A Robust PTAS for Machine Covering and Packing[⋆]

Martin Skutella and José Verschae

Institute of Mathematics, TU Berlin, Germany
{skutella,verschae}@math.tu-berlin.de

Abstract. Minimizing the makespan or maximizing the minimum machine load are two of the most important and fundamental parallel machine scheduling problems. In an online scenario, jobs are consecutively added and/or deleted and the goal is to always maintain a (close to) optimal assignment of jobs to machines. The reassignment of a job induces a cost proportional to its size and the total cost for reassigning jobs must preferably be bounded by a constant r times the total size of added or deleted jobs. Our main result is that, for any $\varepsilon > 0$, one can always maintain a $(1+\varepsilon)$-competitive solution for some constant reassignment factor $r(\varepsilon)$. For the minimum makespan problem this is the first improvement of the $(2+\varepsilon)$-competitive algorithm with constant reassignment factor published in 1996 by Andrews, Goemans, and Zhang.

1 Introduction

We consider two basic scheduling problems where n jobs need to be assigned to m identical parallel machines. Each job j has a processing time $p_j \geq 0$ and the load of a machine is the total processing time of jobs assigned to it. The *machine covering problem* asks for an assignment of jobs to machines that maximizes the minimum machine load. In the *minimum makespan problem* (or *machine packing problem*), we wish to find a schedule minimizing the maximum machine load.

Both problems are well known to be strongly NP-hard and both allow for a polynomial-time approximation scheme (PTAS); see, e. g., [2,9,16]. They have also been studied extensively in the online setting where jobs arrive one by one and must immediately be assigned to a machine at their arrival; see, e. g., [1,13]. The best known online algorithm for the minimum makespan problem is a 1.9201-competitive algorithm [8]. The best lower bound on the competitive ratio of any deterministic online algorithm currently known is 1.88 [10]. For randomized online algorithms there is a lower bound of $e/(e-1) \approx 1.58$; see [6,12].

The online variant of the machine covering problem turns out to be less tractable and there is no online algorithm with constant competitive ratio. The best possible deterministic algorithm greedily assigns jobs to the least loaded machine, and has competitive ratio $1/m$; see [16]. In [5] an upper bound of $O(1/\sqrt{m})$ is shown for the competitive ratio of any randomized online algorithm, and there is an almost matching $\tilde{\Omega}(1/\sqrt{m})$-competitive algorithm.

[⋆] Supported by Berlin Mathematical School and by DFG research center Matheon.

There is also a large amount of literature on *online load balancing* problems where jobs are allowed to arrive or depart the system; see, e. g., [4]. Most authors consider a relaxed notion of competitive ratio, where the solution is not compare against the current offline optimum but the largest objective value seen so far.

Proportional reassignment cost. We study a relaxed online scenario known as *online load balancing with proportional reassignment cost.* In this setting, jobs may arrive or depart at any time, and when a new job enters the system it must immediately be assigned to a machine. Again, the objective is either to minimize the makespan or to maximize the minimum machine load. Furthermore, upon arrival or departure of a job, one is allowed to reassign other jobs by paying an associated cost: reassigning job j incurs a cost of $c \cdot p_j$ for some given constant $c > 0$. By scaling we can assume that $c = 1$.

The cost due to reassignments is controlled by the *reassignment factor* which is defined as follows. Let J be the set of jobs that have so far appeared in the system, and let $J_L \subseteq J$ be the set of jobs that have left the system. We define the reassignment factor r of an algorithm as the worst case ratio between $\sum_{j \in J} p_j + \sum_{j \in J_L} p_j$ and the total cost due to reassignments. Alternatively, we can interpret this framework in the following way: given a parameter $r > 0$, the arrival or departure of a job j adds an amount of $r \cdot p_j$ to the total budget available to spend on reassignments. We call $r \cdot p_j$ the *reassignment potential* induced by job j.

Note that $r = 0$ means that no reassignment is allowed, and thus we are in the classical online setting. On the other hand, $r = \infty$ implies that we are allowed to reassign all jobs at each arrival/departure, and thus we fall back to the offline case. We are interested in developing α-competitive algorithms where the migration factor r is bounded by a constant. Furthermore, we study the trade-off between α and r. Arguably, the best that we can hope for under this framework is a *robust PTAS* (also known as *dynamic PTAS*), that is, a family of polynomial-time $(1 + \varepsilon)$-competitive algorithms with constant reassignment factor $r = r(\varepsilon)$, for all $\varepsilon > 0$.

For the minimum makespan problem with proportional reassignment cost, Westbrook [15] gives a 6-competitive algorithm with reassignment factor 1 (according to our definition[1]). Andrews, Goemans, and Zhang [3] improve upon this result, obtaining a 3.5981-competitive algorithm with reassignment factor 1. Furthermore, they give a $(2 + \varepsilon)$-competitive algorithm with constant reassignment factor $r(\varepsilon) \in O(1/\varepsilon)$.

