Symmetry Exploitation for Online Machine Covering with Bounded Migration

WALDO GÁLVEZ, Department of Informatics, Technical University of Munich, Germany
JOSÉ A. SOTO, DIM & CMM, Universidad de Chile and UMI-CNRS 2807, Chile
JOSÉ VERSCHAE, Instituto de Ingeniería Matemática y Computacional, Facultad de Matemáticas y Escuela de Ingeniería, Pontificia Universidad Católica de Chile, Chile

Online models that allow recourse can be highly effective in situations where classical online models are too pessimistic. One such problem is the online machine covering problem on identical machines. In this setting, jobs arrive one by one and must be assigned to machines with the objective of maximizing the minimum machine load. When a job arrives, we are allowed to reassign some jobs as long as their total size is (at most) proportional to the processing time of the arriving job. The proportionality constant is called the migration factor of the algorithm.

Using a rounding procedure with useful structural properties for online packing and covering problems, we design first a simple $(1.7 + \varepsilon)$-competitive algorithm using a migration factor of $O(1/\varepsilon)$, which maintains at every arrival a locally optimal solution with respect to the Jump neighborhood. After that, we present as our main contribution a more involved $(4/3 + \varepsilon)$-competitive algorithm using a migration factor of $O(1/\varepsilon^3)$. At every arrival, we run an adaptation of the Largest Processing Time first (LPT) algorithm. Since the new job can cause a complete change of the assignment of smaller jobs in both cases, a low migration factor is achieved by carefully exploiting the highly symmetric structure obtained by the rounding procedure.

CCS Concepts: • Theory of computation → Approximation algorithms analysis; Scheduling algorithms; Online algorithms;

Additional Key Words and Phrases: Machine covering, bounded migration, online, scheduling, LPT

ACM Reference format:
Waldo Gálvez, José A. Soto, and José Verschae. 2020. Symmetry Exploitation for Online Machine Covering with Bounded Migration. ACM Trans. Algorithms 16, 4, Article 43 (July 2020), 22 pages.
https://doi.org/10.1145/3397535

This work was partially supported by SNSF Excellence Grant 200020B_182865/1 and CONICYT-Chile through projects FONDECYT 1181527 and 1181180, PCI PII 20150140 and PIA AFB170001.

Authors’ addresses: W. Gálvez, Department of Informatics, Technical University of Munich, Garching, Boltzmannstrasse 3 Germany; email: galvez@in.tum.de; J. A. Soto, DIM & CMM, Universidad de Chile & UMI-CNRS 2807, Beauchef 851, Santiago, Chile; email: jsoto@dim.uchile.cl; J. Verschae, Instituto de Ingeniería Matemática y Computacional, Facultad de Matemáticas y Escuela de Ingeniería, Pontificia Universidad Católica de Chile, Vicuña Mackenna 4860, Santiago, Chile; email: jverschae@uc.cl.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2020 Association for Computing Machinery.
1549-6325/2020/07-ART43 $15.00
https://doi.org/10.1145/3397535

ACM Transactions on Algorithms, Vol. 16, No. 4, Article 43. Publication date: July 2020.
1 INTRODUCTION

We consider a fundamental load balancing problem where $n$ jobs need to be assigned to $m$ identical parallel machines. Each job $j$ is fully characterized by a non-negative processing time $p_j$. Given an assignment of jobs, the load of a machine is the sum of the processing times of jobs assigned to it. The machine covering problem asks for an assignment of jobs to machines maximizing the load of the least loaded machine.

This problem is well known to be strongly NP-hard and allows for a polynomial-time approximation scheme (PTAS) [20]. A well-studied algorithm for this problem is the Largest Processing Time First (LPT) rule, which sorts the jobs non-increasingly and assigns them iteratively to the least loaded machine. Deuermeyer et al. [5] show that LPT is a $\frac{4}{3}$-approximation and that this factor is asymptotically tight; later, Csirik et al. [4] refine the analysis giving a tight bound for each $m$.

In the online setting, jobs arrive one after another, and at each arrival, we must decide on a machine to assign the arriving job. This natural problem does not admit a constant competitive ratio. Deterministically, the best possible competitive ratio is $m$ [20], while randomization allows for a $\tilde{O}(\sqrt{m})$-competitive algorithm, which is the best possible up to logarithmic factors [1].

