Algorithmica (2023) 85:2238–2259
https://doi.org/10.1007/s00453-023-01097-0

Online Minimization of the Maximum Starting Time: Migration Helps

Asaf Levin[1]

Received: 16 September 2020 / Accepted: 14 January 2023 / Published online: 26 January 2023

Abstract
We consider non-preemptive load balancing on m identical machines where the cost of a machine is defined as the maximum starting time of a job assigned to the machine, and the goal is to find a partition of the jobs that minimizes the maximum machine cost. In our variant the last job on each machine is the smallest job assigned to that machine. The online model for this problem is too restrictive as a trivial example shows that there is no competitive algorithm for the problem. We show that a constant migration factor is sufficient to guarantee a $(\frac{3}{2}+\varepsilon)$-competitive algorithm for all $\varepsilon > 0$, and using a constant migration factor cannot lead to a better than a $\frac{3}{2}$-competitive algorithm. We also show that for this problem, constant amortized migration factor is strictly more powerful and allows us to obtain a polynomial time approximation scheme with a constant amortized migration factor. Thus, the ability to move some limited set of jobs on each step allows the algorithm to be much better than in the pure online settings.

Keywords Online algorithms · Scheduling · Migration factor

1 Introduction

In this work we consider a load balancing problem on identical machines. The input consists of n jobs denoted as $1, 2, \ldots, n$ where job j has a processing time $p_j \geq 0$, and m identical machines. We sometimes refer to the processing time of a job as the *size* of the job. A feasible solution is a partition of the job set into m machines, which means that we consider a non-preemptive scheduling problem. Here, for each machine we are concerned with the point in time in which the last job assigned to that machine is starting. In our settings PR- MIN the adversary chooses this permutation, so it will schedule the *smallest* job as the last job. Thus, if the algorithm assigns the set of jobs S to machine i, then the cost of i in PR- MIN will be $\sum_{j\in S} p_j - \min_{j\in S} p_j$ if S is not

✉ Asaf Levin
levinas@ie.technion.ac.il

1 Faculty of Industrial Engineering and Management, The Technion, 32000 Haifa, Israel

empty and zero otherwise (i.e., if S is empty). The goal is to find a feasible solution (that is a partition of the jobs into m machines) so that the maximum cost of a machine is minimized. Given a solution to the problem, the *load* of machine i is the total size of jobs assigned to i. Note that the load of a machine is not smaller than the cost of the machine.

In order to motivate our scheduling problem consider the following scenario. Assume that we are given a computer center with m identical computers, and there is a sequence of jobs arriving to be processed. Once a job arrives the system administrator needs to allocate the job to a machine. However, in order to start processing a job the administrator needs to do additional operation like pressing a key. Therefore, the administrator cannot go home at the end of the working day before the last job has started. The goal of the administrator is to schedule all jobs in a way that minimizes the time in which he could go home. However, the head of the computer is trying to use the administrator for other tasks as well and thus he is reshuffling the permutation of jobs assigned to each machine so as the last processed job is a smallest one (among jobs assigned to that machine).

Given an algorithm ALG for a minimization problem, we say that it has a competitive ratio of at most ρ if it always returns a feasible solution of cost at most ρ times the offline optimal cost on that instance. In this case, we also say that ALG is ρ-competitive. If the competitive ratio of ALG is a constant then we say that ALG is competitive.

Here, we consider an online environment where the jobs are revealed to the algorithm one by one, and the algorithm needs to maintain a partition of the jobs into m identical machines. In these online scenarios, the online algorithm is committed to the partition of the jobs into the m machines, while the internal permutation of jobs assigned to a machine may change once further jobs are assigned to that machine and in fact it is always the worst permutation both for the algorithm as well as for the optimal offline solution.

We motivate our research by observing that in the (pure) online scenario, there is no competitive algorithm. We show this claim by proving that for every constant $M > 1$, there is no deterministic algorithm with competitive ratio M. In order to establish this claim we consider the following instance. The input sequence starts with m identical jobs of size 1. At this point an optimal offline solution places one such job on each machine, and has zero cost. Thus, any competitive algorithm must place one such job on each machine (any other solution has a cost of at least 1 leading to an unbounded competitive ratio). After these m jobs are scheduled the input continues with one additional job of size $M+1$ and then the input ends. At this point the optimal offline cost is (at most) 1: a feasible solution of this cost is obtained by placing a pair of unit sized jobs on one machine, and exactly one job on each other machine (there are $m-1$ additional jobs). However, the algorithm places this job of size $M+1$ on a machine that previously had a job of size 1. The cost of that machine is $M+1$, and the claim that the algorithm has a competitive ratio of at least $M+1$ follows.

Since for the pure online scenario there is no competitive algorithm, we would like to follow the approach of allowing the algorithm to perform some limited changes to the solution whenever a new job is revealed to the algorithm. Here we study two such models.

The first model is the class of algorithms with a constant migration factor that we define next. An online algorithm for some scheduling problem on identical machines has a migration factor f if whenever a new job j is revealed to the algorithm with its size p_j it may migrate a subset of the job set of total size not larger than $f \cdot p_j$. We say that an algorithm has a constant migration factor if there is a constant number f for which the algorithm has a migration factor f.

The second model was suggested in the literature for cases in which there is a non-trivial lower bound on the competitive ratios of algorithms with constant migration factors, where a non-trivial lower bound means a lower bound that is strictly larger than 1. In this second model we define the notion of amortized migration factor. Here, an algorithm with an amortized migration factor f, considers the input jobs one by one, and may change the schedule of previously assigned jobs in later steps of the algorithm (when new jobs are revealed). However, such an algorithm needs to satisfy the following accumulating constraints. For every prefix of the input job sequence of total job size of P, the algorithm can migrate subsets of jobs (throughout this prefix of the input) of total size at most $f \cdot P$ where if a job is migrated several times during this prefix its size is contributed several times to the total size of migrated subsets of jobs. Similarly, we say that an algorithm has a constant amortized migration factor if there is a constant f for which the algorithm has an amortized migration factor f. Note that by definition if an algorithm has a constant migration factor then it also has a constant amortized migration factor (but the other direction need not hold). Furthermore, the study of algorithms with constant amortized migration factors is meaningful only if we are able to obtain smaller competitive ratios using such algorithms with respect to the competitive ratios we are able to get using algorithms with constant migration factors and similarly, this is a meaningful result only if these are smaller than the competitive ratios for the pure online case.

Literature Review. In another variant of the problem the machine (or the algorithm) is choosing the permutation of jobs on each machine so it will schedule the *largest* job as the last job while in a third variant the permutation of jobs assigned to a machine is according to the index of the jobs. The version where the largest job is the last job was studied in the online environment recently in [10] where it is shown that the pure online problem admits a 3-competitive algorithm whereas no online algorithm has a competitive ratio smaller than 2.26953, and furthermore allowing a constant migration factor is sufficient to obtain an approximation scheme. The last version where jobs have to be scheduled sorted by indexes and cannot be reordered (this is the order of arrival for the online case) was studied both as an online problem and as an offline problem [11, 12]. Other scheduling and bin packing problems where the relevant time is the starting time of the last job or item were studied as well [5, 19, 20, 25] where for bin packing problems of this type the term *open-end bin packing* is frequently used. Note that our problem PR- MIN is significantly harder than the two other online scheduling variants of this type. First, for the pure online case, we showed above that PR- MIN does not admit a competitive algorithm whereas [12] showed that the version where jobs are sorted by their indexes has a 12-competitive algorithm, and [10] showed a 3-competitive algorithm for the version where the last job is the largest job. Second, with respect to algorithms with constant migration factor, we will show the non-existence of an algorithm with a competitive ratio smaller than $\frac{3}{2}$ for Pr- min whereas [10] showed the existence of an approximation scheme with constant migration factor for the version where the last job scheduled on each machine is its largest job.

Next, we discuss the earlier studies of other scheduling problems with respect to algorithms with constant migration factor and with respect to algorithms with constant amortized migration factor (see also [24]). In [21], the model of constant migration factor was introduced and analyzed with respect to the makespan minimization on identical machines. They showed that for this makespan objective one is able to get an approximation scheme with constant migration factor. Later [23] established that not every problem that has a standard PTAS still has one in the online scenario with constant migration factors. They showed this result for the Santa-Claus objective on identical machines, and used this result as a motivation for studying algorithms with constant amortized migration factors. Among the last family of algorithms, the ones with constant amortized migration factors, they establish the existence of an approximation scheme for the Santa-Claus objective and other load balancing objectives (not including our problem). See also [14] for additional results on the possible competitive ratio for the Santa-Claus objective when one restrict himself for algorithms with constant migration factor. Namely it is shown that an algorithm that is based on the Longest Processing Time (LPT) rule has a competitive ratio of $\frac{4}{3} + \varepsilon$ and migration factor of $O(\frac{1}{\varepsilon^3})$ for every $\varepsilon > 0$. In [8], the makespan objective on identical machines in the preemptive case was analyzed, and it is shown that one can maintain an optimal schedule with a migration factor smaller than 1. Such maintenance of optimal solutions with constant migration factors is unique, and was shown to be impossible for the non-preemptive case by [21]. The preemptive case on uniformly related machines cannot have a similar result as [8] exhibits a non-constant lower bound on the migration factor of optimal solutions for this uniformly related machines case. In [9] the total weighted completion time objective was analyzed and it is shown that if every machine processes its assigned jobs according to Smith's rule, then there is an approximation scheme with constant migration factor, whereas if the algorithm is committed to the time slots of every assigned job, then such result cannot be obtained.