Related work. Sanders, Sivadasan, and Skutella [11] consider a somewhat tighter online model, known as the *bounded migration* framework. This model can be interpreted as the reassignment model with the following modification: after the arrival or departure of a job j, its reassignment potential $r \cdot p_j$ must be

[1] Our definition differs slightly from the one given in [15]: they do not consider the departure of jobs to add any reassignment potential, and the first assignment of a job also induces cost in their case. However, the concept of constant reassignment factors is the same in both models.

immediately spent or is otherwise lost. In the bounded migration scenario, the value r is called the *migration factor* of the algorithm, and is a measure of the robustness of the constructed solutions.

Sanders et al. study the bounded migration model for the special case when jobs are not allowed to depart. For the minimum makespan problem, they give a 3/2-competitive algorithm with migration factor 4/3. Moreover, using well known rounding techniques, they formulate the problem as an integer linear programming (ILP) feasibility problem in constant dimension. Combining this with an ILP sensitivity analysis result, they obtain a robust PTAS for the bounded migration model with job arrivals only. An important consequence of their analysis is that no special structure of the solutions is needed to achieve robustness. More precisely, it is possible to take an arbitrary $(1+\varepsilon)$-approximate solution and, at the arrival of a new job, turn it into a $(1+\varepsilon)$-approximate solution to the augmented instance while keeping the migration factor constant. This feature prevents their technique from working in the job departure case.

The machine covering problem is also considered by Sanders et al. [11]. They describe an interesting application of the online version of this problem in the context of storage area networks and describe a 2-competitive algorithm with migration factor 1. Moreover, they give a counterexample showing that it is not possible to start with an arbitrary $(1/2+\varepsilon)$-approximate solution, and then maintain the approximation guarantee while keeping the migration factor constant. This implies that the ideas developed in [11] for the minimum makespan problem cannot be applied directly to derive a robust PTAS for the machine covering problem. Based on ideas in [11], Epstein and Levin [7] develop a robust APTAS for the Bin-Packing problem.

Our Contribution. We develop a general framework for obtaining robust PTASes in the reassignment model. Our results can be considered from various different angles and have interesting interpretations in several different contexts:

(i) We make a significant contribution to the understanding of two fundamental *online scheduling* problems on identical parallel machines that are also relevant building blocks for many more complex real-world problems.
(ii) We advance the understanding of *robustness* of parallel machine schedules under job arrival and departure, and give valuable insights related to the *sensitivity analysis* of parallel machine schedules.
(iii) We achieve the best possible performance bound for *machine balancing* with proportional reassignment costs, improving upon earlier work by Westbrook [15] and Andrews, Goemans, and Zhang [3].

Our techniques for deriving the robust PTAS take the ideas in [2] and [11] one step further. We first prove that it is not possible to start with an arbitrary $(1-\varepsilon)$-approximate solution and, at the arrival of a new job, maintain the competitive ratio with constant migration factor. One of our main contributions is to overcome this limitation by giving extra structure to the constructed solutions. Roughly speaking, we do this by asking for solutions such that the sorted vector of machine load values is lexicographically optimal. It turns out that a solution with this property is not only optimal but also robust. In the analysis

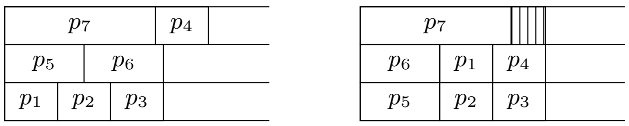

Fig. 1. *Left:* Unique optimal solution to original instance. *Right:* Unique optimal solution to instance with new jobs.

we formulate a rounded scheduling problem as an ILP in constant dimension, exploit the structure of the coefficient matrix, and apply sensitivity analysis for ILPs to derive the result.

To keep the presentation short and clear, we mainly focus on the machine covering problem in this extended abstract and present a robust PTAS for the general case of jobs leaving and entering the system. An easy adaptation of the techniques here presented yields a robust PTAS for the minimum makespan problem, improving upon the $(2+\varepsilon)$-competitive algorithm with constant reassignment factor by Andrews, Goemans, and Zhang [3]. Moreover, all our techniques can be extended to a very broad class of problems, where the objective functions solely depend on the load of each machine. Further details on these topics, as well as the complete proofs of our results can be found in [14].

2 A Lower Bound on the Best Approximation with Constant Migration Factor

We start by showing that it is not possible to maintain near-optimal solutions to the machine covering problem with constant migration factor in the model of Sanders et al. [11], if arriving jobs are arbitrarily small.

Lemma 1. *For any $\varepsilon > 0$, there is no $(19/20+\varepsilon)$-competitive algorithm for the machine covering problem with constant migration factor, even for the special case without job departures.*

The proof of the lemma is based on an instance consisting of 3 machines and 7 jobs depicted in Figure 1. Details can be found in [14]. The proof of the lemma can be found in [14].