Dynamic model. The previous negative facts motivate the study of relaxed online scenarios such as the one with bounded migration. Unlike the classic online model, when a new job $j$ arrives we are allowed to reassign other jobs. More precisely, given a constant $\beta > 0$, we can migrate jobs whose total size is upper bounded by $\beta p_j$. The value $\beta$ is called the migration factor, and it accounts for the robustness of the computed solutions. In one extreme, we can model the usual online framework by setting $\beta = 0$. In the other extreme, setting $\beta = \infty$ allows to compute the optimal offline solution in each iteration. Our main interest is to understand the exact tradeoff between the migration factor $\beta$ and the competitiveness of our algorithms. Besides being a natural problem with an interesting theoretical motivation, its original purpose was to find good algorithms for a problem in the context of Storage Area Networks (SAN) [16]. More precisely, a SAN usually connects many disks of different capacity, grows over time, and can be thought as a single big, fault-tolerant disk of huge capacity and throughput. A simple scheme that implements this concept is a partition of the SAN into several subservers of about equal capacity. In our framework, disks correspond to jobs, disk capacities to job processing times, subservers to machines, and the capacity of the server is determined by the minimum capacity of a subserver. Moreover, reconfiguring the whole system when a new disk is added is not acceptable due to the amount of data that may be required to move. Rather, the user expects a “proportionate response” in terms of data migration, meaning that if a disk of $x$ GB is added, then it would be fine to move this order of magnitude of data, but not much more. This sort of proportionate response has shown to already help in handling the aforementioned limitations of the classical online model [16], and it is captured by the bounded migration model.

Local search and migration. The local search method has been extensively used to tackle different hard combinatorial problems, and it is closely related to online algorithms where recourse is allowed. This comes from the fact that simple local search neighborhoods allow to get considerably improved solutions while having accurate control over the recourse actions needed, and in some cases even a bounded number of local moves leads to good enough approximation guarantees (see Refs [9], [13], and [14] for examples in the context of network design).

Related Work. Sanders et al. [16] develop online algorithms for load balancing problems in the migration framework. For the makespan minimization objective, where the goal is to minimize the maximum load, they give a $(1 + \epsilon)$-competitive algorithm with migration factor $2^{O(1/\epsilon^2)}$. A major open problem in this area is to determine whether a migration factor of $\text{poly}(1/\epsilon)$ is achievable.

The landscape for the machine covering problem is somewhat different. Sanders et al. [16] give a 2-competitive algorithm with migration factor 1, and this is, until now, the best competitive
ratio known for any algorithm with constant migration factor. On the negative side, Skutella and Verschae [18] show that it is not possible to maintain arbitrarily near optimal solutions using a constant migration factor, giving a lower bound of $20/19$ for the best competitive ratio achievable in that case. The lower bound is based on an instance where arriving jobs are small enough so that no migration of larger jobs is allowed. This motivated the study of an amortized version, called reassignment cost model, where they develop a $(1 + \epsilon)$-competitive algorithm using a constant reassignment factor. They also show that if the arriving jobs are larger than $\epsilon \cdot OPT$, where OPT denotes the optimal minimum load at each iteration, then there is a $(1 + \epsilon)$-competitive algorithm with constant migration factor.

Similar migration models have been studied for other packing and covering problems. For example, Epstein and Levin [6] designed a $(1 + \epsilon)$-competitive algorithm for the online bin packing problem using a migration factor of $2\tilde{O}(1/\epsilon^2)$, which was improved later by Jansen and Klein [11] to poly$(1/\epsilon)$ migration factor, and then further refined by Berndt et al. [2]. Also, for makespan minimization with preemption and other objectives, Epstein and Levin [7] designed a best-possible online algorithm using a migration factor of $(1 - \frac{1}{m})$.

Regarding local search applied to load balancing problems, many neighborhoods have been studied such as Jump, Swap, Push, and Lexicographical Jump in the context of makespan minimization on related machines [17], makespan minimization on restricted parallel machines [15], and also multi-exchange neighborhoods for makespan minimization on identical parallel machines [8]. For the case of machine covering, Chen et al. [3] studied the Jump neighborhood in a game-theoretical context, proving that every locally optimal solution is 1.7-approximate and that this factor is tight.

