines.

The last special case we consider here is the machines with activation costs model that was considered by [23]. They considered the special case of our model with makespan objective and \(\tau = 2\), with \(\alpha_i(1) = 0\) for all \(i\), and \(p_j(1) = \pi_j = \infty\) for all \(j \in J\). In this case activating a machine as type 1 means that the machine is not operating and cannot process any job. For this case [23] presents a PTAS.

We summarize the previously-studied special cases with reference to the previously known approximation scheme of the better complexity class (i.e., we cite the first EPTAS if there is one, and the first PTAS if an EPTAS was not known prior to our work) in Table 1.

We note that in our model the type of a machine is fixed for the entire planning horizon and it cannot be changed along time. This is different from models where
jobs need certain amount of resource and we are given a budget constraint on the total resource that can be used for processing jobs at any point in time (see e.g. [21]). The relation between these models and ours is that this resource allocation to a machine can be seen as assigning type to the machine, but in such cases the type of a machine may change along time.

Motivating application A possible application of our model may arise in the area of real-time scheduling as follows. A designer of a real-time system has as an input a set of jobs that need to be performed on the processors of the system within a given time unit. The jobs are control tasks of the system. At this design phase, the designer needs to choose the processors that he wishes to use in the system while satisfying the budget constraint on the total cost of the processors and not exceeding the number of slots for installing the processors. There are several types of processors that are available for use in such real-time systems and each such processor has its own architecture. This architecture determines the processing times of the jobs (where some jobs are faster on one architecture and slower on another while other jobs are faster on the later architecture or even cannot be processed on the first architecture). There is a small number of architectures and we model these as types of the machines where a machine corresponds to an available slot for installing a processor. Once a type of a machine is set, the processing times of the jobs is given and the goal is to find a schedule that minimizes the makespan. So a given collection of processors is a feasible solution if we are able to find a schedule whose makespan is the given time unit.

Outline of the scheme We apply geometric rounding of the parameters of the input (see Sect. 2), followed by a guessing step in which we guess for each type the minimum index of the machine that is activated to this type together with its approximated load (see Sect. 4). This guessing is motivated by a standard characterization of near-optimal solutions that is described earlier in Sect. 3. Based on these rounding and guessing steps, we formulate a mixed integer linear program (MILP) that is solved to optimality in polynomial time using [22,24] and the property that the number of integer variables is a constant (see Sect. 5). Furthermore, we prove that the optimal cost to our scheduling problem LBP is approximated by the solution of the MILP. Last, we use the solution of the MILP to round it into a feasible solution to problem LBP whose cost is approximately the cost of the solution of the MILP (see Sect. 6).

2 Rounding of the Input

In what follows we would like to assume that the speed of each machine is an integer power of $1 + \varepsilon$, and that for every job $j$ and type $t$, we have that $p_j(t)$ is an integer power of $1 + \varepsilon$. Given an instance $I$ of problem LBP that does not satisfy these conditions, we round down the speed of each machine $i$ to an integer power of $1 + \varepsilon$, and for each job $j$ and type $t$, we round up the value of $p_j(t)$ to an integer power of $1 + \varepsilon$. That is, we create a new rounded instance $I'$ in which the speed of machine $i$ is $s'_i$, and for each job $j$ and type $t$, we let $p'_j(t)$ be its size if it is assigned to a machine of type $t$, where we define
\[ s'_i = (1 + \varepsilon)^{\lfloor \log_{1+\varepsilon} s_i \rfloor} \quad \forall i, \quad p'_j(t) = (1 + \varepsilon)^{\lfloor \log_{1+\varepsilon} p_j(t) \rfloor} \quad \forall j, t . \] (2)

The other parameters of the input are left in \( I' \) as they were in \( I \). The analysis of this step is proved in the following lemma that follows using standard arguments. Recall that a feasible solution to LBP means selecting a type for each machine satisfying the total activation cost constraint (1) and specifying a job assignment function.

**Lemma 2** Given a feasible solution to \( I \) of cost \( C_I \), then the same solution is a feasible solution to \( I' \) of cost (evaluated as a solution to \( I' \)) at most \((1 + \varepsilon)^2 \phi \cdot C_I\). Given a feasible solution to \( I' \) of cost \( C_I' \), then the same solution is a feasible solution to \( I \) of cost (evaluated as a solution to \( I \)) at most \( C_I' \).

**Proof** Consider a job assignment function \( \sigma \), and a selection of type \( t_i \) for every machine \( i \). The feasibility conditions in \( I \) and in \( I' \) are the same because the total activation budget constraint (that is, (1)) is satisfied in \( I \) if and only if it is satisfied in \( I' \) as the activation cost functions as well as the value of \( \hat{A} \) are the same in the two instances. It remains to consider the cost of this assignment as a solution to \( I \) and as a solution to \( I' \). The total rejection penalty of the jobs rejected by \( \sigma \) is the same in the two instances.

Consider machine \( i \), and let \( \Lambda_i \) be its load in \( I \), and \( \Lambda'_i \) be its load in \( I' \) (with respect to the given solution). Assume that the solution we consider activates machine \( i \) as type \( t \). Then,

\[ \Lambda_i = \sum_{j \in J: \sigma(j) = i} \frac{p_j(t)}{s_i} \quad \text{and} \quad \Lambda'_i = \sum_{j \in J: \sigma(j) = i} \frac{p'_j(t)}{s'_i} . \]

By our definition of the rounding, i.e., by (2), we conclude that \( s'_i \leq s_i \leq (1 + \varepsilon) s'_i \) and for all \( j \), \((1 + \varepsilon) p_j(t) \geq p'_j(t) \geq p_j(t)\).

Therefore,

\[ \Lambda_i = \sum_{j \in J: \sigma(j) = i} \frac{p_j(t)}{s_i} \leq \sum_{j \in J: \sigma(j) = i} \frac{p'_j(t)}{s'_i} = \Lambda'_i , \]

while

\[ \Lambda'_i = \sum_{j \in J: \sigma(j) = i} \frac{p'_j(t)}{s'_i} \leq \sum_{j \in J: \sigma(j) = i} \frac{(1 + \varepsilon)^2 \cdot p_j(t)}{s_i} = (1 + \varepsilon)^2 \Lambda_i . \]

Thus, we conclude that for every machine \( i \) we have \( \Lambda_i \leq \Lambda'_i \leq (1 + \varepsilon)^2 \Lambda_i \). Therefore, by Lemma 1 and the definition of the objective function \( obj \), the claim follows. \( \square \)

Using this lemma and noting that applying the rounding step takes linear time, we conclude that without loss of generality with a slight abuse of notation, we assume that the input instance satisfies the properties that \( s_i \) and \( p_j(t) \) are integer powers of \( 1 + \varepsilon \) (for all \( i, j, t \)).
3 Characterization of Near-Optimal Solutions

We say that a feasible solution to LBP is nice if the following property holds. Let \( i < i' \), be a pair of machines that are activated to a common type \( t \) such that \( i \) is the minimum index of a machine that is activated to type \( t \), then the load of \( i \) is at least the load of \( i' \) times \( \varepsilon^2 \). The following lemma together with the guessing step described in the next section serve as an alternative to the dual approximation method of [15,16]. We use them for cases in which the dual approximation method does not work (i.e., non-bottleneck load balancing problems).

**Lemma 3** Given an instance of LBP and a feasible solution \( \text{SOL} \) of cost \( C(\text{SOL}) \), there exists a feasible solution \( \text{SOL}' \) that is a nice solution whose cost \( C(\text{SOL}') \) satisfies \( C(\text{SOL}') \leq (1 + \varepsilon)\Phi \cdot C(\text{SOL}) \).