As mentioned before, this lemma justifies the use of the reassignment cost model instead of the bounded migration framework. Moreover, we see in the proof of the lemma that the limitation of the bounded migration model is caused by arbitrarily small jobs, whose reassignment potential do not allow any other job to be migrated. Nonetheless, in the reassignment model we can deal with small jobs by accumulating them as follows.

Let OPT denote the value of an optimum solution for the current set of jobs. If a new job j with $p_j < \varepsilon \cdot OPT$ arrives, we do not schedule it immediately[2]. Instead,

[2] In order to still satisfy the strict requirements of the considered online scheduling problem, we can assume that job j is temporarily assigned to an arbitrary machine, say machine 1. Notice that this causes an increase of the reassignment factor by at most 1.

we accumulate several small jobs, until their total processing time surpasses $\varepsilon \cdot OPT$. We can then incorporate them as one larger job with processing time at least $\varepsilon \cdot OPT$. This can only decrease the value of the solution by a $1 - \varepsilon$ factor.

The situation for the departure of small jobs is slightly more complicated. We ignore the fact that certain small jobs are gone as long as the following property holds: There is no machine which has lost jobs of total processing time at least $\varepsilon \cdot OPT$. Under this condition, the objective function is affected by less than a factor $1+\varepsilon$. If, on the other hand, there is such a machine, we can treat the set of jobs that have left the machine as one single job of size at least $\varepsilon \cdot OPT$ and act accordingly. Notice that the property above has to be checked dynamically after each reassignment of jobs caused by newly arriving or departing jobs.

Assumption 1. *W.l.o.g., all arriving/departing jobs are bigger than $\varepsilon \cdot OPT$.*

3 A Stable Estimate of the Optimum Value

In this section we describe an upper bound on the optimum solution value of the machine covering problem, also introduced in [2]. However, for it to be useful for the robust PTAS, we need to show that this upper bound is stable. That is, at the arrival/departure of a new job, its value must not change by more than a constant factor.

Let $\mathcal{I} = (J, M)$ be an instance of our problem, where J is a set of n jobs and M a set of m machines. Given a subset of jobs L, we denote by $p(L)$ the total processing time of jobs in L, i. e., $p(L) := \sum_{j \in L} p_j$. Instance $\mathcal{I}$ satisfies property $(*)$ if $p_j \leq p(J)/m$, for all $j \in J$. The most natural upper bound to use for our problem is the average load $p(J)/m$. Under condition $(*)$, the average load is always within a factor 2 of OPT; see [2,14].

Lemma 2. *If instance $\mathcal{I}$ satisfies $(*)$, then $\frac{p(J)}{2m} \leq OPT \leq \frac{p(J)}{m}$.*

Now we show how to transform arbitrary instances to instances satisfying $(*)$ without changing the optimal solution value. If $p_j > p(J)/m \geq OPT$, then we can assume that j is being processed on a machine of its own. Thus, removing j plus its corresponding machine does not change the optimal solution value, but it does reduce the average load. We can iterate this idea until no job is strictly larger than the average load. We call this procedure Algorithm Stable-Average. Also, we call L the set of jobs and w the number of machines of the corresponding remaining instance. More importantly, we define A to be the average load of this instance, i. e., $A := p(L)/w$. We call value A the *stable average* of instance $\mathcal{I}$. Also, we obtain that solving the instance with job set L and w identical machines is equivalent to solving $\mathcal{I}$. Thus Lemma 2 yields:

Lemma 3. *The upper bound A computed by the algorithm above satisfies $OPT \leq A \leq 2 \cdot OPT$.*

It is easy to see that, in general, the factor by which the upper bound changes at the arrival/departure of a job is not bounded (consider two machines and two jobs of sizes 1 and $K \gg 1$, respectively; then one job of size $K-1$ arrives). However, we can show that if A is increased by more than a factor 2, then the instance was trivial to solve in the first place. To this end, we show that if the value A is increased by more than a factor of 2, then a significant amount of jobs must have arrived to the system. The proof of the next lemma is omitted.

Lemma 4. *Consider two arbitrary instances $\mathcal{I} = (J, M)$ and $\mathcal{I}' = (J', M)$. Let A, L and w (resp. A', L' and w') be the returned values when applying Algorithm* Stable-Average *to $\mathcal{I}$ (resp. $\mathcal{I}'$). If $A' > 2A$, then $|J \triangle J'| > w/2$ (*here $\triangle$ denotes the symmetric difference between the two sets*).*

Moreover, we say that an instance is *trivial* if Algorithm Stable-Average returns $w = 1$. In this case, the optimal solution to the instance can be constructed by processing the $m-1$ largest jobs each on a machine of their own, and the remaining jobs on the remaining machine. Moreover, the optimal value OPT equals A. With this definition, we obtain the following easy consequence of Lemma 4.