**Our Contribution.** Our main result is a $(4/3 + \epsilon)$-competitive algorithm using poly$(1/\epsilon)$ migration factor, achieved by running a carefully crafted version of LPT at the arrival of each new job. We would like to stress that, even though LPT is a simple and very well-studied algorithm in the offline context, directly running this algorithm in each timestep in the online context yields an unbounded migration factor; see Figure 1 for an illustrative example and Lemma 2.5 in Section 2.2 for a proof.

To overcome this barrier, we first adapt a less standard rounding procedure to the online framework. Roughly speaking, the rounding reduces the possible number of sizes of jobs larger than $\Omega(\epsilon OPT)$ (where OPT is the offline optimum value) to $\tilde{O}(1/\epsilon)$ many values; furthermore, these values are multiples of a common number $g \in \Theta(\epsilon^2 OPT)$. This implies that the number of possible loads for machines having only big jobs is constant since they are multiples of $g$ as well. Unlike known techniques used in previous work that yield similar results (see, e.g., Ref. [12]), our rounding is well suited for online algorithms and helps simplify the analysis as it does not depend on OPT (which varies through iterations).
In order to show the usefulness of the rounding procedure, we first present a simple \((1.7 + \varepsilon)\)-competitive algorithm using a migration factor of \(O(1/\varepsilon)\). This algorithm maintains through the arrival of new jobs a locally optimal solution with respect to Jump for large jobs followed by a greedy assignment for small jobs. Although, for general instances, this can induce a very large migration factor as discussed before, for rounded instances, we can have a very accurate control on the jumps needed to reach a locally optimal solution by exploiting the fact that there are constantly many possible processing times for large jobs.

In the second part of the article, we proceed with the analysis of our \((4/3 + \varepsilon)\)-competitive algorithm. Here, we crucially make use of the properties obtained by the rounding procedure to create symmetries. After a new job arrival, we re-run the LPT algorithm for the new instance. While assigning a job to a current least loaded machine, since there is a constant number of possible machine loads, there will usually be multiple least loaded machines to assign the job. All options lead to different (but symmetric) solutions in terms of job assignments, all having the same load vector and, thus, the same objective value. Broadly speaking, the algorithm will construct one of these symmetric schedules, trying to maintain as many machines with the same assignments as in the previous timestep. The analysis of the algorithm will rely on monotonicity properties implied by LPT, which, coupled with rounding, implies that for every job size, the increase in the number of machines with different assignments (w.r.t. the solution of the previous timestep) is constant. This finally yields a migration factor that only grows polynomially in \(1/\varepsilon\). We finish by commenting some limitations of the standard geometric rounding procedure in our developed approach, and then presenting a lower bound of \(17/16\) for the best competitive ratio achievable by an algorithm with constant migration, improving the previous bound by Skutella and Verschae [18].

2 PRELIMINARIES

Consider a set of \(n\) jobs \(J\) and a set of \(m\) machines \(M\). In our problem, a solution or schedule \(S : J \rightarrow M\) corresponds to an assignment of jobs to machines. The set of jobs assigned to a machine \(i\) is then \(S^{-1}(i) \subseteq J\). The load of machine \(i\) in \(S\) corresponds to \(\ell_i(S) = \sum_{j \in S^{-1}(i)} p_j\). The minimum load is denoted by \(\ell_{\text{min}}(S) = \min_{i \in M} \ell_i(S)\), and a machine \(i\) is said to be least loaded in \(S\) if \(\ell_i(S) = \ell_{\text{min}}(S)\).

For an algorithm \(A\) and an instance \((J, M)\), we denote by \(S_A(J, M)\) the schedule returned by \(A\) when run on \((J, M)\). Similarly, \(S_{\text{OPT}}(J, M)\) denotes the optimal schedule, being OPT\((J, M)\) its minimum load. When it is clear from the context, we will drop the dependency on \(J\) or \(M\).

2.1 Algorithms with a Robust Structure

An important fact used in the design of the robust PTAS for makespan minimization from Sanders et al. [16] is that small jobs can be assigned greedily almost without affecting the approximation guarantee. This is, however, not the case for machine covering; see, e.g., Ref. [18] or Section 5. One way to avoid this inconvenience is to restrict our attention to algorithms that are oblivious to the arrival of small jobs, that is, algorithms where the assignment of big jobs is not affected when a new small job arrives.

**Definition 2.1.** Let \(h \in R_+\). An algorithm \(A\) has a robust structure at level \(h\) if, for any instance \((J, M)\) and \(j^* \notin J\), such that \(p_{j^*} < h\), \(S_A(J, M)\) and \(S_A(J \cup \{j^*\}, M)\) assign to the same machines all the jobs in \(J\) with processing time at least \(h\).