Similar studies regarding algorithms with constant migration factors and constant amortized migration factors were carried out for the bin packing problem [3, 6, 13, 15], cube packing [7], strip packing [16], and bin covering [2]. See also [1] for a study of algorithms with constant amortized migration factors for such packing problems.

Paper Outline. In Sect. 2 we prove that there is no algorithm whose competitive ratio is strictly smaller than $\frac{3}{2}$ that has a constant migration factor. This lower bound for algorithms with constant migration factor is in fact tight, as we show in Sect. 3 that for every $\varepsilon > 0$ there is an algorithm with a competitive ratio of $\frac{3}{2} + \varepsilon$ and a constant migration factor (this migration factor is some function of ε). In order to break the lower bound of $\frac{3}{2}$ we consider in Sect. 4 the more powerful set of algorithms, namely the ones with constant amortized migration factors and show that in this family there is an approximation scheme with a constant amortized migration factor.

2 Lower Bound on the Competitive Ratio of Algorithms with a Constant Migration Factor

In this section we prove the following lower bound which applies for every number of identical machines (that is at least two).

Theorem 2.1 *The online problem* PR- MIN *on m identical machines where $m \geq 2$ does not have an algorithm whose competitive ratio ρ is strictly smaller than $\frac{3}{2}$ and its migration factor is bounded.*

Proof Assume by contradiction that the competitive ratio of ALG is $\rho < 3/2$ and the migration factor of ALG is some constant M. Next, let $\delta > 0$ be such that $\frac{1}{\delta} > M$ is an even integer number and $\rho < \frac{3}{2} - \delta$. Our proof distinguishes between the case that m is an even number and the case where m is an odd number.

First consider the case where m is an even number. The input starts with m jobs of size 1. At this stage, the optimal offline cost is 0 as the solution that schedules one job on each machine has a zero cost. Thus, the algorithm is forced to schedule these jobs exactly like that, namely one job on each machine. In the next stage the input continues with $\frac{m}{2\delta}$ jobs each of which of size δ. The algorithm cannot migrate any of the unit-sized jobs as otherwise it will violate the migration factor, and by the pigeonhole principle it must place at least $\frac{1}{2\delta}$ jobs on a common machine. In this machine the resulting maximum starting time is at least $\frac{3}{2} - \delta$. However, an offline solution may place a pair of unit sized jobs on each of the first $\frac{m}{2}$ machines, and on each other machine it schedules $\frac{1}{\delta}$ jobs each of which of size δ. The resulting cost of this offline solution is 1, and we get a contradiction to the claimed competitive ratio of ALG.

Next, consider the case where m is an odd number. The input starts with one job of size 2 and $m - 1$ jobs of size 1. At this stage, the optimal offline cost is 0 as the solution that schedules one job on each machine has a zero cost. Thus, the algorithm is forced to schedule these jobs like that, namely one job on each machine. Assume without loss of generality that the algorithm schedules the job of size 2 on machine m. In the next stage the input continues with $\frac{m-1}{2\delta}$ jobs each of which of size δ. The algorithm cannot migrate any of the unit-sized jobs or the job of size 2 as otherwise it will violate the migration factor. We claim that the cost of the solution returned by the algorithm is at least $\frac{3}{2} - \delta$. First, if the algorithm places a job of size δ on machine m, then the cost of that machine is at least 2 and the claim follows. Otherwise, the set of jobs of size δ is assigned to the first $m - 1$ machines, and by the pigeonhole principle ALG must place at least $\frac{1}{2\delta}$ jobs on one of those machines. In this machine the resulting maximum starting time is at least $\frac{3}{2} - \delta$. However, an offline solution may place a pair of unit sized jobs on each of the first $\frac{m-1}{2}$ machines, another machine with one job of size 2 (and no other jobs) and on each other machine it schedules $\frac{1}{\delta}$ jobs each of which of size δ. The resulting cost of this offline solution is 1, and we get a contradiction to the claimed competitive ratio of ALG. □

3 Our Algorithm with Constant Migration Factor

Our aim is to present for every $\varepsilon > 0$ an algorithm whose competitive ratio is $\frac{3}{2} + \varepsilon$ and its migration factor is bounded by some function of ε. Thus, the first step of our algorithm, namely to round up the size of every (arriving) job to the next integer power of $1+\varepsilon$ could be carried out without loss of generality (at the end we will need to scale ε by some constant factor). This is so, as the cost of every solution cannot decrease by this rounding and the cost of an optimal offline solution (for the rounded instance) is at most $1+\varepsilon$ times its cost on the original instance, the migration factor of the resulting algorithm in terms of the original instance is at most $1+\varepsilon$ times the migration factor on the rounded instance. Thus, without loss of generality we could assume the following.

Assumption 3.1 Without loss of generality, the input to the problem satisfies that the size of every job is an integer power of $1+\varepsilon$. Assuming this assumption increases the competitive ratio as well as the migration factor by a multiplicative factor of at most $1+\varepsilon$.

We let OPT be a valid upper bound on the cost of an optimal offline solution that can only increase when new jobs arrive (observe that the optimal offline cost cannot decrease when new jobs are added to the instance). We maintain this value as new jobs are revealed by the adversary as we discuss below. Consider a given point in time where some jobs were released. We partition the current job set into the following partitions (named classes): let $\mathcal{H} = \{j : p_j > \text{OPT}\}$, $\mathcal{L} = \{j : \text{OPT} \geq p_j > \frac{3}{4} \cdot \text{OPT}\}$, $\mathcal{M} = \{j : \frac{3}{4} \cdot \text{OPT} \geq p_j > \varepsilon \cdot \text{OPT}\}$, and $\mathcal{T} = \{j : p_j \leq \varepsilon\text{OPT}\}$. With respect to this partition, we say that a job j is *huge* if $j \in \mathcal{H}$, it is *large* if $j \in \mathcal{L}$, it is *medium* if $j \in \mathcal{M}$, and *tiny* if $j \in \mathcal{T}$. Observe that when new jobs arrive, the value of OPT may increase and thus jobs may move from one class to another. However, since the maintenance of this partition is carried out in an auxiliary data structure we are able to maintain it whenever a new job arrives without migrating jobs.

The following lemma follows by observing the cost of solutions to PR- MIN.

Lemma 3.2 *Consider a feasible solution of cost at most* OPT. *Let S be the set of jobs assigned by this solution to a common machine i. Then the following holds:*

1. *If $S \cap \mathcal{H} \neq \emptyset$, then S has a unique job that is a huge job (and no other jobs).*
2. *If $|S| \geq 2$, then either both $|S| = 2$ and $|S \cap \mathcal{L}| \geq 1$, or the load of S is at most $\frac{3}{2} \cdot$ OPT.*

Proof If S has a huge job then if S has another job then the maximum starting time of a job on machine i is at least as large as the size of a huge job assigned to i and this is more than OPT, contradicting the cost of this feasible offline solution. Thus, the first claim holds.

To prove the second claim, first note that if $|S| = 2$, then either the load of S is at most $\frac{3}{2}$OPT or at least one of its jobs has size larger than $\frac{3}{4}$OPT (and thus by the first claim it is large). Thus, to prove the claim assume that $|S| \geq 3$. Since the maximum starting time of machine i is at most OPT, adding to it the size of the smallest job assigned to i adds at most $\frac{\text{OPT}}{2}$ time units and so the total size of jobs in S is at most $\frac{3}{2} \cdot$ OPT. □

Given the value of OPT, the partition of the jobs into huge, large, medium, and tiny is fixed. We will keep the schedule in a way that maintains the following invariants:

Invariants 3.3 *The schedule of the jobs in $\mathcal{H} \cup \mathcal{L} \cup \mathcal{M} \cup \mathcal{T}$ satisfies the following invariants:*

1. *Each job in $\mathcal{H}$ is scheduled on a dedicated machine without other jobs.*
2. *There is at most one machine containing both one large job and a non-empty set of medium jobs. If such machine exists, then the large job scheduled on that machine is a smallest large job (in terms of the rounded size of the jobs) and we call this machine containing both large and medium jobs the* mixed machine. *This mixed machine has either exactly two jobs where one of those is a smallest large job and the other is a largest medium job, or its load is at most $(\frac{3}{2} + 3\varepsilon) \cdot \text{OPT}$. Furthermore, if a machine has exactly one large job and a set of tiny jobs, then its load is at most $(\frac{3}{2} + 3\varepsilon) \cdot \text{OPT}$.*
3. *The other large jobs are scheduled in the following way where each machine is assigned at most two large jobs. For a machine containing two large jobs, we require that the sizes of these jobs are either equal or consecutive in the sorted list of sizes of large jobs in the instance. For a pair of consecutive sizes s_1, s_2 of large jobs, we require that there will be at most one machine with jobs of sizes s_1, s_2 (one of each).*
4. *Each machine that contains only medium and tiny jobs has load of at most $(\frac{3}{2} + 3\varepsilon) \cdot \text{OPT}$.*

We observe that if we can maintain the Invariants 3.3, then the competitive ratio of $\frac{3}{2} + 3\varepsilon$ is guaranteed (if in every step OPT denotes the optimal cost of the instance after adding the last arriving job of this step). This holds as machines with one huge job or two large jobs (or one large job and one medium job) have maximum starting time of at most OPT while for every other machine the maximum starting time is at most its load which is at most $(\frac{3}{2} + 3\varepsilon) \cdot \text{OPT}$. Thus, we established the following.