Corollary 1. *Assume that $\mathcal{I}$ is nontrivial and that instance $\mathcal{I}'$ is obtained from $\mathcal{I}$ by adding one job. Then, it must hold that $A \leq A' \leq 2 \cdot A$.*

4 The Structure of Robust Solutions

In the following, we show a sufficient condition to guarantee that we can achieve near optimal solutions when jobs arrive or depart. For clarity, we first consider a static case: Given an instance $\mathcal{I}$, we construct a $(1-O(\varepsilon))$-approximate solution having enough structure so that at the arrival or departure of a job larger than $\varepsilon \cdot OPT$, we can maintain the approximation guarantee using constant migration factor. Note that since we are using constant migration factor, we only use the reassignment potential induced by the arriving or departing job. Nonetheless, we do not take care of maintaining the structure so that this procedure can be iterated when further jobs arrive (or depart). We deal with this more complicated scenario in Section 5.

We concentrate on the case of a newly arriving job. But the presented ideas and techniques can be easily adapted to the case of a departing job. Let $\mathcal{I} = (J, M)$ be an arbitrary instance with optimal value OPT. If there is no possible confusion, we will also use OPT to refer to some optimal schedule for $\mathcal{I}$. We call $\mathcal{I}' = (J', M)$ the instance with the additional arriving job p_{j^*}, and OPT' the new optimal value.

Lemma 5. *Assume that $\mathcal{I}$ is trivial. Then, starting from an optimal solution, one can construct a $(1-\varepsilon)$-approximate solution to $\mathcal{I}'$ by using migration factor at most $2/\varepsilon$.*

The proof of the lemma can be found in [14]. We devote the rest of this section to the case of nontrivial instances.

4.1 Compact Description of a Schedule

As usual in PTASes, we first simplify our instance by rounding. In this section we briefly show how to do this for our problem. The techniques are similar to the ones found, e. g., in [2,11,16]. Nonetheless, we must be careful to ensure that the resulting compact description of schedules is also compatible with schedules containing any new job that may arrive.

It is a well known fact that by only loosing a $1/(1+\varepsilon)$ factor in the objective function, we can round down all processing times to the nearest power of $1+\varepsilon$. Thus, in the rest of this paper we assume that, for every job j, it holds that $p_j = (1+\varepsilon)^k$ for some $k \in \mathbb{Z}$. Moreover, we need to compute an upper bound, UB, which is within a constant factor $\gamma > 1$ of the optimal value: $OPT \leq \text{UB} \leq \gamma \cdot OPT$. Throughout this section we use $\text{UB} = A$, so that $\gamma = 2$. For the general case in Section 5, however, we have to choose this upper bound more carefully.

In what follows, we round our instance such that the number of different processing times is constant. To this end, let $\sigma, \Sigma \geq 1$ be two constant parameters that will be chosen appropriately later. Our rounding ensures that all processing times belong to the interval $[\varepsilon \cdot \text{UB}/\sigma, \Sigma \cdot \text{UB}]$. The value σ will be chosen big enough so that every job that is smaller than $\varepsilon \cdot \text{UB}/\sigma$ is also smaller than $\varepsilon \cdot OPT$. On the other hand, since $\Sigma \geq 1$, every job that is larger than $\Sigma \cdot \text{UB}$ is also larger than OPT, and thus should be processed on a machine of its own. Moreover, since we are assuming that $\mathcal{I}$ is nontrivial, Corollary 1 implies that $\text{UB}' := A' \leq 2 \cdot \text{UB}$. We can therefore choose $\Sigma \geq 2$ to ensure that a job that is larger than $\Sigma \cdot \text{UB}$ is also larger than OPT', and thus can also be processed on a machine of its own in optimal solutions to $\mathcal{I}'$. This will help to simultaneously round $\mathcal{I}$ and $\mathcal{I}'$, yielding the same approximation guarantee for both instances. More importantly, we note that since the lower and upper bounds are within a constant factor, the rounded instances only have $O(\log_{1+\varepsilon}(1/\varepsilon)) = O(1/\varepsilon \log(1/\varepsilon))$ different job sizes. Consider the index set

$$I(\text{UB}) := \left\{ i \in \mathbb{Z} : \varepsilon \cdot \text{UB}/\sigma \leq (1+\varepsilon)^i \leq \Sigma \cdot \text{UB} \right\} = \{\ell, \ldots, u\} .$$