This definition highlights also the usefulness of working with the LPT rule, since the addition of a new small job to the instance does not affect the assignment of larger jobs. Indeed, it is easy to see the following.

**Proposition 2.2.** For any \(h \in R_+\), LPT has a robust structure at level \(h\).
Symmetry Exploitation for Online Machine Covering with Bounded Migration

The claim follows since the subsequence of sorted jobs with processing time at least \( h \) is the same in \( J^* \) and \( J \cup \{ f^* \} \); hence, LPT assigns these jobs to the same machines in both solutions.

We proceed now to define relaxed solutions where, roughly speaking, small jobs are added greedily on top of the assignment of big jobs.

**Definition 2.3.** Let \( \mathcal{A} \) be an \( \alpha \)-approximation algorithm for the machine covering problem, with \( \alpha \geq 1 \) constant, \( 1 \leq k_1 \leq k_2 \) constants, and \( \varepsilon > 0 \). Given an instance \((J, M)\), a schedule \( S \) is a \((k_1, k_2)\)-relaxed version of \( S_{\mathcal{A}} \) if:

1. jobs with processing time at least \( k_1 \varepsilon \)OPT are assigned exactly as in \( S_{\mathcal{A}} \), and
2. for every machine \( i \in M \), if \( S \) assigns at least one job of size less than \( k_1 \varepsilon \)OPT to \( i \), then \( \ell_i(S) \leq \ell_{\min}(S) + k_2 \varepsilon \)OPT.

The following lemma shows that we can consider relaxed versions of known algorithms or solutions while almost not affecting the approximation factor. This will be helpful to control the migration of small jobs.

**Lemma 2.4.** Let \( \mathcal{A} \) be an \( \alpha \)-approximation, \( \alpha \geq 1 \) constant, \( 1 \leq k_1 \leq k_2 \) constants, \( 0 < \varepsilon < \frac{1}{2k_2 \alpha^2} \), and \((J, M)\) a machine covering instance. Every \((k_1, k_2)\)-relaxed version of \( S_{\mathcal{A}} \) is an \((\alpha + O(\varepsilon))\)-approximate solution.

**Proof.** Suppose by contradiction that there exists a \((k_1, k_2)\)-relaxed version of \( S_{\mathcal{A}} \), say \( S \), which is not \((\alpha + 2k_2 \alpha^2 \varepsilon)\)-approximate. This implies that \( \ell_{\min}(S) \leq \frac{1}{\alpha + 2k_2 \alpha^2} \)OPT \( \leq (\frac{1}{\alpha} - k_2 \varepsilon) \)OPT.

Let \( M_s \) be the set of machines where \( S \) assigns at least one job of size less than \( k_1 \varepsilon \)OPT. Notice that \( M_s \neq \emptyset \) and actually the least loaded machine in \( S \) belongs to \( M_s \), because otherwise \( \ell_{\min}(S_{\mathcal{A}}) = \ell_{\min}(S) < (\frac{1}{\alpha} - k_2 \varepsilon) \)OPT, which contradicts that \( S_{\mathcal{A}} \) is \( \alpha \)-approximate. Since \( S \) and \( S_{\mathcal{A}} \) assign to the same machines jobs of size at least \( k_1 \varepsilon \)OPT, we have that the total processing time of jobs assigned by \( S \) to \( M_s \) is at most \(|M_s|\ell_{\min}(S) + k_2 \varepsilon \)OPT. Thus,

\[
\ell_{\min}(S_{\mathcal{A}}) \leq \min_{i \in M_s} \ell_i(S_{\mathcal{A}}) \leq \ell_{\min}(S) + k_2 \varepsilon \)OPT \( < \frac{1}{\alpha} \)OPT,
\]

which contradicts that \( S_{\mathcal{A}} \) is \( \alpha \)-approximate.

The described results allow us to significantly simplify the analysis of our algorithms. For example, consider LPT, and suppose that at the arrival of jobs with processing time at least some specific value \( h = \Theta(\varepsilon \)OPT), we can construct relaxed versions of solutions constructed by LPT. Dealing with an arriving job of size smaller than \( h \) becomes a simple task since assigning it to the current least loaded machine does not affect the assignment of big jobs, and we can prove that, for suitable constants \( k_1, k_2 \), a \((k_1, k_2)\)-relaxed version of a solution constructed by LPT is maintained that way, thus preserving the approximation guarantee. It is important to remark that this approach is useful only if the algorithm has a robust structure as, in general, the arrival of small jobs does not allow migration of big jobs, and their structure may need to be changed because of these arrivals in order to maintain the approximation factor (see, for example, Section 5).