**Proof** We apply the following process for modifying SOL into SOL’. The process changes the assignment of some jobs that are not rejected in SOL. Thus, the value of the total rejection penalty (i.e., \( \sum_{j \in J, \sigma(j) = 0} \pi_j \)) is left without modification. The process is defined for every type \( t \) (one type at a time) by changing the assignment of some jobs that are assigned to machines of type \( t \) and are moved to the lowest index machine of type \( t \).

Consider a fixed type \( t \) and let \( i \) be a machine of lowest index that is assigned type \( t \) in SOL. We will modify the set of jobs assigned to \( i \), and we let \( \lambda \) denote the current load of \( i \). We perform the following iteration until the first time where the cost of the solution is increasing (or we decide to stop and move to the next type). Let \( i'' \) be a machine of maximum load among the machines that are assigned type \( t \) in the current solution. If \( \lambda \) is at least the load of \( i'' \) times \( \varepsilon^2 \) (and in particular if \( i'' = i \)), we do nothing and continue to the next type (the condition of nice solutions is satisfied for type \( t \) by definition of maximum). Otherwise, we move all jobs assigned to machine \( i'' \) to be assigned to machine \( i \). We recalculate \( \lambda \) and the cost of the resulting solution and check if we need to apply the iteration again.

If we stop and move to the next type, then by the convexity and monotonicity of \( \Phi \), the new load of \( i \) is larger than the load of \( i'' \) prior to this iteration, and by the definition of \( i'' \) the condition of nice solutions is satisfied for type \( t \).

Consider a specific type \( t \) with \( i \) as defined above, and let \( i' \) be the value of \( i'' \) in the last iteration. We let \( \lambda' \) be the load of \( i \) just before the last iteration of the last procedure. Using Lemma 1 and the symmetry of \( \Phi \), it suffices to show that in the last iteration the maximum of the loads of \( i \) and \( i' \) is increased by a multiplicative factor of at most \( 1 + \varepsilon \).

There are two cases. In the first case assume that \( s_{i'} > \varepsilon s_i \). Consider first moving all jobs assigned to \( i \) (prior to the last iteration) to run on \( i' \). The resulting load of \( i' \) is increased by at most \( \frac{\lambda'}{\varepsilon} \), and this is at most \( (1 + \varepsilon) \) times the load of \( i' \) prior to the last iteration. Then by moving all the jobs that are assigned to \( i'' \) to run on \( i \) the load incurred by these jobs can only decrease and the claim follows. Otherwise, we conclude that \( s_{i'} \leq \varepsilon s_i \). Thus, by moving the jobs previously assigned to \( i' \) to be assigned to \( i \), the total processing time of these jobs is at most \( \varepsilon \) times the load of \( i' \) (in SOL). Since \( i' \) is selected as the machine of maximum load (of type \( t \) in the solution
prior to the last iteration), the new load of \( i \) is at most \( 1 + \varepsilon \) times the load of \( i' \) in the solution obtained prior to the last iteration.

\[ \square \]

4 Guessing Step

We apply a guessing step of (partial) information on an optimal solution among nice solutions. See e.g. [26] for an overview of this technique of guessing (or partitioning the solution space) in the design of approximation schemes.

In what follows, we consider one nice solution of minimal cost among all nice solutions to the (rounded) instance and denote this solution by \( \text{opt} \) and its cost (as a solution to \( \text{LBP} \)) by \( \text{Cost}(\text{opt}) \) together with its job assignment function \( \sigma^o \) and the type \( t^o_i \) assigned by \( \text{opt} \) to machine \( i \) (for all \( i \)).

We guess the approximated value of the makespan in \( \text{opt} \), and denote it by \( C_{\max}(\text{opt}) \). That is, if \( \text{opt} \) rejects all jobs then \( C_{\max}(\text{opt}) = 0 \), and otherwise the makespan of \( \text{opt} \) is in the interval \( (C_{\max}(\text{opt})/(1 + \varepsilon), C_{\max}(\text{opt})) \). Furthermore, for every type \( t \), we guess a minimum index \( \mu(t) \) of a machine of type \( t \) (namely, \( \mu(t) = \min \{ i : t^o_i = t \} \)), and its approximated load \( L_t \) that is a value such that the load of machine \( \mu(t) \) is in the interval \( (L_t - \varepsilon C_{\max}(\text{opt})/\tau, L_t) \]. If no machine is assigned to type \( t \), we let \( \mu(t) = m + 1 \) and \( L_t = 0 \). Without loss of generality, we assume that \( C_{\max}(\text{opt}) \geq \max_t L_t \).

Lemma 4 The number of different possibilities for the guessed information on \( \text{opt} \) is

\[ O(nm\tau \log_{1+\varepsilon} n \cdot ((m + 1)\tau/\varepsilon)^\tau) \cdot T(n, m, \varepsilon, \tau) \]

Proof To bound the number of values for \( C_{\max}(\text{opt}) \), let \( i \) be a machine where the makespan of \( \text{opt} \) is achieved and let \( j \) be the job of maximum size (with respect to type \( t^o_i \)) assigned to \( i \). Then, the makespan of \( \text{opt} \) is in the interval \( [p_j(t^o_i)/s_i, n \cdot p_j(t^o_i)/s_i] \), and thus the number of different values that we need to check for \( C_{\max}(\text{opt}) \) is \( O(nm\tau \log_{1+\varepsilon} n) \). For every type \( t \) we guess the index of machine \( \mu(t) \) (and there are \( m + 1 \) possible such indices) and there are at most \( \frac{\tau}{\varepsilon} + 1 \) integer multiplies of \( \varepsilon \cdot C_{\max}(\text{opt})/\tau \) that we need to check for \( L_t \) (using \( L_t \leq C_{\max}(\text{opt}) \)).

Remark 1 If we consider the model of machines with pre-specified types, then we do not need to guess the value of \( \mu(t) \) (for all \( t \)) and the number of different possibilities for the guessed information on \( \text{opt} \) is \( O(nm\tau \log_{1+\varepsilon} n \cdot (\tau/\varepsilon)^\tau) \).

Remark 2 In what follows we will present an algorithm with time complexity \( T(n, m, \varepsilon, \tau) \) for a given value of the guessed information. Then, the time complexity of the scheme will be \( O(nm\tau \log_{1+\varepsilon} n \cdot ((m + 1)\tau/\varepsilon)^\tau \cdot T(n, m, \varepsilon, \tau)) \) for the general case and \( O(nm\tau \log_{1+\varepsilon} n \cdot (\tau/\varepsilon)^\tau \cdot T(n, m, \varepsilon, \tau)) \) for the cases of machines with pre-specified types.
5 The Mixed Integer Linear Program

Our mixed integer linear program consists of three blocks and there is a limited interaction between those blocks. The first block is for assigning machines to types. This block will enforce our guessing of \( \mu(t) \) for all \( t \), and will enforce the activation budget constraint. The information that will be decided by this block and will be used by the third block is the number of machines of each rounded speed that are assigned to every type. The second block is for assigning jobs to types (where assigning a job to type 0 means we reject the job). Once again for each type and each possible job size the third block will use the number of jobs assigned to this type that have this rounded size (when assigned to a machine of this type). The third block is a configuration mixed integer linear program that decides upon the configuration of relatively large jobs of the machine as well as the total size of small jobs assigned to this machine. Similarly to [9] and unlike [17], our configurations encode the load of the machine and the notion of large and small jobs are based on this load of the machine.