Lemma 3.4 *An algorithm that maintains the Invariants 3.3 for* OPT *being the cost of an optimal offline solution of the prefix of jobs considered so far has competitive ratio of at most $\frac{3}{2} + 3\varepsilon$.*

In order to describe our algorithm we note that in some iterations the values of OPT is increased while in other iterations it stays unchanged. In order to simplify the presentation, we consider iterations in which OPT is increased as a pair of iterations, in the first iteration of such pair OPT increases but there is no new job that we need to schedule, and in the second iteration of this pair the new job is scheduled (perhaps migrating other jobs). Then, there are two types of iterations. In the first type of iterations OPT is increased, no job is released and we are not allowed to migrate jobs. In the second type of iterations, a new job of index j is released and the value of OPT remains the same (so it is still a valid upper bound on the cost of an optimal offline solution even after the new job is released). In this second type of iterations we need to schedule the new job so that the invariants are maintained and to migrate jobs of small total size so that the migration factor will be maintained.

The events where OPT increases are easily handled as follows (the arguments here are the motivation for allowing large jobs to be scheduled on machines without other large jobs).

Lemma 3.5 *If a solution* SOL *maintains the invariants with a smaller value of* OPT *(denoted as* OPT'*) then it also satisfies the invariants with a larger value of* OPT *denoted as* OPT''.

Proof Let $\mathcal{H}', \mathcal{L}', \mathcal{M}'$ be the three classes of huge, large, and medium jobs prior to the increase of the value of OPT, respectively. Similarly, let $\mathcal{H}'', \mathcal{L}'', \mathcal{M}''$ be the three classes of huge, large, and medium jobs just after the increase of the value of OPT, respectively.

First, observe that $\mathcal{H}'' \subseteq \mathcal{H}'$ and since SOL used to satisfy invariant 1 prior to the increase of OPT, we conclude that each job in $\mathcal{H}''$ is scheduled on a dedicated machine without other jobs so invariant 1 continues to hold after the increase of OPT.

Every machine has the same total size of jobs as it used to have, and the machines containing medium and tiny jobs after the increase of OPT may have either at most two jobs (so the bound clearly holds by the upper bound on the size of medium jobs) or used to have a job set of total size at most $(\frac{3}{2} + 3\varepsilon) \cdot \text{OPT}' \leq (\frac{3}{2} + 3\varepsilon) \cdot \text{OPT}''$ so invariant 4 is satisfied.

Consider invariants 2 and 3. If there is no job in $\mathcal{L}' \setminus \mathcal{L}''$, then the claim follows since SOL used to satisfy these invariants and no machine becomes a mixed machine so the claim about the load of that machine (if it has at least three jobs) continues to hold by $\text{OPT}' \leq \text{OPT}''$. Otherwise some large jobs become either medium or tiny. The set of jobs that stop being large contains the job in $\mathcal{L}'$ of the earlier mixed machine (if such machine used to exist), so that machine stops being a mixed machine. Since SOL used to satisfy invariants 1 and 3, it continues to satisfy invariants 2 and 3. □

We will use the following properties of a solution of cost at most OPT.

Lemma 3.6 *Let* ALG *denote the solution constructed by the algorithm that satisfies the invariants. Assume that there is (an offline) solution of cost at most* OPT, *then there is a solution* SOL *of cost at most* OPT *that satisfies the following properties.*

1. SOL *has at most one machine containing exactly one large job (and perhaps some tiny or medium jobs) and all other large jobs are scheduled in pairs on machines each of which is assigned two large jobs. If such machine with exactly one large job exists, it is called a* mixed machine in SOL.
2. *Every pair of large jobs that* ALG *places on a common machine, are also scheduled on a common machine in* SOL. *Furthermore, if* SOL *has a mixed machine then the large job of this mixed machine is not scheduled in one of the pairs of large jobs in* ALG.

Proof We first show that without loss of generality the first property holds. Assume by contradiction that there is no such solution SOL of cost smaller than OPT with at most one such machine (containing exactly one large job). Let SOL be a feasible solution for PR- MIN with an objective value of at most OPT that minimizes the number of such machines. Since SOL does not satisfy the requirement, there are at least two such machines. We pick a pair of machines i, i' each of which has exactly one large job ℓ, ℓ', respectively, and some set of non-large jobs. We let $\sigma_i, \sigma_{i'}$ denote these two sets of other jobs scheduled by SOL on i, i', respectively. We replace the schedule of the jobs on those two machines. Without loss of generality, we assume that among the set of jobs in $\sigma_i \cup \sigma_{i'}$, if it is not empty, then there is a minimum sized job in $\sigma_{i'}$. On machine i, we schedule the two large jobs ℓ, ℓ', and note that since these two jobs are large, the cost of i would be at most OPT. On machine i' we schedule the jobs in $\sigma_i \cup \sigma_{i'}$. Note that if σ_i consists of a single job, then since it is not large its size is smaller than ℓ', so adding this set of jobs to $\sigma_{i'}$ would not increase the cost of i' with respect to its cost in SOL. Otherwise, if σ_i has more than one job, then by Lemma 3.2, their total size is smaller than $\frac{3}{4} \cdot$ OPT. Thus, once again adding this set of jobs to $\sigma_{i'}$ would not increase the cost of i' with respect to its cost in SOL. Last, if $\sigma_i = \emptyset$, then the new cost of i' is smaller than its cost in SOL. Thus, we get a contradiction to our choice of SOL among all solutions with objective function value of at most OPT.

Next, we claim that without loss of generality, we can assume that every pair of large jobs that ALG schedules on a common machine, are also scheduled on a common machine in SOL, and if there is a mixed machine in SOL (by the previous claim this happens if and only if $|\mathcal{L}|$ is an odd number), then the large job of this mixed machine is not scheduled in one of the pairs of large jobs in ALG. This is indeed without loss of generality, as any pair of large jobs could be scheduled on a common machine with a resulting cost of at most OPT and replacing the large job of the mixed machine of SOL with the smallest such job does not increase the cost of that mixed machine. Thus, we can reorder the pairs of jobs in SOL to fit the last assumption and this reordering does not change the number of machines with pairs of large jobs so the first property continues to hold. □

Next, we consider an iteration in which job j is revealed to the algorithm and we need to schedule it, but the value of OPT remains the same. Here, we first consider the case in which j is tiny, and later the other cases where j is non-tiny. For a tiny job j, we will schedule the job without migrating any job. Specifically, among the machines not containing a huge job, we pick one where the load prior to the current step is at most $(\frac{3}{2} + 2\varepsilon) \cdot$ OPT and schedule j there. Observe that this step clearly maintains the invariants because the size of j is at most $\varepsilon \cdot$ OPT. Thus, we only need to prove that whenever the new job is tiny, we can indeed find such machine to schedule j there.

Lemma 3.7 *If j is tiny, then prior to the scheduling of j, there is a machine without huge jobs whose load is at most $(\frac{3}{2} + 2\varepsilon) \cdot$ OPT.*

Proof Let ALG be the solution of the algorithm just before j is revealed. Assume by contradiction that such machine does not exist. Observe that for every huge job j' we have that both the algorithm and every optimal offline solution to the instance of PR- MIN place j' on a dedicated machine. Thus, in order to prove the claim we can delete this huge job j' as well as one machine. We are left with an instance of $m - |\mathcal{H}|$ machines and without huge jobs, and we need to show that if there is a solution SOL to the instance with j with an objective value of at most OPT then there is a machine whose load is at most $(\frac{3}{2} + 2\varepsilon) \cdot$ OPT prior to scheduling of j.