The new rounded instance derived from $\mathcal{I}$ is described by defining a vector $N = (n_i)_{i \in I}$, whose entry n_i denotes the number of jobs of size $(1+\varepsilon)^i$. More precisely, vector N is defined as follows. For each $i = \ell + 1, \ldots, u-1$, we let

$$n_i := \left| \left\{ j \in J : p_j = (1+\varepsilon)^i \right\} \right| , \tag{1}$$

i. e., n_i is the number of jobs of size $(1+\varepsilon)^i$ in the original instance. We call these jobs *big* with respect to UB. Bigger jobs are rounded down to $(1+\varepsilon)^u$, i. e., we set

$$n_u := \left| \{ j \in J : p_j \geq (1+\varepsilon)^u \} \right| . \tag{2}$$

We call these jobs *huge* with respect to UB. Finally, jobs that are smaller than or equal to $(1+\varepsilon)^\ell$ are said to be *small* with respect to UB. We replace them by jobs of size $(1+\varepsilon)^\ell$, i. e., we set

$$n_\ell := \left\lfloor \tfrac{1}{(1+\varepsilon)^\ell} \sum\nolimits_{j : p_j \leq (1+\varepsilon)^\ell} p_j \right\rfloor . \tag{3}$$

By definition (3), the total processing time of small jobs in N and $\mathcal{I}$ is roughly equal. By slightly abusing notation, in what follows we also use the symbol N to refer to the scheduling instance defined by the vector N. The next lemma is also used in [2]. We omit the proof.

Lemma 6. *The value of an optimal solution to N is within a $1 - O(\varepsilon)$ factor of OPT.*

Notice that a solution to the rounded instance can be turned into a schedule for the original instance $\mathcal{I}$ by simply removing all jobs of size $(1+\varepsilon)^{\ell}$ from the schedule of N, and then applying a list scheduling algorithm to process the original small jobs. By the same argument as in the proof of Lemma 6, this only costs a factor $1 - O(\varepsilon)$ in the objective function. We can thus restrict to work with instance N. To describe a schedule for N in a compact way, we consider the following definition.

Definition 1. *For a given schedule, a machine* obeys configuration $k : I(\textsc{ub}) \to \mathbb{N}_0$*, if $k(i)$ equals the number of jobs of size $(1+\varepsilon)^i$ assigned to that machine, for all $i \in I(\textsc{ub})$. The* load of configuration k *is defined as* $\text{load}(k) := \sum_{i \in I(\textsc{ub})} k(i)(1+\varepsilon)^i$.

Let us now consider set $K := \{k : I(\textsc{ub}) \to \mathbb{N}_0 : k(i) \leq \sigma\Sigma/\varepsilon + 1 \text{ for all } i \in I\}$. Notice that $|K| \leq (\sigma\Sigma/\varepsilon + 1)^{|I(\textsc{ub})|} \in 2^{O(\frac{1}{\varepsilon}\log^2\frac{1}{\varepsilon})}$. The next lemma assures that these are all necessary configurations that we need to consider.

Lemma 7. *There is an optimal solution to N with all machine configurations in K.*

The proof is given in [14]. Note that the number of jobs per machine in an optimal solution can be upper bounded by $\sigma/\varepsilon + 1$. Thus, the set K contains more configurations than are really needed. Nonetheless, the overestimation of the number of jobs is necessary so that, when a new job arrives and the upper bound increases (by at most a factor of 2), the set K still contains all necessary configurations.

The optimal schedule for N found in Lemma 7 can be described by a vector $(x_k)_{k\in K}$, where x_k denotes the number of machines that obey configuration k. We can see that vector x satisfies the following set of constrains:

$$\sum_{k\in K} x_k = m, \qquad \sum_{k\in K} k(i)\cdot x_k = n_i \quad \text{for all } i \in I(\textsc{ub}) \tag{4}$$

and $x_k \in \mathbb{Z}_{\geq 0}$ for all $k \in K$. We denote by $\mathcal{A} = \mathcal{A}(K, I)$ the matrix defining the set of equations (4); the corresponding right-hand-side is denoted by $b(N, m)$. Also, $D = \left\{x \in \mathbb{Z}^K_{\geq 0} : \mathcal{A}\cdot x = b(N, m)\right\}$ denotes the set of non-negative integral solutions to (4).

4.2 Constructing Stable Solutions

In the following we present the main structural contribution of this paper: We show how to obtain a robust optimal solution to N such that, upon arrival (or departure) of a new job of size at least $\varepsilon \cdot OPT$, we need to migrate jobs of total processing time at most $f(\varepsilon) \cdot OPT$ in order to maintain optimality. This implies that the migration factor needed for this case is upper bounded by $f(\varepsilon)/\varepsilon$.

Let us order and relabel the set of configurations $K = \{k_1, \dots, k_{|K|}\}$ in non-decreasing order of their load, i. e., $\text{load}(k_1) \leq \text{load}(k_2) \leq \dots \leq \text{load}\left(k_{|K|}\right)$.