In what follows we assume without loss of generality that \( \mu(t) \leq m \) for all \( t \) as other types can be omitted from the instance for the current value of the guessed information. Let \( \gamma \geq 10 \) be a constant that is chosen later (\( \gamma \) is a function of \( \tau \) and \( \varepsilon \) and we will use \( \gamma = \tau \cdot \frac{20}{\varepsilon^2} \)). For a type \( t \) and a real number \( W \), we say that job \( j \) is large for \( (t, W) \) if \( p_j(t) \geq \varepsilon \gamma \cdot W \), and otherwise it is small for \( (t, W) \).

Configurations Our MILP follows the configuration-MILP paradigm as one of its main ingredients. Thus, next we define our notion of configurations. A configuration \( C \) is a vector encoding partial information regarding the assignment of jobs to one machine where \( C \) consists of the following components: \( t(C) \) is the type assigned to a machine with configuration \( C \), \( s(C) \) is the speed of a machine with configuration \( C \), \( w(C) \) is an approximated upper bound on the total size of jobs assigned to a machine with this configuration where we assume that \( w(C) \) is an integer power of \( 1 + \varepsilon \) and the total size of jobs assigned to this machine is at most \( (1 + \varepsilon)^3 \cdot w(C) \), \( r(C) \) is an approximated upper bound on the total size of small jobs (small for \( (t(C), w(C)) \)) assigned to a machine with this configuration where we assume that \( r(C) \) is an integer multiple of \( \varepsilon \cdot w(C) \) and the total size of small jobs assigned to this machine is at most \( r(C) \), last, for every integer value of \( v \) such that \( (1 + \varepsilon)^v \geq \varepsilon \gamma \cdot w(C) \) we have a component \( \ell(C, v) \) counting the number of large jobs assigned to a machine of configuration \( C \) with size \( (1 + \varepsilon)^v \). Furthermore we assume that \( r(C) + \sum_v (1 + \varepsilon)^v \cdot \ell(C, v) \leq (1 + \varepsilon)^3 \cdot w(C) \). Let \( \mathcal{C} \) be the set of all configurations.

Lemma 5 For every pair \((s, w)\), we have

\[
|\{C \in \mathcal{C} : s(C) = s, w(C) = w\}| \leq \tau \cdot \left( \frac{2}{\varepsilon} \right)^{2\gamma^2 \log_{1+\varepsilon}(1/\varepsilon)}.
\]

The right hand side is an exponential function of \( 1/\varepsilon \) and \( \tau \) that we denote by \( \beta \).

Proof The number of types is \( \tau \), so \( t(C) \) has \( \tau \) possible values. The value of \( r(C) \) is an integer multiple of \( \varepsilon \cdot w \) that is smaller than \( (1 + \varepsilon)^3 w \leq 2w \) so there are at most \( \frac{2}{\varepsilon} \) such
values. The number of different values of \( v \) for which a job of size \((1 + \varepsilon)^v \geq \varepsilon \gamma \cdot w\) is smaller than \(2w\) is at most \( \log_{1 + \varepsilon} \frac{2}{\varepsilon \gamma} \leq 2 \gamma \cdot \log_{1 + \varepsilon} \frac{1}{\varepsilon} \) and for each such value of \( v \) we have that the value of \( \ell(C, v) \) is a non-negative integer smaller than \( \frac{2}{\varepsilon \gamma} \). This proves the claim using \( \gamma \geq 10 \). \( \square \)

Our MILP formulation involves several blocks and different families of variables that are presented next (these blocks have limited interaction). We present the variables and the corresponding constraints before presenting the objective function of the MILP.

**First block—machine assignment constraints** For every machine \( i \) and every type \( t \), we have a variable \( z_{i,t} \) that encodes if machine \( i \) is assigned type \( t \), where \( z_{i,t} = 1 \) means that machine \( i \) is assigned type \( t \). For every type \( t \), we have \( z_{i,t} = 0 \) for all \( i < \mu(t) \) while \( z_{\mu(t),t} = 1 \) enforcing our guessing. The (additional) machine assignment constraints are as follows:

For every machine \( i \), we require

\[
\sum_{t=1}^{\tau} z_{i,t} = 1,
\]

encoding the requirement that for every machine \( i \), exactly one type is assigned to \( i \).

Let \( m_s \) be the number of machines in the rounded instance of speed \( s \), then

\[
\sum_{t=1}^{\tau} \sum_{i:s_i = s} z_{i,t} = m_s.
\]

Last, we have the machine activation budget constraint

\[
\sum_{t=1}^{\tau} \sum_{i=1}^{m} \alpha_i(t) z_{i,t} \leq \hat{A}.
\]

The variables \( z_{i,t} \) are fractional variables and their number is \( O(m \tau) \). The rounding of the \( z_{i,t} \) will be based on solving a linear program with a totally unimodular constraint matrix where we enforce to maintain the values of \( \sum_{i:s_i = s} z_{i,t} \) close to the solution obtained for the MILP.

**Second block—job assignment to machine types and rejection constraints** For every job \( j \) and every \( t \in \{0, 1, \ldots, \tau\} \), we have a variable \( y_{j,t} \) that encodes if job \( j \) is assigned to machine that is activated to type \( t \) (for \( t \geq 1 \)) or rejected (for \( t = 0 \)). That is, for \( t \geq 1 \), if \( y_{j,t} = 1 \), then job \( j \) is assigned to machine of type \( t \), and if \( y_{j,0} = 1 \) it means that \( j \) is rejected (and we will pay the rejection penalty \( \pi_j \)). Furthermore for every type \( t \) and every possible integer value \( \xi \) we have two variables \( n(\xi, t) \) and \( n'(\xi, t) \) denoting the number of jobs assigned to machine of type \( t \) whose (rounded) size (if they are assigned to machine of type \( t \)) is \((1 + \varepsilon)^\xi\) that are assigned as large jobs and that are assigned as small jobs, respectively. Here, possible values of \( \xi \) for a given \( t \) are all integers for which \((1 + \varepsilon)^\xi \leq s_{\mu(t)} \cdot \min\{L_t/(\varepsilon^3), C_{max}(opt)\} \) such that the rounded input contains at least one job whose size (when assigned to a machine of type \( t \)) is \((1 + \varepsilon)^\xi\) (where recall that \( L_t \) is the guessed load of machine \( \mu(t) \) and \( C_{max}(opt) \))
is the guessed value of the makespan). We denote by $\zeta(t)$ the set of possible values of $\zeta$ for the given $t$. We implicitly use the variables $n(\zeta, t)$ and $n'(\zeta, t)$ for $\zeta \notin \zeta(t)$ (i.e., impossible values of $\zeta$) by setting those variables to zero.