Thus, by Lemma 3.2, we conclude that on every machine of SOL that does not contain large jobs, the load is at most $\frac{3}{2} \cdot$ OPT if it does not contain tiny jobs, and it is at most $1+\varepsilon$ if it has a tiny job. Next, we allow SOL to fractionally schedule the medium and tiny jobs subject to these constraints (where the schedule of the large jobs is kept without modifications). The (relaxed) constraints for the fractional solution are as follows. If a machine has a fraction of a tiny job then its load is at most $(1+\varepsilon)\cdot \text{OPT}$, and otherwise if it has at most one large job it has a load of at most $(\frac{3}{2}+\varepsilon)\cdot \text{OPT}$, we allow the mixed machine to contain exactly two jobs even in cases where their total size is above $(\frac{3}{2}+\varepsilon)\cdot \text{OPT}$. The solution SOL is still a feasible solution. In order to prove the lemma, it suffices to show that we can transform this (fractional) solution into a different fractional solution where the large jobs are scheduled as in ALG while the load of every machine with at most one large job is at most $(\frac{3}{2}+2\varepsilon)\cdot \text{OPT}$ (except perhaps the mixed machine whose content is not changed). If we will prove the last claim, then by the pigeonhole principle it will show that ALG has at least one machine whose load is at most $(\frac{3}{2}+2\varepsilon)\cdot \text{OPT}$.

Note that by Lemma 3.6, if the number of machines with pairs of large jobs in SOL is the same as their number in ALG, then the claim follows. Since SOL has a maximum number of pairs of large jobs scheduled on common machines, we conclude that this number in SOL is not smaller than its number in ALG.

Next, we delete from the instance as well as from ALG and SOL every pair of large jobs that are scheduled on a common machine (together with one machine for each such removed pair). By Lemma 3.6, we conclude that in ALG every remaining large job is scheduled on a separate machine (together with some non-large jobs) while in SOL there are still pairs of large jobs on common machines. Let x be the number of remaining large jobs in the instance. Then, there are at least $x-1$ machines with one large job and no medium jobs in ALG. Since the load of every such machine is larger than $(\frac{3}{2}+2\varepsilon)\cdot \text{OPT}$ (otherwise the lemma holds), we conclude that the total size of tiny jobs scheduled on such machine is at least $(\frac{1}{2}+2\varepsilon)\cdot \text{OPT}$.

Next, we modify SOL for the fractional problem by sorting the fractions of medium and tiny jobs so that the mixed machine of SOL has medium jobs (if there is one) and there is at most one machine containing both fractions of medium jobs and fractions of tiny jobs. This feasibility of the fractional solution obtained as a result of this rearrangement of SOL follows by a trivial exchange argument. Thus, there are more than

$$\frac{(x-1)\cdot(\frac{1}{2}+2\varepsilon)\cdot \text{OPT}}{(1+\varepsilon)\cdot \text{OPT}} > \frac{x-1}{2}$$

machines in (the modified) SOL with load at most $1+\varepsilon$ containing fractions of tiny jobs, each of which has no large jobs except possibly for the mixed machine.

First, consider the case where SOL does not have a mixed machine, that is, we assume that x is an even number. We take the large jobs that SOL schedules on machines and we reschedule them by having each large job together with fractions of tiny and medium jobs (that were scheduled on machines with load at most $(1+\varepsilon)\cdot \text{OPT}$) of total size of $\frac{1+\varepsilon}{2}\cdot \text{OPT}$ to create a schedule of a machine with load at most $(\frac{3}{2}+\varepsilon)\cdot \text{OPT}$ that replaces the $\lceil\frac{x-1}{2}\rceil=\frac{x}{2}$ machines without large jobs and load of at most $1+\varepsilon$ as well as the $\frac{x}{2}$ machines with pairs of large jobs. Other machines are not modified and we know that the load of each such machine is at most $(\frac{3}{2}+\varepsilon)\cdot \text{OPT}$.

Last, consider the case where x is an odd number. Thus, $\frac{x-1}{2}$ is the number of pairs of large jobs scheduled on a common machine. However, we have at least $\frac{x-1}{2}+1$ machines in SOL with load of at most $(1+\varepsilon)\cdot\text{OPT}$. If the mixed machine has load larger than $\frac{3}{2}\cdot\text{OPT}$, then we take one such machine with load of at most $(1+\varepsilon)\cdot\text{OPT}$ and we move fractions of medium jobs from the mixed machine to this selected machine so that the resulting load on both these machines will be at most $\frac{3}{2}\cdot\text{OPT}$ (we move a fraction of size $\frac{1}{4}\cdot\text{OPT}$ so the selected machine will have a new load not larger than $(\frac{5}{4}+\varepsilon)\cdot\text{OPT} < \frac{3}{2}\cdot\text{OPT}$). If the mixed machine has load of at most $1+\varepsilon$ and it is counted in these $\frac{x-1}{2}+1$ machines we do not change its solution and not consider it in the next step. We are left with other $\frac{x-1}{2}$ machines of load at most $1+\varepsilon$ (which are not the mixed machine nor the selected machine) and together with the $\frac{x-1}{2}$ pairs of large jobs we apply the same transformation we have described for the case without a mixed machine in SOL. □

It remains to maintain a solution that satisfies Invariants 3.3 with a bounded migration factor when the new job j is in $\mathcal{H}\cup\mathcal{L}\cup\mathcal{M}$, that is, a non-tiny job. The huge jobs will be scheduled on dedicated machines, while the jobs in $\mathcal{L}\cup\mathcal{M}$ will be scheduled so as to satisfy the invariants 3.3 using a configuration-IP, that is an integer program we define below. The migration of large and medium jobs will be carried out whenever the new job is in $\mathcal{H}\cup\mathcal{L}\cup\mathcal{M}$ but we will not migrate huge jobs as long as they are huge. Whenever we decide to migrate some jobs we will fix the schedule of jobs previously assigned to some machines while we reschedule the jobs that were assigned to some constant number of machines. This includes the large and medium jobs that were assigned to those machines but also the tiny jobs that were assigned to those machines. Since the load of a machine not containing huge jobs was at most 2OPT while the size of the new job is at least εOPT (as it is not tiny), if we ensure that the number of machines whose previous assigned jobs are migrated is a constant (that may depend on ε) then the constant upper bound on the migration factor will follow. Namely, if this number of machines is at most τ, then the bound on the migration factor is at most $\frac{2\tau}{\varepsilon}$.

Configuration of a machine is an encoding of the multi-set of sizes of jobs $\mathcal{L}\cup\mathcal{M}$ that are scheduled on that machine. A configuration-IP means that we formulate a feasibility integer program whose decision variables are associated with the possible configurations, one variable per possible configuration, and a feasible solution for this integer program defines the assignment of jobs from $\mathcal{L}\cup\mathcal{M}$ to the machines not having huge jobs. Recall that we insist on satisfying the invariants, so we define the set of possible configurations as follows.

A set of jobs of load at most $(\frac{3}{2}+3\varepsilon)\cdot\text{OPT}$ is a possible configuration that has a corresponding decision variable. Furthermore, a pair of jobs where at least one of those is large, is a possible configuration. Observe that two identical sized jobs is a configuration that always satisfy the invariants, however, for a pair of distinct sized jobs of total size larger than $(\frac{3}{2}+3\varepsilon)\cdot\text{OPT}$, they correspond to a possible configuration, however, such configuration satisfies the invariants only if the sizes are consecutive in the sorted list of sizes and if this pair of sizes appears as a configuration of at most one machine. While we will have a decision variable for every pair of jobs of distinct sizes of total size larger than $(\frac{3}{2}+3\varepsilon)\cdot \textsc{opt}$, we will enforce the invariants by placing upper bound constraints on these variables where these upper bounds are either 0 or 1. We let $\mathcal{P}$ be the set of configurations of pair of jobs of distinct sizes with total size larger than $(\frac{3}{2}+3\varepsilon)\cdot \textsc{opt}$, and we let $\mathcal{C}$ be the set of other possible configurations. For a configuration $C \in \mathcal{C} \cup \mathcal{P}$ we assume that C is a vector that specifies the number of jobs of each size in this configuration, and we let x_C be the decision variable counting the number of machines of this configuration. With respect to the current solution, let μ be the number of machines that do not have a huge job (namely, $\mu = m - |\mathcal{H}|$), and let $n^{\rightarrow}$ be the vector with a component for each size of job that will be either medium or large (if it exists) whose corresponding value of this component is the number of jobs in the instance of this size.

Now, we are ready to formulate the integer program that is a feasibility problem, namely there is no objective function and the goal is to find a feasible integer solution to the following set of constraints where for $C \in \mathcal{P}$ we let $\delta(C)$ be 1 if the two sizes in C are currently consecutive in the list of distinct sizes of jobs in the current set of $\mathcal{L} \cup \mathcal{M}$ and $\delta(C) = 0$ otherwise.

$$\begin{array}{rl} \sum_{C\in\mathcal{P}\cup\mathcal{C}} x_C = \mu & \\ \sum_{C\in\mathcal{P}\cup\mathcal{C}} C\cdot x_C = n^{\rightarrow} & \\ x_C \le \delta(C) & \forall C \in \mathcal{P} \\ x_C \ge 0 & \forall C \in \mathcal{P}\cup\mathcal{C}\ . \end{array}$$

We are going to use the following sensitivity analysis result for upper bounding the change of feasible solutions due to bounded change of the right hand side. In our setting the common objective function v is the zero vector so every feasible solution is also optimal.