Definition 2. *Let $x, x' \in D$. We say that x' is lexicographically smaller than x, denoted $x' \prec_{lex} x$, if $x_k = x'_k$ for all $k \in \{k_1, \dots, k_q\}$, and $x'_{k_{q+1}} < x_{k_{q+1}}$, for some q.*

By definition, $\prec_{lex}$ defines a total order on the solution set D. Thus there exists a unique lexicographically minimum vector in $x^* \in D$. We show that x^* has the proper structure needed for our purposes. Note that, in particular, it maximizes the minimum machine load. Moreover, x^* can be computed in polynomial time by solving a sequence of ILPs in constant dimension as follows: for $q = 1, \dots, |K|$, set

$$x^*_{k_q} := \min\left\{x_{k_q} : x \in D \text{ and } x_{k_r} = x^*_{k_r} \text{ for all } r = 1, \dots, q-1\right\} .$$

Alternatively, we can find x^* by solving a single ILP in constant dimension, i. e., by minimizing a carefully chosen linear function over D. Let $\lambda := (m+1)^{-1}$, and define $c_q := \lambda^q$ for $q = 1, \dots, |K|$. The following ILP is denoted by [LEX]:

$$\min\left\{\sum_{q=1}^{|K|} c_q \cdot x_{k_q} : \mathcal{A} \cdot x = b(N, m) \text{ and } x_k \in \mathbb{Z}_{\geq 0} \text{ for all } k \in K\right\} .$$

The proof of the following lemma can be found in [14].

Lemma 8. *Let z be an optimal solution to [LEX]. Then, z is the lexicographically minimal solution in D.*

Let S be the schedule corresponding to z. We next show that S is robust. The new job j^* can be incorporated into the ILP by only slightly changing the right-hand-side of [LEX]. Indeed, as discussed before, we can assume that p_{j^*} is a power of $(1+\varepsilon)$ and is larger than $\varepsilon \cdot OPT \geq \varepsilon \cdot \text{UB}/\sigma$ (by choosing $\sigma \geq \gamma$). Then, we can round the new instance $\mathcal{I}'$ by defining a vector $N' = (n'_i)_{i \in I}$ as follows: for $i = \ell, \dots, u-1$ let

$$n'_i = \begin{cases} n_i + 1 & \text{if } p_{j^*} = (1+\varepsilon)^i, \\ n_i & \text{otherwise,} \end{cases} \quad \text{and} \quad n'_u = \begin{cases} n_u + 1 & \text{if } p_{j^*} \geq (1+\varepsilon)^u, \\ n_u & \text{otherwise.} \end{cases}$$

In other words, if $p_{j^*} > \Sigma\text{UB} \geq 2\text{UB} \geq \text{UB}' \geq OPT'$ then job j^* is processed on a machine of its own. Therefore we can assume that its size is just $(1+\varepsilon)^u$. Also, note that all jobs whose size was rounded down to $(1+\varepsilon)^u$ in the original instance $\mathcal{I}$ are still larger than $\Sigma \cdot \text{UB} \geq \text{UB}'$, and thus get a machine of their own in OPT'. Moreover, jobs that are smaller than $\varepsilon \cdot \text{UB}/\sigma$ are also smaller than $\varepsilon \cdot OPT'$. Thus, using the same argument as in Lemma 6, solving instance N' yields a $(1 - O(\varepsilon))$-approximate solution to $\mathcal{I}'$. Also, analogously to Lemmas 7 and 8, we can solve this instance by optimizing the following modification of [LEX], which we call [LEX]':

$$\min\left\{\sum_{q=1}^{|K|} c_q \cdot x_{k_q} : \mathcal{A}x = b(N', m) \text{ and } x_k \in \mathbb{Z}_{\geq 0} \text{ for all } k \in K\right\}.$$

Let z' be an optimum solution to [LEX]'. Notice that [LEX] and [LEX]' only differ in one entry of the right-hand-side vector by one. Thus, by a result from sensitivity analysis of ILPs, the optimum solutions z and z' are relatively close. This yields the next theorem, whose detailed proof is given in [14].

Theorem 1. *There exists a static robust PTAS if the arriving job is larger than $\varepsilon \cdot OPT$.*

Running time. By the proof of Theorem 1, given z, we can compute z' by exhaustive search through all vectors whose components differ from z by at most $2^{O(\frac{1}{\varepsilon}\log^2\frac{1}{\varepsilon})}$. Thus, the running time needed to compute the desired solution to $\mathcal{I}'$ is $2^{2^{O(\frac{1}{\varepsilon}\log^2\frac{1}{\varepsilon})}}$.