### 2.2 Rounding Procedure

Another useful tool is rounding the processing times to simplify the instance and create symmetries while affecting the approximation factor only by a negligible value. In our context, the usage of a rounding procedure is motivated by the following lower bound on the migration factor required by LPT on general instances.
**Lemma 2.5.** For any $k \geq 2$, there exists a set $J$ of $4k + 1$ jobs and an extra job $j^*$ not in $J$ such that, for every schedule $S$ constructed using LPT on $2k + 1$ machines, it is not possible to construct a schedule $S'$ using LPT for $J \cup \{j^*\}$ with migration factor less than $m/2$.

**Proof.** Fix a constant $0 < \varepsilon \leq \frac{1}{4k}$. Consider a set $J$ consisting of the following $4k + 1$ jobs: $k + 1$ jobs of size $1$; for each $i \in \{0, \ldots, k - 1\}$, a job of size $\frac{1}{2} + i\varepsilon$ and a job of size $\frac{1}{2} - (i + 1)\varepsilon$, and, finally, $k$ jobs of size $\frac{1}{2} - k\varepsilon \geq \frac{1}{3}$. Assume the jobs in $J$ are sorted non-increasingly by size. There is a unique schedule constructed using LPT for this instance (up to symmetry), which assigns the jobs in the following way (see Figure 1(a)): The $k + 1$ jobs of size 1 to a machine on their own, and for each $i = 1, \ldots, k$, it assigns to machine $k + i$ a job of size $\frac{1}{2} + (k - i - 1)\varepsilon$, a job of size $\frac{1}{2} - (k - i)\varepsilon$, and a job of size $\frac{1}{2} - k\varepsilon$ (since the total load of the first two jobs is $1 - \varepsilon$, the last $k$ jobs must be assigned to these $k$ machines).

Now consider an arriving job $j^*$ of size $\frac{1}{2} + k\varepsilon \leq \frac{3}{4}$. There is a unique schedule constructed using LPT for the new instance (up to relabeling of machines or jobs of equal size), which assigns the jobs in the following way (see Figure 1(b)): it assigns to the first $k + 1$ machines a job of size 1 and a job of size $\frac{1}{2} - k\varepsilon$; to machine $k + 2$ job $p_{j^*}$ and a job of size $\frac{1}{2} - (k - 1)\varepsilon$; for each $i = 2, \ldots, k - 1$ it assigns to machine $k + i + 1$ a job of size $\frac{1}{2} + (k + 1 - i)\varepsilon$ and a job of size $\frac{1}{2} - (k - i)\varepsilon$; and finally to machine $2k + 1$ a job of size $\frac{1}{2} + \varepsilon$ and a job of size $\frac{1}{2}$. (Now the total load of machines $k + 2, \ldots, 2k + 1$ is $1 + \varepsilon$, then the last $k + 1$ jobs must be assigned to the first $k + 1$ machines).

It is not difficult to see that, in the new schedule, every machine has a different subset of jobs assigned to it compared to the original schedule; hence, at least one job must have been migrated per machine. Thus, the migrated total load is at least the load of the smallest $2k + 1$ jobs, which implies that the needed migration factor is at least

\[
\frac{\sum_{i=0}^{2k} p_{(4k+1)-i}}{p_{j^*}} \geq \frac{(2k + 1)(\frac{1}{2} - k\varepsilon)}{\frac{1}{2} + k\varepsilon} \geq m \frac{1/3}{2/3} = \frac{m}{2}.
\]

Now, we proceed to describe in detail the rounding procedure: Let us consider $0 < \varepsilon < 1$ such that $1/\varepsilon \in \mathbb{Z}$. We use the following rounding technique, which is a slight modification of the one presented by Hochbaum and Shmoys in the context of makespan minimization on related machines [10]. For any job $j$, let $e_j \in \mathbb{Z}$ be such that $2e_j^\ell \leq p_j < 2e_j^{\ell+1}$. We round down $p_j$ to the previous number of the form $2e_j^\ell + k\varepsilon 2^{\ell+1}j$ for $k \in \mathbb{N}$; that is, we define $p_j := 2e_j^\ell + \lfloor \frac{p_j - 2e_j^\ell}{2e_j^{\ell+1}} \rfloor 2e_j^{\ell+1}$. Hence, an $\alpha$-approximation algorithm for a rounded instance has an approximation ratio of $\alpha/(1 - \varepsilon) = \alpha + O(\varepsilon)$ for the original instance. From now on, we work exclusively with the rounded processing times.