Theorem 3.8 [4] (see also Corollary 17.2a, [22]) *Let A be an integral $m \times d$ matrix such that each sub-determinant of A is at most Δ in absolute value, let $\hat{u}$ and u' be column m-vectors, and let v be a row d-vector. Suppose* $\max\{vx|Ax \le \hat{u};\ x$ *is integral*$\}$ *and* $\max\{vx|Ax \le u';\ x$ *is integral*$\}$ *are finite. Then, for each optimum solution y of the first maximum there exists an optimum solution y' of the second maximum such that* $||y-y'||_\infty \le d\Delta\left(||\hat{u}-u'||_\infty + 2\right)$.

Our use of this theorem is based on the following lemma.

Lemma 3.9 *Let A be the constraint matrix of the configuration-IP. Then A has at most $O((\log_{1+\varepsilon}\frac{1}{\varepsilon})^2)$ rows, and at most $O((\frac{2}{\varepsilon})^{\log_{1+\varepsilon}1/\varepsilon})$ columns. Each entry of A is a non-negative integer that is at most $\frac{2}{\varepsilon}$ so the maximum absolute value of a sub-determinant of A is at most*

$$\Delta \le \left(\frac{2}{\varepsilon}\right)^{O((\log_{1+\varepsilon}\frac{1}{\varepsilon})^2)} \cdot O((\log_{1+\varepsilon}\frac{1}{\varepsilon})^{O((\log_{1+\varepsilon}\frac{1}{\varepsilon})^2}\ .$$

Furthermore, when a new job is released (and the value of OPT *stays without changes), the configuration-IP is modified only through a change of the right hand side such that the infinity norm of the change is* 1.

Proof First consider the bounds on the matrix A. The number of rows in A is the number of constraints excluding the non-negativity constraints, and if we denote by ν the number of distinct sizes of jobs that are medium or large, then there is one row for $\sum_{C\in\mathcal{P}\cup\mathcal{C}} x_C = \mu$, there are ν rows for $\sum_{C\in\mathcal{P}\cup\mathcal{C}} C\cdot x_C = n^{\rightarrow}$ and there are $O(\nu^2)$ rows for $x_C \le \delta(C) \quad \forall C\in\mathcal{P}$. The claim now follows since $\nu = O(\log_{1+\varepsilon}\frac{1}{\varepsilon})$ due to the rounding and the fact that medium and large jobs have sizes in the interval $(\varepsilon\text{OPT}, \text{OPT}]$.

Next, consider the maximum entry of A, that is either 0 or 1 or represents the number of medium or large jobs (of a common size) that can be scheduled on one machine. Since the load of every machine is at most 2OPT and the size of every such job is at least εOPT, we conclude that the maximum entry of A as well as the maximum entry of every configuration is at most $\frac{2}{\varepsilon}$.

Now, consider the number of columns in A that is the number of possible configurations. These are a subset of vectors with ν components, where every component is a non-negative integer that is at most $\frac{2}{\varepsilon}$. Thus, the claim regarding the number of columns follows. The upper bound on Δ follows by the bound on the number of rows of such sub-matrix and the bound on the maximum entry in A.

Last, consider the changes for the configuration-IP once a new job j is released where this job is in $\mathcal{H}\cup\mathcal{L}\cup\mathcal{M}$. If j is a huge job, then the only change is the decrease of μ by 1, so the right hand side after j is released is changed from the right hand side before its release by exactly one component that is decreased by 1, so the claim is satisfied. If $j\in\mathcal{L}\cup\mathcal{M}$, there might be two types of changes. The first change is that the component of $n^{\rightarrow}$ corresponding to the size of j is increased by 1. The second type of changes occur only if j is the first job in the instance of its (rounded) size. In this last case the set of consecutive sizes of jobs is changing so at most three values of δ are modified between 0 and 1 (at most two δ values are changed from 0 to 1, while at most one value of δ is changed from 1 to 0). Observe that if j is tiny there are no changes to the configuration-IP. □

Our next goal is to establish the connection between feasible schedules that satisfy the invariants and solutions for the configuration-IP. Given a feasible schedule of non-huge jobs into μ machines that satisfy the invariants, we consider the schedule of the large and medium jobs to these μ machines. Then, we use the definition of components of a configuration in order to define a configuration for each machine. After defining a configuration for each machine, we count the number of machines of each configuration. We note that this is a feasible solution to the configuration-IP as the constraint $\sum_{C\in\mathcal{P}\cup\mathcal{C}} x_C = \mu$ is satisfied as each machine has exactly one configuration, $\sum_{C\in\mathcal{P}\cup\mathcal{C}} C\cdot x_C = n^{\rightarrow}$ are satisfied as every job is scheduled to exactly one machine, and $x_C \le \delta(C)$ are satisfied by the invariants. Thus, if prior to the release of the current job, we have a schedule that satisfies the invariants, then this solution implies a feasible solution for the configuration-IP.

Next, we modify the solution for the configuration-IP based on the new right hand side (after the release of the current job). By Lemma 3.9, we conclude that the configuration-IP has a feasible solution such that the infinity norm of the difference between the old solution and the new solution is bounded by a constant that is upper bounded by a function of ε, namely by $3d\Delta$ where $d = O((\frac{2}{\varepsilon})^{\log_{1+\varepsilon} 1/\varepsilon})$ and $\Delta \le \left(\frac{2}{\varepsilon}\right)^{O((\log_{1+\varepsilon}\frac{1}{\varepsilon})^2)} \cdot O((\log_{1+\varepsilon}\frac{1}{\varepsilon})^{O((\log_{1+\varepsilon}\frac{1}{\varepsilon})^2}$. We let $\tau = 3d\Delta$ that is a constant. We find a new solution for the configuration-IP with the modified right hand side such that the infinity norm of its difference from the previous solution is bounded by τ as follows. Observe that this norm difference is a linear inequality, so by imposing this new linear inequality we get an integer program in fixed dimension and by solving it using Lenstra's algorithm [17, 18] we get the required solution for the modified right hand side configuration-IP.

Next, we modify the schedule of the non-huge jobs to correspond to the new solution of the configuration-IP. To do that, we define a set of machines M' whose configurations should change. We do this one configuration at a time, and when the new solution has less machines with the current configurations (with respect to their number in the old solution) then we pick a minimal set of machines of that configuration in the old solution (so their number is exactly the difference of the two corresponding variables in the two solutions) and we add those machines to M'. All jobs previously scheduled to M' will be re-scheduled and no other job will migrate. This is sufficient for guaranteeing the constant migration factor as the set of machines M' has at most $3d\Delta$ times the number of columns in A machines and this is a function of ε, and the load of such machine was at most 2OPT while the size of the new arriving job is at least $\varepsilon \cdot$ OPT. Rescheduling the jobs in M' is done as follows. If the new job is huge, it means that one of the machines in M' is reserved to the new job, and the other machines are available for the currently non-scheduled non-tiny jobs. The resulting set of available machines without configuration is denoted as M''. Then, M'' is M' if the new job is non-huge or it has one machine less, if the new job is huge. Next, we define a configuration to each machine in M''. The configurations are defined so that the resulting number of machines of each configuration will be exactly the value of the corresponding decision variable in the new solution (for the configuration-IP with the new right hand side). By constraint $\sum_{C\in\mathcal{P}\cup\mathcal{C}} x_C = \mu$, this is indeed possible. Then, we allocate the non-scheduled non-tiny jobs to the machines in M'' based on the configurations of these machines. Last, the tiny jobs that were used to be scheduled on the machines in M' are re-inserted into the schedule while maintaining the invariants as shown in Lemma 3.7. Indeed the new schedule satisfies the invariants and we can continue to process the release of the next job.

We conclude the following result.

Theorem 3.10 *For every $\varepsilon > 0$, there is a polynomial time algorithm whose competitive ratio is $\frac{3}{2} + \varepsilon$ and whose migration factor is bounded by a function of ε.*

4 Algorithms with Constant Amortized Migration

We have established the existence of a $(\frac{3}{2} + \varepsilon)$-competitive algorithm with a constant migration factor, and showed that there is no better than $\frac{3}{2}$ competitive that maintains a constant migration factor. In this section we analyze the impact of allowing constant amortized migration factor where the possibility to migrate some jobs due to the arrival of other jobs might be postponed to later iterations of the algorithm. More precisely, in this section we design a polynomial time approximation scheme for PR- MIN with a constant amortized migration factor.

Let $\varepsilon > 0$ be such that $\frac{1}{\varepsilon} \geq 2$, so $\varepsilon \leq \frac{1}{2}$.

Preliminaries: The Offline Problem. First, we discuss the offline problem and we design a new PTAS for this problem such that we will be able to maintain its output with a constant amortized migration. Even though there are known approximation schemes for PR- MIN, it seems hard to adapt these schemes for the scenario with migration.