The constraints that we introduce for this block are as follows:

For every job $j$, we should either assign it to a machine (of one of the types) or reject it, and thus we require that

$$\sum_{t=0}^{\tau} y_{j,t} = 1.$$ 

Furthermore, for every type $t$ and possible value of $\zeta$ (i.e., $\zeta \in \zeta(t)$) we require,

$$\sum_{j: p_j(t) = (1+\varepsilon)^t} y_{j,t} \leq n(\zeta, t) + n'(\zeta, t).$$

For the MILP formulation, the variables $y_{j,t}$ and $n(\zeta, t)$ are fractional, while the variables $n'(\zeta, t)$ are integer variables only if $\zeta \in \zeta(t)$ and $(1 + \varepsilon)^t \geq s_{\mu(t)} L_t \varepsilon^\gamma$ (and otherwise they are fractional). Observe that we introduce for this block $O(n\tau)$ fractional variables (excluding variables that are set to 0 corresponding to impossible values of $\zeta$) and $O(\tau \gamma \log_2(1+\varepsilon) \varepsilon)$ integer variables. Here, the rounding of $y_{j,t}$ will be based on solving an integer program with a totally unimodular constraint matrix while the rounding of $n(\zeta, t)$ and $n'(\zeta, t)$ will be simply by rounding up each component. This rounding up of $n'(\zeta, t)$ will be charged to the work of machine $\mu(t)$ as their total size is negligible with respect to the work of $\mu(t)$, and the rounding of $n(\zeta, t)$ will be considered together with the rounding of the variables of the third block.

Third block—configuration constraints

For every $C \in \mathcal{C}$ we have a variable $x_C$ denoting the number of machines of speed $s(C)$ activated to type $t(C)$ whose job assignment is according to configuration $C$.

For every $t$, we require that machine $\mu(t)$ has a configuration where $s_{\mu(t)} L_t$ is approximately $w(C)$. Thus, for every $t$, we will have the constraint

$$\sum_{C \in \mathcal{C}: s(C) = s_{\mu(t)}, t(C) = t, s_{\mu(t)} L_t \leq w(C) \leq (1+\varepsilon)^t s_{\mu(t)} L_t} x_C \geq 1.$$

For the MILP formulation, $x_C$ is required to be integer only if $C$ is a heavy configuration, where $C$ is heavy if $w(C) \geq (1+\varepsilon)^t s_{\mu(t)} L_t$. Observe that the number of integer variables is $O(\beta \tau \gamma^4 (\log_2(1+\varepsilon) \varepsilon)^2)$. The rounding of the configuration variables will be by rounding up each variable and assigning the work of the excess machines to machine $\mu(t)$ (of the same type $t$). Since the total work assigned to excess machines with non-heavy configurations of type $t$ is relatively small with respect to the work of $\mu(t)$, this rounding step is applicable.

It remains to consider the constraints bounding these variables together with the $n(\zeta, t)$, $n'(\zeta, t)$ and $z_{i,t}$ introduced for the earlier blocks. Here, the constraints have one sub-block for each type $t$. The sub-block of type $t$ (for $1 \leq t \leq \tau$) consists of the following constraints:
For every type $t$ and every (rounded) speed $s$ we cannot have more than $\sum_{i : s_i = s} z_{i,t}$ machines with configurations satisfying $t(C) = t$ and $s(C) = s$, and therefore we have the constraint

$$\sum_{C \in \mathcal{C} : t(C) = t, s(C) = s} x_C \leq \sum_{i : s_i = s} z_{i,t}.$$ 

For every $\zeta \in \zeta(t)$, we have that all the $n(\zeta, t)$ jobs of size $(1 + \varepsilon)\zeta$ that we guarantee to schedule on machine of type $t$ are indeed assigned to such machine as large jobs. Thus, we have the constraints

$$\sum_{C \in \mathcal{C} : t(C) = t} \ell(C, \zeta) \cdot x_C = n(\zeta, t).$$

The last constraints ensures that for every $\zeta \in \zeta(t)$, the total size of all jobs of size at least $(1 + \varepsilon)\zeta$ that are scheduled as small jobs fits the total area of small jobs in configurations for which $(1 + \varepsilon)\zeta$ is small with respect to $(t, w(C))$. Here, we need to allow some additional slack, and thus for configuration $C$ we will allow to use $r(C) + 2\varepsilon w(C)$ space for small jobs. Thus, for every integer value of $\zeta$ we have the constraint

$$\sum_{\zeta' \geq \zeta} n'(\zeta', t) \cdot (1 + \varepsilon)^{\zeta'} \leq \sum_{C \in \mathcal{C} : t(C) = t, \varepsilon^{\gamma(C)} \cdot w(C) > (1 + \varepsilon)^\zeta} (r(C) + 2\varepsilon w(C)) x_C.$$ 

Observe that while we define the last family of constraints to have an infinite number of constraints, we have that if when we increase $\zeta$, the summation on the left hand side is the same, then the constraint for the larger value of $\zeta$ dominates the constraint for the smaller value of $\zeta$. Thus, it suffices to have the constraints only for $\zeta \in \bigcup_{t=1}^T \zeta(t)$.

In addition to the last constraints we have the non-negativity constraints (of all variables).

**The objective function** Using these variables and (subject to these) constraints we define the minimization (linear) objective function of the MILP as

$$\psi \cdot C_{\max}(\text{OPT}) + (1 - \psi) \cdot \sum_{C \in \mathcal{C}} \left( \frac{w(C)}{s(C)} \right)^{\phi} x_C + \sum_{j=1}^n \pi_j \cdot y_{j,0}.$$ 

Our algorithm solves optimally the MILP and as described in the next section uses the solution for the MILP to obtain a feasible solution to problem LBP without increasing the cost too much. Thus, the analysis of the scheme is crucially based on the following proposition.

**Proposition 1** The optimal objective function value of the MILP is at most $(1 + \varepsilon)^{\phi}$ times $\text{COST(OPT)}$. 

$\square$ Springer
Proof Based on the nice solution OPT to the rounded instance, specified by the type $t_i^o$ assigned to machine $i$ (for all $i$) and the job assignment function $\sigma^o$, we specify a feasible solution to the MILP as follows. Later we will bound the cost of this feasible solution.

First, consider the variables introduced for the machine assignment block and its constraints. The values of $z_{i,j}$ are as follows: $z_{i,j} = 1$ and for $t \neq t_i^o$, we let $z_{i,t} = 0$. Then, for every type $t$, we will have $z_{i,t} = 0$ for $i < \mu(t)$ and $z_{\mu(t),t} = 1$ by our guessing. For every machine $i$, we have $\sum_{t=1}^{r} z_{i,t} = 1$ since every machine is assigned exactly one type. The constraint $\sum_{t=1}^{r} \sum_{i,s_{i,t} = s} z_{i,t} = m_x$ is satisfied as every machine is activated with exactly one type and thus contributes to exactly one of the indicators of the summation on the left hand side. Last, we have that the machine activation budget constraint $\sum_{i=1}^{s} \sum_{t=1}^{m} a_i(t) z_{i,t} \leq A$ is satisfied by our assignment of values to the variables as the left hand side is exactly the total activation cost of OPT, and OPT is a feasible solution to LBP.