Job departure. The approach presented in this section also works for the job departure case. Indeed, if instance $\mathcal{I}'$ contains one job less than $\mathcal{I}$, we can assume that $\mathcal{I}'$ is nontrivial (otherwise see Lemma 5). Therefore $\textsc{ub}' \geq \textsc{ub}/2$, and thus choosing $\sigma \geq 2\gamma$ implies that $I(\textsc{ub})$ contains all job sizes between $\varepsilon \cdot OPT'$ and OPT'. Thus, rounding instance $\mathcal{I}'$ within this range decreases the optimum value by at most a factor $(1 - O(\varepsilon))$. Again, the right hand sides of [LEX] and [LEX]' differ in only one entry and thus the migration factor needed to construct the solution given by [LEX]' is at most $2^{O(\frac{1}{\varepsilon}\log^2\frac{1}{\varepsilon})}$.

5 Maintaining Robust Solutions Dynamically

In the previous section we showed how to construct a robust $(1-\varepsilon)$-approximate solution, so that we can maintain the approximation guarantee at the arrival (departure) of an arbitrary new big job and keep the migration factor bounded. Nonetheless, we cannot further iterate this method when more jobs arrive or depart since the optimum values of the new instances may become arbitrarily large (small), and thus the range $I(\textsc{ub})$ is not large enough to guarantee the approximation ratios of the rounded instances. On the other hand, we cannot make the index set $I(\textsc{ub})$ larger so as to simultaneously round all possible instances and maintain the number of job sizes constant.

We deal with this difficulty by dynamically adjusting the set $I(\textsc{ub})$. In doing so, we must be extremely careful not to destroy the structure of the constructed solutions and maintain the reassignment cost bounded. Whenever the set $I(\textsc{ub})$ is shifted to the right (left), we must regroup small jobs into larger (smaller) groups. Therefore, $I(\textsc{ub})$ should be shifted only if we can guarantee that there is enough reassignment potential accumulated to regroup all small jobs and simultaneously maintain the structure of optimal solutions. Let $\mathcal{I}$ be the instance after the t-th job arrival/departure. For $t = 1, 2, \ldots,$ we run the following algorithm on the current instance $\mathcal{I}$.

Robust PTAS

1. Run Algorithm Stable-Average on $\mathcal{I}$ to compute A and w.
2. If variable A_0 is undefined or $A \notin [A_0/2, 2A_0]$, then set $\mathcal{I}_0 := \mathcal{I}$ and $A_0 := A$.
3. Set $\text{UB} := 2A_0$ and define sets $I(\text{UB})$, K, and the ILP [LEX] as in Section 4. Compute the optimum solution z to [LEX] and the corresponding schedule S.

Notice that throughout the algorithm, $OPT \leq A \leq 2A_0 = \text{UB}$, and thus UB is indeed an upper bound on OPT. Moreover, $OPT \geq A/2 \geq A_0/4 = \text{UB}/8$, and thus UB is within a factor 8 of OPT (i. e., $\gamma = 8$ with the notation of Section 4). By the discussion in Section 4.2, an appropriate choice of values σ and Σ guarantees that all constructed solutions are $(1 - O(\varepsilon))$-approximate. It remains to show that the needed reassignment factor is bounded by a constant.

For the analysis we partition the iterations of the algorithm into blocks. Each block B consists of a consecutive sequence of iterations where the value of A_0 is kept constant. Thus, for each instance $\mathcal{I}$ occurring in B, its stable average A belongs to the interval $[A_0/2, 2A_0]$. Consider two consecutive instances $\mathcal{I}$ and $\mathcal{I}'$ that belong to the same block. We add a symbol *prime* to denote the variables corresponding to $\mathcal{I}'$ (e. g., A' is the stable average of $\mathcal{I}'$). Since $\mathcal{I}$ and $\mathcal{I}'$ belong to the same block B, it holds that $\text{UB} = 2A_0 = \text{UB}'$, and thus the sets $I(\text{UB})$ and K coincide with $I(\text{UB}')$ and K', respectively. Therefore, as already discussed in Section 4.2, the ILPs [LEX] and [LEX]' only differ in one entry of their right-hand-side vectors and the same reasoning as in Theorem 1 yields the following lemma.

Lemma 9. *If two consecutive instances $\mathcal{I}$ and $\mathcal{I}'$ belong to the same block of iterations, then the migration factor used to obtained S' from S is at most $2^{O(\frac{1}{\varepsilon}\log^2\frac{1}{\varepsilon})}$.*

It remains to consider the case that instance $\mathcal{I}$ and the next instance $\mathcal{I}'$ belong to different blocks. For this, we assume that $\mathcal{I}'$ contains one more job than $\mathcal{I}$ and that $A' > 2A_0$. We can deal with the case $A' < A_0/2$ in an analogous way. By Lemma 5 we can assume that $\mathcal{I}$ is nontrivial. Assume that $\mathcal{I}$ belongs to block B, and consider the value of A_0 corresponding to this block. It holds that $A_0 \leq A \leq 2A_0$. Also, since $\mathcal{I}$ is nontrivial, Corollary 1 ensures that $A \leq A' \leq 2A$, and therefore $\text{UB} \leq \text{UB}' \leq 4\text{UB}$.