First, the size of every job is rounded up to the next integer power of $1+\varepsilon$. Next, we consider the resulting rounded instance and let OPT be the cost of an optimal solution rounded up to the next integer power of $1+\varepsilon$. Observe that we can indeed guess this value of OPT in the offline settings as this value is either 0 or in the interval $[\min_j p_j, (1+\varepsilon)\cdot\sum_j p_j]$.

In what follows, we will use the fact that we can find an approximated solution to the rounded instance and obtain a PTAS for that. The solution constructed for the rounded instance is the output of the PTAS for the original instance though the competitive ratio is increased by a multiplicative factor of $1+\varepsilon$. Thus, without loss of generality we assume that the original instance is already rounded and p_j is the (rounded) size of job j.

Next, we note that every job of size larger than OPT must be scheduled on a dedicated machine by any optimal solution for the problem. Letting $\mathcal{H}$ be the subset of jobs of size larger than OPT, we denote $\mu = m - |\mathcal{H}|$. We place every job of $\mathcal{H}$ on a dedicated machine and later we will schedule all other jobs (jobs not in $\mathcal{H}$) on the μ other machines. Let J' be the set of remaining jobs (the ones not in $\mathcal{H}$). We define a valid lower bound on the optimal cost:

$$LB = \frac{\sum_{j\in J'} p_j}{2\mu}.$$

Lemma 4.1 *We have that $LB \leq$ OPT $\leq 2\cdot(1+\varepsilon)\cdot LB$.*

Proof The inequality $LB \leq$ OPT follows, as each machine in the optimal solution that does not contain a job in $\mathcal{H}$ (there are μ such machines) has only jobs from J' and their total size is at most 2OPT. Thus, by summing over all these machines we conclude that $\sum_{j\in J'} p_j \leq 2\mu\cdot$ OPT and thus $LB = \frac{\sum_{j\in J'} p_j}{2\mu} \leq$ OPT.

The inequality OPT $\leq 2\cdot(1+\varepsilon)\cdot LB$ will follow as we next establish the existence of a feasible solution of the jobs in J' on μ machines with cost of at most $2\cdot LB$. We sort the jobs in non-increasing order of their size and schedule them to the machines in a next-fit way as follows. At each point in time the procedure has one open machine, and the next job (in the sorted list) is scheduled there. If the total size of jobs on this machine has exceeded the value of $2\cdot LB$, then this machine is closed and a new machine is defined as an open machine. Note that every closed machine has total size of jobs of more than $2\cdot LB$ but its cost is at most $2\cdot LB$. Thus, all jobs of J' are assigned before we close the last machine among the available μ machines. The resulting feasible solution shows that the optimal one has cost of at most $2 \cdot LB$, and thus after rounding the optimal cost we get at most $2 \cdot (1+\varepsilon) \cdot LB$ as required. □

In order to present the remaining steps of the scheme, we will use a value LB such that $LB \leq \text{OPT} \leq 2 \cdot (1+\varepsilon) \cdot LB$. One such choice of LB is given above, but any other formula that can be computed in polynomial time and satisfies this inequality is sufficient for the correctness of our PTAS as we show next and use later.

The jobs in $\mathcal{H}$ will be called huge jobs and J' is the set of non-huge jobs. We let LB' be a rounded up value of LB to the next integer power of $1+\varepsilon$. Since OPT is also an integer power of $1+\varepsilon$, and the value is rounded up, we conclude that $LB' \leq \text{OPT} \leq 2 \cdot (1+\varepsilon) \cdot LB'$. Define a job j as a *small* job if its size is smaller than $\varepsilon^2 \cdot LB'$, and *large* if its size is in the interval $[\varepsilon^2 \cdot LB', \text{OPT}]$. Our scheme finds the value of LB', and based on that value partitions the jobs of J' into small and large as follows. Based on the last lemma, the number of distinct sizes of large jobs is a constant (it is at most $\log_{1+\varepsilon} \frac{2}{\varepsilon^2} + 2$) that we denote as ψ.

The next step is to pack the small jobs into *bundles*. The size of each bundle is $\varepsilon \cdot LB'$ and this is an upper bound on the total size of small jobs packed into the bundle. We pack the small jobs into bundles using Next-Fit. That is, in every step (if there are small jobs) we have an open bundle that is used to pack the next small job, and the first such bundle is opened for the first small job. If the total size of jobs in the open bundle is at least $\varepsilon \cdot LB' - \varepsilon^2 \cdot LB'$, this open bundle is closed and a new bundle is opened (and declared as the open bundle). We note that scheduling the large jobs and bundles into μ machines can be done with cost of at most $(1+5\varepsilon) \cdot \text{OPT}$ as we show next. In what follows, we say that we schedule a bundle on a machine and mean that we schedule the entire set of small jobs packed into this bundle on the same machine.

Lemma 4.2 *There is a feasible solution of cost at most* $(1+5\varepsilon) \cdot \text{OPT}$ *that schedules the bundles and large jobs on* μ *machines.*

Proof First, we modify the solution such that the assignment of large jobs is the same as in the optimal solution while the assignment of small jobs is modified as follows. For a machine that used to obtain small jobs of total size σ, we assign some bundles, so that if $\sigma = 0$, then the number of bundles is 0, while if $\sigma > 0$, the number of bundles assigned to this machine is $\lceil \frac{\sigma}{\varepsilon \cdot LB' - \varepsilon^2 \cdot LB'} \rceil + 1$. Observe that the number of closed bundles is at most the total size of small jobs divided by $\varepsilon \cdot LB' - \varepsilon^2 \cdot LB'$ and since there is at least one machine for which $\sigma > 0$ we will have an available slot for the open bundle as well. Moreover, in the case $\sigma > 0$, the machine with these jobs has completion time of at least σ, and its smallest job has size at most $\varepsilon^2 \cdot \text{OPT}$, and therefore, the cost for this machine is at least $\sigma - \varepsilon^2\text{OPT}$, so $\text{OPT} \geq \sigma - \varepsilon^2\text{OPT}$, and therefore $\sigma \leq (1+\varepsilon^2)\text{OPT}$.

For a machine that contained no small jobs in the optimal solution, we have a cost that equals the cost of the optimal solution on that machine (and in particular at most OPT). For a machine that in the optimal solution contained small jobs of total size $\sigma > 0$, the total size of jobs assigned to that machines in the optimal solution is at most $(1+\varepsilon^2) \cdot \text{OPT}$ (as it contained a small job), while we schedule jobs and bundles of total size at most $(1+\varepsilon^2) \cdot \text{OPT} - \sigma + \left(\lceil \frac{\sigma}{\varepsilon \cdot LB' - \varepsilon^2 \cdot LB'} \rceil + 1\right) \cdot \varepsilon \cdot LB' \leq$

$(1+\varepsilon^2)\cdot \text{OPT} + \sigma(-1+\frac{1}{1-\varepsilon}) + 2\varepsilon \cdot LB' \le (1+\varepsilon^2+4\varepsilon)\cdot \text{OPT}$. The last inequality holds using $\sigma \le (1+\varepsilon^2)\text{OPT}$ and $LB' \le \text{OPT}$, and by $\varepsilon^2 \le \frac{\varepsilon}{4}$ (since $\varepsilon \le \frac{1}{2}$), and we get at most $(1+5\varepsilon)\cdot \text{OPT}$. □

Next, we consider the problem of placing the large jobs and bundles into μ machines. We formulate this problem as a configuration IP that will be an integer program of fixed dimension with $\psi + 2$ constraints (excluding the non-negativity constraints on all variables). While the configuration IP of this section is similar to the one of the previous section, these are not the same ones. The decision variables of this integer program are once again counters of the number of machines of a given *configuration*. That is, we define a *configuration of a machine* in a schedule as a vector with $\psi + 1$ components, the first ψ components are the number of jobs of each size of large jobs that are scheduled on the machine, while the last component is the number of bundles which are scheduled on this machine. If the number of bundles of a given configuration is at least 1, we say that the cost of the configuration is the total size of jobs and bundles on this machine, while if the number of bundles is zero, then we compute the total size of jobs excluding one large job of the smallest size whose corresponding component (in the configuration vector) is non-zero. In order to solve the problem, by Lemma 4.2, it suffices to consider the set of configurations $\mathcal{C}$ whose costs are at most $(1+6\varepsilon)\text{OPT}$.

The constraints of this configuration IP are as follows. First, the sum of the decision variables is μ, second, the sum of the product of the configuration and the decision variable counting the number of machines with this configuration is equal to the number of large jobs and bundles of the corresponding sizes. Thus, if we denote by $n^{\rightarrow}$ the vector of dimension $\psi + 1$ specifying the number of large jobs of each size and the number of bundles, and we denote by x_C the decision variable of the configuration IP for configuration C, then the configuration IP is to find a feasible solution for the following integer program.

$$\begin{aligned} \textstyle\sum_{C\in\mathcal{C}} x_C &= \mu \\ \textstyle\sum_{C\in\mathcal{C}} C\cdot x_C &= n^{\rightarrow} \\ x_C &\ge 0 \qquad \forall C \in \mathcal{C}. \end{aligned}$$

Since the number of large jobs and bundles in such configurations is upper bounded by a constant (at most $\frac{5}{\varepsilon^2}$, since the total size of jobs assigned to a machine is at most $\text{OPT}\cdot(1+6\varepsilon)+\text{OPT} \le 5\cdot LB'$, where the smallest job size is $\varepsilon^2 \cdot LB'$), the number of configurations is also a constant. Thus, the integer program can be solved in polynomial time. To obtain an actual schedule, we can assign the large jobs and bundles according to the optimal solution of this integer program, and the resulting cost does not exceed the maximum cost of a configuration of the IP. This concludes the description of the offline PTAS.