Next, consider the other variables. For every job $j$, we let $y_{j,0} = 1$ if $j$ is rejected by OPT, and otherwise we let $y_{j,t} = 1$ if and only if the following holds for some value of $i$

$$\sigma^o(j) = i \text{ and } t_i^o = t.$$  

Observe that the total rejection penalty of the jobs that are rejected by OPT is exactly $\sum_j \pi_j y_{j,0}$. Next, we assign a configuration to every machine based on OPT. Consider a specific machine $i$, we let $C(i)$ be the configuration we define next. $t(C(i)) = t_i^o$ is the type assigned to machine $i$ by OPT and $s(C(i)) = s_i$ is its speed. The value of $w(C(i))$ is computed by rounding up the total size of jobs assigned to $i$ in OPT to the next integer power of $1 + \varepsilon$, $r(C(i))$ is computed by first computing the total size of small jobs assigned to $i$ and then rounding up to the next integer multiple of $\varepsilon \cdot w(C(i))$, last, for every integer value of $v$ such that $(1 + \varepsilon)^v \geq \varepsilon^v \cdot w(C(i))$, the component $\ell(C(i), v)$ counts the number of jobs assigned to $i$ whose size is $(1 + \varepsilon)^v$. Thus, the requirement $r(C(i)) + \sum_v (1 + \varepsilon)^v \cdot \ell(C(i), v) \leq (1 + \varepsilon)^3 \cdot w(C(i))$ is satisfied as the left hand side exceeds the total size of jobs assigned to $i$ in OPT by at most $\varepsilon w(C(i))$ and the right hand side exceeds the total size of jobs assigned to $i$ in OPT by at least $3\varepsilon w(C(i))$. For every configuration $C \in C$ we let $x_C$ be the number of machines whose assigned configuration is $C$. By our guessing, we conclude that

$$\sum_{C \in C : s(C) = s_{\mu(t)}, t(C) = t_i, s_{\mu(t)} \cdot L_t \leq w(C) \leq (1 + \varepsilon)^3 \cdot s_{\mu(t)} \cdot L_t} x_C \geq 1$$

is satisfied for every type $t$. For every type $t$ and every $\zeta \in \zeta(t)$, we let $n(\zeta, t)$ be $n(\zeta, t) = \sum_{i \in t(C(i)) = t} \ell(C(i), \zeta)$, and $n'(\zeta, t)$ be defined as $n'(\zeta, t) = \max \{0, \sum_{j : p_j(t) = t} y_{j,t} - n(\zeta, t)\}$. This completes the assignment of values to the variables that we consider.

Then, since for every job $j$, if $\sigma^o(j) \geq 1$, then $\sigma^o(j)$ has exactly one type, and otherwise $y_{j,0} = 1$, by definition of the values of the $y$-variables, we have $\sum_{t=0}^{r} y_{j,t} = 1$. Furthermore, for every type $t$ and $\zeta \in \zeta(t)$, by definition of $n'(\zeta, t)$, we have
\[ \sum_{j : p_j(t) = (1 + \varepsilon)\xi} y_{j,t} \leq n(\xi, t) + n'(\xi, t). \]

For every type \( t \) and every (rounded) speed \( s \), OPT does not have more than \( \sum_{i : s_i = s} z_{i,t} \) machines with configurations satisfying \( t(C) = t \) and \( s(C) = s \), and therefore we have the constraint

\[ \sum_{C \in \mathcal{C} : t(C) = t \land s(C) = s} x_C \leq \sum_{i : s_i = s} z_{i,t}. \]

For every \( \xi \in \xi(t) \), we have that

\[ \sum_{C \in \mathcal{C} : t(C) = t} \ell(C, \xi) \cdot x_C = \sum_{i : t(C(i)) = t} \ell(C(i), \xi) = n(\xi, t), \]

where the first equality holds by changing the order of summation (using the definition of \( x_C \)) and the second holds by the definition of the value of \( n(\xi, t) \). Last, for every machine \( i \) and every \( \xi \in \xi(t(C(i))) \), the total size of all jobs of size at least \( (1 + \varepsilon)\xi \) that are scheduled as small jobs on machine \( i \) is smaller than \( r(C(i)) \). Thus, for every integer value of \( \xi \), we have

\[ \sum_{\xi' \geq \xi} n'(\xi', t) \cdot (1 + \varepsilon)^{\xi'} \leq \sum_{C \in \mathcal{C} : t(C) = t \land s(C) > (1 + \varepsilon)^{\xi}} (r(C) + 2\varepsilon w(C)) x_C. \]

We summarize that the solution we defined is a feasible solution to the MILP (where all variables are integers, so the integrality constraints of the MILP hold as well).

Last, consider the objective function value of the solution we constructed. It is

\[ \psi \cdot C_{\text{max}}(\text{OPT}) + (1 - \psi) \cdot \sum_{C \in \mathcal{C}} \left( \frac{w(C)}{s(C)} \right)^{\phi} \cdot x_C + \sum_{j=1}^{n} \pi_j \cdot y_{j,0}. \]

By our guessing, we conclude that \( C_{\text{max}}(\text{OPT}) \) is at most \( 1 + \varepsilon \) times the makespan of OPT. By definition, for machine \( i \) whose assigned configuration is \( C(i) \) and its load in OPT is \( \Lambda_i^o \) we have \( \frac{w(C(i))}{s_i} \leq (1 + \varepsilon) \Lambda_i^o \). Therefore, we have that

\[ \psi \cdot C_{\text{max}}(\text{OPT}) + (1 - \psi) \cdot \sum_{C \in \mathcal{C}} \left( \frac{w(C)}{s(C)} \right)^{\phi} \cdot x_C + \sum_{j=1}^{n} \pi_j \cdot y_{j,0} \]

\[ \leq (1 + \varepsilon)^{\phi} \cdot F(\Lambda_1^o, \ldots, \Lambda_m^o) + \sum_{j=1}^{n} \pi_j \cdot y_{j,0}, \]

and using the fact that the total rejection penalty in OPT equals \( \sum_{j=1}^{n} \pi_j \cdot y_{j,0} \), the claim follows.
6 Transforming the Solution to the MILP into a Schedule

Consider the optimal solution \((z^*, y^*, n^*, n^*, x^*)\) for the MILP, our first step is to round up each component of \(n^*\) and \(n^*\). That is, we let \(\hat{n}(\zeta, t) = \lceil n^*(\zeta, t) \rceil\) and \(\hat{n}'(\zeta, t) = \lceil n^*(\zeta, t) \rceil\) for every \(\zeta\) and every \(t\).

Furthermore, we solve the following linear program (denoted as \((LP - y)\)). The constraint matrix of \((LP - y)\) is totally unimodular since each variable appears with coefficient 1 exactly once in the set of constraints \(\sum_{j=1}^n \pi_j \cdot y_{j,0} \leq 1\) and exactly once in the set of constraints \(\sum_{j \in J} \sum_{p=1}^n \pi_j \cdot y_{j,p} \leq 1\) for every \(p \in \{1, 2, \ldots, \tau\}\), \(\forall \zeta \in \zeta(t)\), \(\forall \zeta \in \zeta(t)\). Furthermore, the right hand side of \((LP - y)\) is integer, and therefore a basic optimal solution of this program is integer. Let \(\hat{y}\) be an optimal integer solution for this linear program:

\[
\begin{align*}
\min & \quad \sum_{j=1}^n \pi_j \cdot y_{j,0} \\
\text{subject to} & \quad \sum_{j \in J} \sum_{p=1}^n \pi_j \cdot y_{j,p} = 1, \quad \forall j \in J, \\
& \quad \sum_{j \in J} \sum_{p=1}^n \pi_j \cdot y_{j,p} \leq \hat{n}(\zeta, t) + \hat{n}'(\zeta, t), \quad \forall t \in \{1, 2, \ldots, \tau\}, \forall \zeta \in \zeta(t), \\
& \quad y_{j,p} \geq 0, \quad \forall j \in J, \forall t \in \{0, 1, \ldots, \tau\}.
\end{align*}
\]

We will assign jobs to types (and reject some of the jobs) based on the values of \(\hat{y}\), that is if \(\hat{y}_{j,t} = 1\) we will assign \(j\) to a machine of type \(t\) (if \(t \geq 1\)) or reject it (if \(t = 0\)). Since \(y^*\) is a feasible solution to \((LP - y)\) of cost that equals the total rejection penalty of the solution to the MILP, we conclude that the total rejection penalty of this (integral) assignment of jobs to types is at most the total rejection penalty of the solution to the MILP. In what follows we will assign \(\hat{n}(\zeta, t) + \hat{n}'(\zeta, t)\) jobs of size \((1 + \varepsilon)^{\hat{y}}\) to machines of type \(t\) (for all \(t\)).