In order to compare solutions $z \in \mathbb{N}_0^K$ and $z' \in \mathbb{N}_0^{K'}$, we need to interpret them in a common euclidean space containing them. Notice that huge jobs of $\mathcal{I}$ have processing time larger than $\Sigma \cdot \text{UB} \geq \text{UB}'$ (assuming $\Sigma \geq 4$). These jobs get a machine of their own in solutions OPT, OPT', S, and S'; thus, we do not need to consider them. We can therefore assume that all jobs of $\mathcal{I}$ and $\mathcal{I}'$ have processing time at most $\Sigma \cdot \text{UB}$. In particular, the entries of vector $N' = (n'_i)_{i \in I(\text{UB}')}$ are zero if $(1+\varepsilon)^i > \Sigma \cdot \text{UB}$. We can thus interpret N' as a vector in $\mathbb{N}_0^{I(\text{UB})}$ by setting to zero the entries n'_i with $(1+\varepsilon)^i < \varepsilon \cdot \text{UB}'/\sigma$. With this simplification, all feasible solutions to [LEX]' corresponds to solutions to [LEX] where the right hand side has been modified according to N'. Thus, z' can be regarded as an optimal solution to this modified version of [LEX].

We bound the difference between N and N', which allows us to bound the difference between z and z'. This will imply the result on the reassignment factor.

Lemma 10. *Let q be the number of jobs that have arrived in block B, including the job that made the algorithm change to the next block. Then, $\|N - N'\|_1 \in O(q/\varepsilon)$.*

The proof of the lemma can be found in [14]. Applying Lemma 10 and the same proof technique as in Theorem 1, we obtain the following lemma.

Lemma 11. *The reassignment potential used to construct S' is at most $q \cdot A_0 \cdot 2^{O(\frac{1}{\varepsilon}\log^2\frac{1}{\varepsilon})}$.*

Theorem 2. *For the machine covering problem with jobs arriving and departing online, there exists a $(1-\varepsilon)$-competitive polynomial algorithm with constant reassignment factor at most $2^{O(\frac{1}{\varepsilon}\log^2\frac{1}{\varepsilon})}$.*

References

1. Albers, S.: Online algorithms: a survey. Mathematical Programming 97, 3–26 (2003)
2. Alon, N., Azar, Y., Woeginger, G.J., Yadid, T.: Approximation schemes for scheduling on parallel machines. Journal of Scheduling 1, 55–66 (1998)
3. Andrews, M., Goemans, M., Zhang, L.: Improved bounds for on-line load balancing. Algorithmica 23, 278–301 (1999)
4. Azar, Y.: On-line load balancing. In: Fiat, A., Woeginger, G.J. (eds.) Dagstuhl Seminar 1996. LNCS, vol. 1442, pp. 178–195. Springer, Heidelberg (1998)
5. Azar, Y., Epstein, L.: On-line machine covering. Journal of Scheduling 1, 67–77 (1998)
6. Chen, B., van Vliet, A., Woeginger, G.J.: Lower bounds for randomized online scheduling. Information Processing Letters 51, 219–222 (1994)
7. Epstein, L., Levin, A.: A robust APTAS for the classical bin packing problem. Mathematical Programming 119, 33–49 (2009)
8. Fleischer, R., Wahl, M.: Online scheduling revisited. Journal of Scheduling 3, 343–353 (2000)
9. Hochbaum, D.S., Shmoys, D.B.: Using dual approximation algorithms for scheduling problems theoretical and practical results. Journal of the ACM 34, 144–162 (1987)
10. Rudin III, J.F., Chandrasekaran, R.: Improved bounds for the online scheduling problem. SIAM Journal on Computing 32, 717–735 (2003)
11. Sanders, P., Sivadasan, N., Skutella, M.: Online scheduling with bounded migration. Mathematics of Operations Research 34, 481–498 (2009)
12. Sgall, J.: A lower bound for randomized on-line multiprocessor scheduling. Information Processing Letters 63, 51–55 (1997)
13. Sgall, J.: On-line scheduling — a survey. In: Fiat, A., Woeginger, G.J. (eds.) Dagstuhl Seminar 1996. LNCS, vol. 1442, pp. 196–231. Springer, Heidelberg (1998)
14. Skutella, M., Verschae, J.: A robust PTAS for machine covering and packing. Technical Report 011-2010, Technische Universität Berlin (2010), `http://www.math.tu-berlin.de/coga/publications/techreports/2010/Report-011-2010.xhtml`
15. Westbrook, J.: Load balancing for response time. Journal of Algorithms 35, 1–16 (2000)
16. Woeginger, G.J.: A polynomial-time approximation scheme for maximizing the minimum machine completion time. Operations Research Letters 20, 149–154 (1997)