Transforming the Offline PTAS into a PTAS with Constant Amortized Migration Factor. We would like to obtain an algorithm that imitates the offline PTAS. Here, it is also assumed that all job sizes are powers of $(1+\varepsilon)$, by rounding the sizes and never considering the original sizes. The value OPT is still the optimal cost for the rounded sizes, also rounded up to the next power of $(1+\varepsilon)$, where jobs are considered as they are, without bundles. The value OPT is monotonically non-decreasing throughout the execution since the set of jobs is only augmented along time. However, in order to obtain a schedule, the small jobs are assigned to bundles. Such bundles are treated as jobs for the purpose of assignment, exactly as in the offline version of the PTAS. The algorithm often assigns small jobs to an existing bundle, and occasionally it opens a new bundle, at times explained below.

Similarly, we keep track of the value of LB in every prefix of the input, and note that by the monotonicity of the optimal cost, we can revise the definition of LB that will be the maximum value of this parameter corresponding to a prefix of the current input. Once using this new definition for the value of LB, we let LB' be the value of LB rounded up to the next integer power of $1+\varepsilon$. Since the value of OPT is monotone, we maintain the required property that $LB \le \text{OPT} \le 2 \cdot (1+\varepsilon)LB$, and since OPT is an integer power of $1+\varepsilon$ then $LB' \le \text{OPT} \le 2 \cdot (1+\varepsilon) \cdot LB'$. Thus, as mentioned in the offline case, the resulting algorithm with this definition of LB is indeed a PTAS. Furthermore, in iterations in which LB' changes, it increases by a multiplicative factor of at least $1+\varepsilon$ and cannot decrease. Furthermore, in such step the value of LB is defined as $LB = \frac{\sum_{j \in J'} p_j}{2\mu}$ (and it is not defined as the outcome of this formula for a strict prefix of the current input sequence).

In each step, the solution will be a suitable output of the PTAS. We have iterations in steady states and once in a while we have an iteration in a non-steady state. A steady state is as long as there are no changes in the crucial values: OPT, LB', and μ, where a change in OPT may trigger a change in the set $\mathcal{H}$, resulting in a change in the values μ and LB' (and the value LB' may change even without a change in $\mathcal{H}$), and we will also consider such situations of non-steady state. In the steady state, there will be changes in the set of jobs due to arrivals of jobs, which require modifications of the schedule. The value LB' is a power of $1+\varepsilon$ and thus it is possible that an arrival of a job will not change it. Since OPT will never decrease, jobs that are not new cannot be inserted back to $\mathcal{H}$. It is possible that a job moves from $\mathcal{H}$ to J' after an increase in the value OPT (in which case the value of μ increases), and it is also possible that a new job joins $\mathcal{H}$, such that the value of μ decreases by 1. We use the monotonicity assumption of LB that holds by its modified definition to conclude that the value of LB' never decreases.

Our constant amortized migration scheme is divided into phases where a phase is a maximal consecutive set of jobs in the input sequence for which the value of LB' is the same. Observe that the value of LB' can only increase and in every such increase its value is increased by a multiplicative factor of (at least) $1+\varepsilon$. In every such increase of LB' our scheme reschedules all jobs in J' on μ machines (both J' and μ are the values of these arguments after considering the new job as well) using the offline approximation scheme (using the modified definition of LB, but in such steps this new definition is the same as the original one). In such events the set of migrated jobs might be the entire set J'. In particular in such iteration all bundles are split and created from scratch.

Furthermore, in every phase, when the offline approximation scheme is about to open the first new bundle during the current phase, we reschedule all jobs of J' according to the offline approximation scheme. These are the events in which we reschedule all jobs of the current J' using the amortized migration power of jobs that had arrived or moved from $\mathcal{H}$ to J' during the last few phases. Next, we upper bound the impact on the amortized migration factor due to these rescheduling events.

Lemma 4.3 *Let $\tilde{J}$ be the set of jobs J' at the end of phase i (and without the job that arrives and causes the start of phase $i+1$), and let S be the set of jobs that either arrive during phase i or moved from $\mathcal{H}$ to J' during phase i including the job that causes the start of phase $i+1$. Then, $\sum_{j\in\tilde{J}} p_j \le \frac{16\cdot\sum_{j\in S} p_j}{\varepsilon}$.*

Proof Let μ_{i-1}, μ_i be the value of μ at the end of phase $i-1$ and after the first job of phase $i+1$, respectively. First assume that $\mu_i \le (1-\frac{\varepsilon}{3})\cdot\mu_{i-1}$ or $\mu_i \ge (1+\frac{\varepsilon}{3})\cdot\mu_{i-1}$. Then in the set S there are at least $\frac{\varepsilon}{4}\cdot\mu_i$ jobs that were either in $\mathcal{H}$ at the end of phase $i-1$ and were moved to J' until the first iteration of phase $i+1$ or jobs that arrived during this period and were added to $\mathcal{H}$ upon arrival. These jobs are of sizes at least LB' (that is the value of LB' during the i-th phase). Thus, the total size of these jobs is at least $\frac{\varepsilon}{4}\cdot\mu_i\cdot LB'$, while by Lemma 4.1 the total size of jobs in $\tilde{J}$ is at most $4\cdot LB'\cdot\mu_i$ and the claim follows in this case.

Thus, in the remaining case we assume that $(1+\frac{\varepsilon}{3})\cdot\mu_{i-1} > \mu_i > (1-\frac{\varepsilon}{3})\cdot\mu_{i-1}$. Let $\hat{J}$ be the set of jobs J' at the end of phase $i-1$. Then, we know that $\hat{J}\subseteq\tilde{J}\subseteq S\cup\hat{J}$. We know that the value of LB'' that is defined using the original formula for LB (that need not be monotone) at the end of phase $i-1$ is $\frac{\sum_{j\in\hat{J}} p_j}{2\mu_{i-1}} \le \frac{(1+\frac{\varepsilon}{3})\cdot\sum_{j\in\hat{J}} p_j}{2\mu_i}$ and thus the value of $2\mu_i\cdot LB''$ at the end of phase $i-1$ is at most $(1+\frac{\varepsilon}{3})\cdot\sum_{j\in\hat{J}} p_j$. On the other hand, the value of LB'' after the first iteration of phase $i+1$ (and since LB' increases at this iteration, we conclude that $LB''=LB$) is at least $(1+\varepsilon)$ times the value of LB'' at the end of phase $i-1$ that is at least $(1+\varepsilon)\cdot\frac{\sum_{j\in\hat{J}} p_j}{2\mu_{i-1}} \ge (1+\varepsilon)\cdot(1-\frac{\varepsilon}{3})\cdot\frac{\sum_{j\in\hat{J}} p_j}{2\mu_i}$. Therefore, after the first iteration of phase $i+1$ the value of $2\mu_i\cdot LB''$ that by definition is $\sum_{j\in\tilde{J}\cup S} p_j$ satisfies that it is at least $(1+\varepsilon)\cdot(1-\frac{\varepsilon}{3})\cdot\sum_{j\in\hat{J}} p_j$. By $\tilde{J}\subseteq S\cup\hat{J}$, we conclude that $\sum_{j\in S} p_j \ge \sum_{j\in\tilde{J}\cup S} p_j - \sum_{j\in\hat{J}} p_j \ge \left[(1+\varepsilon)\cdot(1-\frac{\varepsilon}{3})-1\right]\cdot\sum_{j\in\hat{J}} p_j \ge \frac{\varepsilon}{3}\cdot\sum_{j\in\hat{J}} p_j$ and the claim follows in this case as well. □

Thus, the rescheduling of the jobs of J' at most twice in every phase could be charged to the jobs that arrive (or moved to J') in phases $i, i-1$ and $i-2$ so every job is charged at most six times for these operations. Thus, in order to maintain a constant amortized migration it suffices to consider iterations in which the value of LB' is not changed and these iterations are not the first times of their phase in which we open a new bundle for small jobs.