The next step is to round up each component of \(x^*\), that is, let \(\hat{x}_C = \lceil x^*_C \rceil\), and allocate \(\hat{x}_C\) machines of speed \(s(C)\) that are activated as type \(t(C)\) and whose schedule follows configuration \(C\). These \(\hat{x}_C\) machines are partitioned into \(x'_C = \lfloor x^*_C \rfloor\) actual machines and \(\hat{x}_C - x'_C\) virtual machines. Both actual and virtual machines are not machines of the instance but temporary machines that we will use for the next step.

**Lemma 6** It is possible to construct (in polynomial time) an allocation of \(\hat{n}(\zeta, t)\) jobs of size \((1 + \varepsilon)^{\hat{y}}\) for all \(t, \zeta\) to (actual or virtual) machines that follow configurations in \(\{C \in C : t(C) = t, (1 + \varepsilon)^{\hat{y}} \geq \varepsilon^{\hat{y}} \cdot w(C)\}\), and of \(\hat{n}'(\zeta, t)\) jobs of size \((1 + \varepsilon)^{\hat{y}}\) for all \(t, \zeta\) to (actual or virtual) machines that follow configurations in \(\{C \in C : t(C) = t, (1 + \varepsilon)^{\hat{y}} \geq \varepsilon^{\hat{y}} \cdot w(C)\}\), such that for every machine that follows configuration \(C \in C\), the total size of jobs assigned to that machine is at most \((1 + \varepsilon)^{\hat{y}} w(C)\).

**Proof** First, we allocate the large jobs. That is, for every value of \(t\) and \(\zeta\) we allocate \(\hat{n}(\zeta, t)\) jobs whose size (when assigned to machine of type \(t\)) is \((1 + \varepsilon)^{\hat{y}}\) to (actual or virtual) machines that follow configurations in \(\{C \in C : t(C) = t, (1 + \varepsilon)^{\hat{y}} \geq \varepsilon^{\hat{y}} \cdot w(C)\}\). We allocate for every machine that follows configuration \(C\) exactly \((1 + \varepsilon)^{\hat{y}}\) such jobs (or less if there are no additional jobs of this size to allocate). Since \(\sum_{C \in C : t(C) = t} \ell(C, \zeta, C) \cdot x^*_C = n^*(\zeta, t)\), we allocate in this way at least \(n^*(\zeta, t)\) such jobs as large jobs. By the constraint \(\sum_{C \in C : t(C) = t} \ell(C, \zeta, C) \cdot x_C = n(\zeta, t)\), we have the following
\[
\sum_{C \in \mathcal{C} : t(C) = t} \ell(C, \xi) \cdot \hat{x}_C \geq \sum_{C \in \mathcal{C} : t(C) = t} \ell(C, \xi) \cdot x_C^*
\]
\[= n^*(\xi, t) \]
\[> \hat{n}(\xi, t) - 1 \]

Since both sides of the last sequence of inequalities are integer number, we conclude that \(\sum_{C \in \mathcal{C} : t(C) = t} \ell(C, \xi) \cdot \hat{x}_C \geq \hat{n}(\xi, t)\), and thus all these \(\hat{n}(\xi, t)\) jobs are assigned to machines that follow configurations in \(\{C \in C : t(C) = t, (1 + \varepsilon)^{\xi} \geq \varepsilon \gamma^* \cdot w(C)\}\) without exceeding the bound of \(\ell(C, \xi)\) on the number of jobs of size \((1 + \varepsilon)^{\xi}\) assigned to each such machine.

Next, consider the allocation of small jobs. We apply the following process for each type separately. We first allocate one small job of each size to machine \(\mu(t)\) where here we mean that if machine \(\mu(t)\) follows configuration \(C(\mu(t))\), then we will allocate one job of each size of the form \((1 + \varepsilon)^{\xi}\) for all \(\xi\) for which \((1 + \varepsilon)^{\xi-3} < \varepsilon \gamma^* \cdot w(C(\mu(t)))\). Observe that by the integrality of \(n^*(\xi, t)\), we conclude that these values of \(\xi\) are the only values for which we may have \(\hat{n}(\xi, t) \neq n^*(\xi, t)\). Let \(\tilde{n}'(\xi, t)\) denote the number of jobs of size \((1 + \varepsilon)^{\xi}\) that we still need to allocate. Since \(\gamma \geq 10\), this step increases the total size of jobs assigned to machine \(\mu(t)\) by at most \(\varepsilon \gamma^* \cdot w(C(\mu(t))) \cdot \sum_{h=0}^{\infty} \frac{1}{(1 + \varepsilon)^{h-3}} \leq \varepsilon w(C(\mu(t)))\).

Given a type \(t\), we sort the machines that follow configurations with type \(t\) according to the values of \(w(C)\) of the configuration \(C\) that they follow. We sort the machines in a monotonically non-increasing order of \(w(C)\). Similarly, we sort the collection of \(\tilde{n}'(\xi, t)\) jobs of size \((1 + \varepsilon)^{\xi}\) (for all values of \(\xi\)) in a non-increasing order of \(\xi\). We allocate the jobs to (actual or virtual) machines using the next fit heuristic. That is, we start with the first machine (in the order we described) as the current machine. We pack one job at a time to the current machine, whenever the total size of the jobs that are assigned to the current machine (that follows configuration \(C\)) exceeds \(r(C) + 2\varepsilon w(C)\) (it is at most \(r(C) + 3\varepsilon w(C)\) as we argue below), we move to the next machine and define it as the current machine. We need to show that all jobs are indeed assigned in this way and that whenever we pack a job into the current machine that follows configuration \(C\), the size of the job is smaller than \(\varepsilon \gamma^* \cdot w(C)\). If we append one (non-existing) extra machine of type \(t\) that follows a configuration with \(w(C) < \min_j p_j(t)/\varepsilon \gamma^*\), then it suffices to show that whenever we move to a new current machine that follows configuration \(C\), all the (small) jobs of size \((1 + \varepsilon)^{\xi}\) for \(\xi \in \{\xi': (1 + \varepsilon)^{\xi'} \geq \varepsilon \gamma^* \cdot w(C)\}\) are assigned. This last required property holds, as whenever the last set of values of \(\xi\) is changed, we know that the total size of the (small) jobs we already assigned is (unless all these jobs are assigned) at least

\[
\sum_{C \in \mathcal{C} : t(C) = t, \varepsilon \gamma^* \cdot w(C) > (1 + \varepsilon)^{\xi}} (r(C) + 2\varepsilon w(C)) \cdot \hat{x}_C
\]
\[\geq \sum_{C \in \mathcal{C} : t(C) = t, \varepsilon \gamma^* \cdot w(C) > (1 + \varepsilon)^{\xi}} (r(C) + 2\varepsilon w(C)) \cdot x_C^*\]
\[ \sum_{\zeta' \geq \zeta} n'(\zeta', t) \cdot (1 + \varepsilon)^{\zeta'} \geq \sum_{\zeta' \geq \zeta} \tilde{n}'(\zeta', t) \cdot (1 + \varepsilon)^{\zeta'}. \]

By allocating a total size of at most \( r(C) + 3\varepsilon w(C) \) of small jobs to a machine that follows configuration \( C \), the resulting total size of jobs assigned to that machine is at most

\[ (1 + \varepsilon)^3 w(C) + 3\varepsilon w(C) \leq (1 + \varepsilon)^6 w(C) \]

and the claim follows.

The assignment of jobs for which \( y_{j,0} \neq 0 \) to machines is specified by assigning every job that was assigned to a virtual machine that follows configuration \( C \) to machine \( \mu(t(C)) \) instead, and assigning the jobs allocated to actual machines by allocating every actual machine to an index in \( \{1, 2, \ldots, m\} \) following the procedure described in the next step. Before describing the assignment of actual machines to indices in \( \{1, 2, \ldots, m\} \) of machines in the instance (of problem LBP), we analyze the increase of the load of machine \( \mu(t) \) due to the assignment of jobs that were assigned to virtual machines that follow configurations with type \( t \).

**Lemma 7** Setting \( \gamma = \frac{20\varepsilon}{\varepsilon^3} \) guarantees that the resulting total size of jobs assigned to machine \( \mu(t) \) is at most \( (1 + \varepsilon)^8 \cdot w(C(\mu(t))) \) where machine \( \mu(t) \) follows the configuration \( C(\mu(t)) \).

**Proof** Since \( x^* \) is forced to be integral for all heavy configurations, we conclude that if there exists a virtual machine that follows configuration \( C \) with type \( t(C) = t \), then

\[ w(C) \leq \gamma^3 \cdot L_t(C) \cdot s_{\mu(t)}(C) \leq 2\gamma^3 w(C(\mu(t))), \]

where the last inequality holds using the constraint \( \sum_{C \in C(C)} s(C) = s_{\mu(t)}(C) = s_{\mu(t)}(C) \cdot L_t(C) \cdot s_{\mu(t)}(C) \geq 1 \) and allocating configuration \( C(\mu(t)) \) that causes \( x^* \) to satisfy this inequality, to machine \( \mu(t) \). For each configuration \( C \), there is at most one virtual machine that follows \( C \).

First, we upper bound the total size of jobs assigned to virtual machines of type \( t \) and speed at least \( \gamma^3 \cdot s_{\mu(t)} \). Observe that the number of distinct speeds of such virtual machines is at most \( 2 + \log_{1+\varepsilon} \frac{1}{\varepsilon^3} \leq 2 + \gamma^3 \cdot \log_{1+\varepsilon} \frac{1}{\varepsilon^3} \). Since there are at most \( \beta \) configurations with a common component of \( w(C) \) and the given type \( t \) and speed \( s \), we conclude that the total size of jobs assigned to virtual machines of type \( t \) and speed at least \( \gamma^3 \cdot s_{\mu(t)} \) is at most

\[ \beta \cdot \left( 2 + \gamma^3 \cdot \log_{1+\varepsilon} \frac{1}{\varepsilon^3} \right) \cdot w(C(\mu(t))) \cdot \gamma^3 \cdot \sum_{h=0}^{\infty} \frac{1}{(1 + \varepsilon)^h} \leq \beta \gamma^3 \cdot \varepsilon \gamma^3 \cdot w(C(\mu(t))). \]

Next, we upper bound the total size of jobs assigned to virtual machines of type \( t \) and speed smaller than \( \gamma^3 \cdot s_{\mu(t)} \) using the fact that \( \text{OPT} \) is nice. Here, for a fixed
speed $s < \varepsilon^{-3} \cdot s_{\mu(t)}$ the total size of jobs assigned to virtual machines of speed $s$ is (using that OPT is nice) at most

$$\beta \cdot w(C(\mu(t))) \cdot \frac{1}{\varepsilon^2} \cdot \frac{s_{\mu(t)}}{s} \cdot \sum_{h=0}^{\infty} \frac{1}{(1 + \varepsilon)^h},$$

and when we sum over all such $s$ the total size of jobs assigned to virtual machines of type $t$ and speed smaller than $\varepsilon^{-3} \cdot s_{\mu(t)}$ is at most

$$\beta \cdot w(C(\mu(t))) \cdot \frac{1}{\varepsilon^2} \cdot \sum_{h=0}^{\infty} \frac{1}{(1 + \varepsilon)^h} \cdot \varepsilon^{-3} \cdot \sum_{h'=0}^{\infty} \frac{1}{(1 + \varepsilon)^{h'}} \leq \beta \cdot w(C(\mu(t))) \cdot \varepsilon^{-3-5}.$$

Therefore, the total size of jobs that were assigned to virtual machines of type $t$ is at most

$$2\beta \cdot \gamma^{-3} \cdot \varepsilon^{-3-5} \cdot w(C(\mu(t))).$$

The claim will follow if the value of $\gamma$ satisfies that $2\beta \gamma^{-3} < \varepsilon^{-6-\gamma^{-3}}$.

Recall that $\beta = \tau \cdot (\frac{2}{\varepsilon})^{2\gamma^2 \log_1 + (1/\varepsilon)} \leq \tau \cdot (\frac{2}{\varepsilon})^{2\gamma^2 / \varepsilon^2}$. Thus, in order to ensure that $2\beta \gamma^{-3} < \varepsilon^{-6-\gamma^{-3}}$, it suffices to show that $\tau^4 < (\frac{2}{\varepsilon})^{(2\gamma^2 / \varepsilon^2) + 16 - \gamma^3}$. Observe that for every $\varepsilon > 0$ the function $H(\gamma) = (2\gamma^2 / \varepsilon^2) + 16 - \gamma^3$ decreases without bounds when $\gamma$ increases to $\infty$. Thus, we can select a value of $\gamma$ (as a function of $\tau$ and $\varepsilon$) for which the last inequality holds, e.g. selecting $\gamma = \tau \cdot 20 / \varepsilon^2$ is sufficient.

Next, we describe the assignment of actual machines to indices in $\{1, 2, \ldots, m\}$. More precisely, the last step is to assign a type $\hat{t}_i$ for every machine $i$ satisfying the total activation cost bound, and to allocate for every $C \in \mathcal{C}$ and for every actual machine that follows configuration $C$, an index $i$ such that $\hat{t}_i = t(C)$. This assignment of types will enforce our guessing of $\mu(t)$ for all $t$. This step is possible as we show next using the integrality of the assignment polytope.

**Lemma 8** There is a polynomial time algorithm that finds a type $t_i$ for every machine $i$, such that the total activation cost of all machines is at most $\hat{A}$, for all $s, t$ the number of machines of speed $s$ that are activated to type $t$ is at least the number of actual machines that follow configurations with type $t$ and speed $s$, and for all $t$ $\mu(t)$ is the minimum index of a machine that is assigned type $t$.