In order to analyze the migrations in other cases, we are going to use the sensitivity result for integer programming, namely Theorem 3.8 with respect to a change of the right hand side of the configuration IP. Our modifications of the right hand side will have an infinity norm of 1, so in order to bound the change of the solution for the configuration IP, we need to bound the parameters of its constraint matrix. Let A be the constraint matrix of the configuration IP (of this section). Then, in A there are $\psi+2$ rows, that is, at most $\log_{1+\varepsilon}\frac{2}{\varepsilon^2}+4$ rows, and each component is at most $\frac{5}{\varepsilon^2}$. Furthermore, the number of columns in A is the number of configurations that is at most the number of non-negative integer vectors with $\psi+1$ components where every component is at most $\frac{5}{\varepsilon^2}$. Thus, the number of columns in A is at most $d \leq \left(\frac{5}{\varepsilon^2}+1\right)^{\log_{1+\varepsilon}\frac{2}{\varepsilon^2}+3}$. The maximum absolute value of a sub-determinant of A is at most $\Delta \leq \left(\frac{5}{\varepsilon^2}\cdot(\log_{1+\varepsilon}\frac{2}{\varepsilon^2}+3)\right)^{\log_{1+\varepsilon}\frac{2}{\varepsilon^2}+3}$. By Theorem 3.8, if the infinity norm of the change of the right hand side of the configuration IP is at most 1, then there is a new solution (with respect to the modified right hand side) whose distance in the 1-norm from the old solution is at most $3d^2\Delta \leq 3\cdot\left(\frac{5}{\varepsilon^2}+1\right)^{2\cdot(\log_{1+\varepsilon}\frac{2}{\varepsilon^2}+3)}\cdot\left(\frac{5}{\varepsilon^2}\cdot(\log_{1+\varepsilon}\frac{2}{\varepsilon^2}+3)\right)^{\log_{1+\varepsilon}\frac{2}{\varepsilon^2}+3}$. We denote this constant (depending on ε) by ϕ.

In each such iteration in which the value of OPT increases, some jobs may move from $\mathcal{H}$ to J'. We first consider the previous solution of the configuration IP with respect to the new value of OPT. This interpretation does not cause any migration. It simply means adding more configurations to $\mathcal{C}$ with zero valued decision variables, and considering the jobs that were moved from $\mathcal{H}$ to J' as jobs which are scheduled according to the configurations of scheduling each such job on a dedicated machine. Next, we consider the maintenance of the solution upon arrival of a new job where both LB' and OPT are not changed.

First, if the new job is a small job, then we pack it into an open bundle (this operation does not cause migration of jobs), and if we need to open a new bundle, it means that the right hand side of the configuration IP is changed by an additive $+1$ in the component of the constraint saying that we need to schedule all bundles. The resulting integer program is solved optimally and using the sensitivity theorem for integer programming, that is, Theorem 3.8 we conclude that we need to reschedule jobs on a constant number of machines (among the μ machines for non-huge jobs). This rescheduling of a constant number of machines is charged to the small jobs of the bundle we have closed while we open the new bundle for small jobs.

Next, consider the case where the new job is a huge job, then in this case we need to release one machine for this new coming job. To do that we modify the right hand side of the configuration IP by decreasing the number of available machines for scheduling all large and small jobs (i.e., the right hand side of the first constraint) to $\mu-1$. Once again using Theorem 3.8 we conclude that we need to reschedule jobs on a constant number of machines and this rescheduling of jobs is charged to the new coming job (of size at least LB').

Last, consider the case that the new arriving job is a large job. We modify the right hand side of the configuration IP by increasing by 1 the number of jobs of the size of the new job. Once again by Theorem 3.8, we conclude that we need to reschedule jobs on a constant number of machines and this rescheduling of jobs is charged to the new coming job (of size at least $\varepsilon^2\cdot LB'$).

In all cases, we modify the schedule of a ϕ number of machines whose jobs are from J' (each of these machines has jobs of total size at most $6\cdot LB'$) and charge this rescheduling operation to a set of jobs of total size at least $\varepsilon^2\cdot LB'$, and each job is charged at most once for these operations. Thus, we incur an amortized migration factor of $\frac{6\phi}{\varepsilon^2}$ due to these iterations. The total amortized migration factor is this value plus the one due to the charging of a new phase and the first time we open a new bundle in a phase (of at most $\frac{96}{\varepsilon}$). Thus, the amortized migration factor is at most $\frac{7\phi}{\varepsilon^2}$, and we conclude the following result.

Theorem 4.4 *For every $\varepsilon > 0$, there is a polynomial time algorithm whose competitive ratio is at most $1 + \varepsilon$ and whose amortized migration factor is upper bounded by a constant.*

We note that in iterations where we do not perform rescheduling of the entire set J', the value of LB' is not modified and thus the partition of jobs into large and small does not change including the definition of bundles so we can indeed maintain the property that in every step there is at most one open bundle for small jobs and there is such a bundle only if the instance so far has at least one small job.

Funding Partially supported by a grant from GIF - the German-Israeli Foundation for Scientific Research and Development (Grant Number I-1366-407.6/2016) and by grants from ISF - Israeli Science Foundation (Grant Numbers 308/18 and 1467/22).

Declarations

Conflict of interest The author has no relevant financial or non-financial interests to disclose. The author has no competing interests to declare that are relevant to the content of this article.

References

1. Berndt, S., Dreismann, V., Grage, K., Jansen, K., Knof, I.: Robust online algorithms for certain dynamic packing problems. In Proceedings of the 17th International Workshop on Approximation and Online Algorithms, (WAOA 2019), pp. 43–59 (2019)
2. Berndt, S., Epstein, L., Jansen, K., Levin, A., Maack, M., Rohwedder, L.: Online bin covering with limited migration. In: Proceedings of the 27th Annual European Symposium on Algorithms, ESA 2019, vol. 144, pp. 18:1–18:14 (2019)
3. Berndt, S., Jansen, K., Klein, K.: Fully dynamic bin packing revisited. Math. Program. **179**(1), 109–155 (2020)
4. Cook, W., Gerards, A., Schrijver, A., Tardos, É.: Sensitivity theorems in integer linear programming. Math. Program. **34**(3), 251–264 (1986)
5. Epstein, L., Levin, A.: Asymptotic fully polynomial approximation schemes for variants of open-end bin packing. Inf. Process. Lett. **109**(1), 32–37 (2008)
6. Epstein, L., Levin, A.: A robust APTAS for the classical bin packing problem. Math. Program. **119**(1), 33–49 (2009)
7. Epstein, L., Levin, A.: Robust approximation schemes for cube packing. SIAM J. Optim. **23**(2), 1310–1343 (2013)
8. Epstein, L., Levin, A.: Robust algorithms for preemptive scheduling. Algorithmica **69**(1), 26–57 (2014)
9. Epstein, L., Levin, A.: Robust algorithms for total completion time. Discrete Optim. **33**, 70–86 (2019)
10. Epstein, L., Levin, A.: Starting time minimization for the maximum job variant. Discrete Appl. Math. **307**, 79–87 (2022)
11. Epstein, L., Tassa, T.: Approximation schemes for the min–max starting time problem. Acta Inform. **40**(9), 657–674 (2004)
12. Epstein, L., van Stee, R.: Minimizing the maximum starting time on-line. Inf. Comput. **195**(1–2), 53–65 (2004)
13. Feldkord, B., Feldotto, M., Gupta, A., Guruganesh, G., Kumar, A., Riechers, S., Wajc, D.: Fully-dynamic bin packing with little repacking. In: Proceedings of the 45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, pp. 51:1–51:24 (2018)
14. Gálvez, W., Soto, J.A., Verschae, J.: Symmetry exploitation for online machine covering with bounded migration. ACM Trans. Algorithms **16**(4), 43:1-43:22 (2020)

15. Jansen, K., Klein, K.: A robust AFPTAS for online bin packing with polynomial migration. SIAM J. Discrete Math. **33**(4), 2062–2091 (2019)
16. Jansen, K., Klein, K., Kosche, M., Ladewig, L.: Online strip packing with polynomial migration. In: Proceedings of the 20th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2017, pp. 13:1–13:18 (2017)
17. Kannan, R.: Improved algorithms for integer programming and related lattice problems. In: Proceedings of the 15th Annual ACM Symposium on Theory of Computing, STOC 1983, pp. 193–206 (1983)
18. Lenstra, H.W., Jr.: Integer programming with a fixed number of variables. Math. Oper. Res. **8**(4), 538–548 (1983)
19. Leung, J.Y.-T., Dror, M., Young, G.H.: A note on an open-end bin packing problem. J. Sched. **4**(4), 201–207 (2001)
20. Lu, L., Zhang, L.: Online single machine scheduling to minimize the maximum starting time. Asia-Pac. J. Oper. Res. **34**(5). Paper number 1750022, 9 pp (2017)
21. Sanders, P., Sivadasan, N., Skutella, M.: Online scheduling with bounded migration. Math. Oper. Res. **34**(2), 481–498 (2009)
22. Schrijver, A.: Theory of Linear and Integer Programming. Wiley, Hoboken (1986)
23. Skutella, M., Verschae, J.: Robust polynomial-time approximation schemes for parallel machine scheduling with job arrivals and departures. Math. Oper. Res. **41**(3), 991–1021 (2016)
24. Verschae, J.: Robust scheduling algorithms. In: Encyclopedia of Algorithms, pp. 1863–1865 (2016)
25. Zhang, G.: Parameterized on-line open-end bin packing. Computing **60**(3), 267–274 (1998)