**Proof** We consider the following linear program (denoted as $(LP - z)$):

$$\min \sum_{i=1}^{\tau} \sum_{t=1}^{m} a_{i}(t)z_{i,t}$$

s.t.

$$\sum_{i=1}^{\tau} z_{i,t} = 1 \quad \forall i,$$

$$\sum_{i:s_i=s} z_{i,t} \geq \sum_{C \in \mathcal{C}; t(C)=t, s(C)=s} x_C' \quad \forall s, \forall t,$$

$$z_{i,t} = 0 \quad \forall t, \forall i < \mu(t)$$

$$z_{\mu(t),t} = 1 \quad \forall t$$

$$z_{i,t} \geq 0 \quad \forall i \forall t.$$
The constraint matrix of $(LP - z)$ is totally unimodular. Once again the reason is that each decision variable $z_{i,t}$ appears once (with coefficient 1) in the family of constraints $\sum_{t=1}^{T} z_{i,t} = 1, \forall i$ and once (with coefficient 1) in the family of constraints $\sum_{i:s_i=s} z_{i,t} \geq \sum_{C \in \mathcal{C} : t(C) = t, s(C) = s} x_C', \forall s, \forall t$, while the other constraints simply restrict ourselves to a face of the polytope defined by these two families of constraints and the non-negativity constraints. Thus, by solving the linear program and finding an optimal basic solution, we find an optimal integer solution in polynomial time. Since the fractional solution $z^*$ is a feasible solution with objective function value that does not exceed $\hat{A}$, we conclude that the optimal integer solution that we find does not violate the upper bound on the total activation cost. This integer solution defines a type $t_i$ for every machine $i$ by letting $t_i$ be the value for which $z_{i,t} = 1$. Then using the constraints of the linear program for every speed $s$ and every type $t$, the number of machines of speed $s$ for which we define type $t$ is at least the number of actual machines that follow configurations with type $t$ and speed $s$ and furthermore for every type $t$ $\mu(t)$ is the minimum index of a machine that is assigned type $t$, as required. \qed

Thus, we conclude:

**Theorem 1** Problem LBP admits an EPTAS.

**Proof** The time complexity of the scheme is $O(f(\frac{1}{\varepsilon}, \gamma, \tau) \cdot m^{O(\tau)} \cdot \text{poly}(n))$, and $\gamma = \frac{20 \tau}{\varepsilon^2}$. The function $f$ is a single exponential function of the number of integer variables in the mixed-integer formulation that we used. So if $d$ variables are forced to be integer, this function is $d^{O(d)}$ (see [22,24]). In our formulation this number of variables that are forced to be integer is $O((\beta \tau)^{\frac{1}{\varepsilon}} \cdot \text{poly}(n))$. Thus, the time complexity of the scheme is upper bounded by a doubly exponential function of $\gamma$ times a polynomial of the input length where the degree of the polynomial is linear in $\tau$. This linear dependency of the degree of the polynomial in $\tau$ does not exist for the special cases of machines with pre-specified types (where the degree of the polynomial is a constant).

Thus, in order to show that the algorithm is an EPTAS, it suffices to prove its approximation ratio, and that the resulting solution is feasible. Based on the sequence of lemmas, the approximation ratio is proved, using the fact that the load of an empty set of jobs is zero no matter what is the type of the machine and thus for every $i$, if machine $i$ is assigned an empty set of jobs the objective function value does not depend on the type assigned to $i$. Thus, in the last step the cost of the solution does not increase.

Thus, we need to show that the resulting solution is feasible. Note that every job assignment is feasible, the feasibility of the assignment of machines to types is feasible using Lemma 8. Thus, our solution is a feasible solution to problem LBP. \qed

### 7 Conclusions

We presented a general load balancing model that allows us to demonstrate the existence of EPTAS’s for all its special cases. In this work, our goal was to show the
existence of such a scheme but we have not tried to optimize the resulting time complexity of the scheme. We expect that such improvements are possible either to this general case or for most of its special cases.

References

1. Alon, N., Azar, Y., Woeginger, G.J., Yadid, T.: Approximation schemes for scheduling on parallel machines. J. Sched. 1(1), 55–66 (1998)
2. Bansal, N., Pruhs, K.R.: Server scheduling to balance priorities, fairness, and average quality of service. SIAM J. Comput. 39(7), 3311–3335 (2010)
3. Bartal, Y., Leonardi, S., Marchetti-Spaccamela, A., Sgall, J., Stougie, L.: Multiprocessor scheduling with rejection. SIAM J. Discrete Math. 13(1), 64–78 (2000)
4. Bonifaci, V., Wiese, A.: Scheduling unrelated machines of few different types (2012). arXiv:1205.0974
5. Cesati, M., Trevisan, L.: On the efficiency of polynomial time approximation schemes. Inf. Process. Lett. 64(4), 165–171 (1997)
6. Chen, L., Jansen, K., Zhang, G.: On the optimality of exact and approximation algorithms for scheduling problems. J. Comput. Syst. Sci. 96, 1–32 (2018)
7. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Berlin (1999)
8. Epstein, L., Levin, A.: An efficient polynomial time approximation scheme for load balancing on uniformly related machines. Math. Program. 147, 1–23 (2013)
9. Epstein, L., Levin, A.: Minimum total weighted completion time: faster approximation schemes (2014). arXiv:1404.1059
10. Epstein, L., Sgall, J.: Approximation schemes for scheduling on uniformly related and identical parallel machines. Algorithmica 39(1), 43–57 (2004)
11. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Berlin (2006)
12. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York (1979)
13. Gehrke, J.C., Jansen, K., Kraft, S.E.J., Schikowski, J.: A PTAS for scheduling unrelated machines of few different types. In: Proceeding of the 42nd International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'16), pp. 290–301 (2016)
14. Hochbaum, D.S.: Various notions of approximations: good, better, best and more. In: Hochbaum, D.S. (ed.) Approximation Algorithms. PWS Publishing Company, Boston (1997)
15. Hochbaum, D.S., Shmoys, D.B.: Using dual approximation algorithms for scheduling problems: theoretical and practical results. J. ACM 34(1), 144–162 (1987)
16. Hochbaum, D.S., Shmoys, D.B.: A polynomial approximation scheme for scheduling on uniform processors: using the dual approximation approach. SIAM J. Comput. 17(3), 539–551 (1988)
17. Jansen, K.: An EPTAS for scheduling jobs on uniform processors: using an MILP relaxation with a constant number of integral variables. SIAM J. Discrete Math. 24(2), 457–485 (2010)
18. Jansen, K., Klein, K.-M., Verschae, J.: Closing the gap for makespan scheduling via sparsification techniques. In: Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming (ICALP’16), vol. 72, pp. 1–13 (2016)
19. Jansen, K., Maack, M.: An EPTAS for scheduling on unrelated machines of few different types. In: Proceedings of the 15th International Symposium on Algorithms and Data Structures (WADS’17), pp. 497–508 (2017)
20. Jansen, K., Maack, M.: An EPTAS for scheduling on unrelated machines of few different types. CoRR (2017). arXiv:1701.03263 (v2)
21. Jansen, K., Maack, M., Rau, M.: Approximation schemes for machine scheduling with resource (in-) dependent processing times. In: Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA’16), pp. 1526–1542 (2016)
22. Kannan, R.: Improved algorithms for integer programming and related lattice problems. In: Proceedings of 15th Symposium on Theory of Computing (STOC), pp. 193–206. ACM (1983)
23. Khuller, S., Li, J., Saha, B.: Energy efficient scheduling via partial shutdown. In: Proceedings of 21st Symposium on Discrete Algorithms (SODA), pp. 1360–1372. ACM/SIAM (2010)
24. Lenstra Jr., H.W.: Integer programming with a fixed number of variables. Math. Oper. Res. 8(4), 538–548 (1983)
25. Marx, D.: Parameterized complexity and approximation algorithms. Comput. J. 51(1), 60–78 (2008)
26. Schuurman, P., Woeginger, G.J.: Approximation schemes—a tutorial (2001). http://www.win.tue.nl/~gwoegi/papers/ptas.pdf
27. Wiese, A., Bonifaci, V., Baruah, S.K.: Partitioned EDF scheduling on a few types of unrelated multiprocessors. Real-Time Syst. 49(2), 219–238 (2013)

Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.