Consider an upper bound $UB$ on OPT such that $OPT \leq UB \leq 2OPT$. This can be computed using any 2-approximation for the problem, in particular, LPT. Consider the index set

\[
I(UB) := \{i \in \mathbb{Z} : e_{UB} \leq 2^i < UB\} = \{\ell, \ldots, u\}.
\]

We classify jobs as small if $p_j < 2^\ell$, big if $p_j \in [2^\ell, 2^{\ell+1})$, and huge, otherwise. Notice that small jobs have size at most $2e_{UB}$ and huge jobs have size at least $UB$. As we will see, our main difficulty will be given by big jobs, while small and huge jobs will be easy to handle. Notice that in every solution $S$ constructed using LPT, if we ignore small jobs, huge jobs are assigned to a machine on their own, and every machine $i \in M$ without huge jobs has load at most $2UB$. This is because $i$ either has a big job alone, which has size at most $2UB$, or it has load at most $\ell_{\min}(S) + p_j \leq 2\ell_{\min}(S) \leq 2UB$, where $j$ is the smallest job assigned to $i$. Let

\[
\tilde{P} = \{2^i + k\varepsilon 2^i : i \in \{\ell, \ldots, u\}, k \in \{0, 1, \ldots, (1/\varepsilon) - 1\}\}
\]

ACM Transactions on Algorithms, Vol. 16, No. 4, Article 43. Publication date: July 2020.
be the set of all (rounded) processing times that a big job may take. The following lemma highlights the main properties of our rounding procedure.

**Lemma 2.6.** Consider the rounded job sizes $\tilde{p}_j$ for all $j$. Then, it holds that

1. $|\tilde{p}| \in O((1/\varepsilon) \log(1/\varepsilon))$, and
2. for each big and huge job $j$, it holds that $\tilde{p}_j = h \cdot \varepsilon 2^\ell$ for some $h \in \mathbb{N}_0$.

**Proof.** From the definition of $\tilde{p}$, we have that $|\tilde{p}| = \frac{1}{\varepsilon}(u - \ell - 1) - 1$. Since $\ell \geq \log(\varepsilon UB)$ and $u \leq \log(UB)$, then $(u - \ell - 1) \leq \log(\varepsilon)$. Altogether, $|\tilde{p}| \in O((1/\varepsilon) \log(1/\varepsilon))$. Also, if $j$ is a big or huge job, then $\tilde{p}_j = 2^i + k\varepsilon 2^i$ for some $i \geq \ell$ and $k \in \{0, \ldots, \frac{1}{\varepsilon} - 1\}$. We conclude by noticing that $\tilde{p}_j = (\frac{1}{\varepsilon} + k)2^{i-\ell} \cdot \varepsilon 2^\ell = h \cdot \varepsilon 2^\ell$ for $h = (\frac{1}{\varepsilon} + k)2^{i-\ell} \in \mathbb{N}_0$. 

Unlike other standard rounding techniques (e.g., Refs [12] and [18]), the rounded sizes do not depend on OPT (or UB). This avoids possible migrations provoked by new rounded values, greatly simplifying our techniques.

### 3 A SIMPLE (1.7 + $\varepsilon$)-COMPETITIVE ALGORITHM WITH $O(1/\varepsilon)$ MIGRATION

In this section, we will adapt a local search algorithm for machine covering to the online context with migration, using the properties of instances rounded as described in Section 2.2 to bound the migration factor.

In the context of online load balancing with migration, it is a good strategy to look for local search algorithms with good approximation guarantees and efficient running times. The main reason is that the migrated load can be analyzed for each local move separately, and for simplified instances (rounded, for example), the number of local moves required to reach some locally optimal solution is usually a constant. That is the case for two natural neighborhoods used in local search algorithms for load balancing problems: Jump and Swap. Two solutions $S, S'$ are jump-neighbors if they assign the jobs to the same machines (up to relabeling of machines or jobs of equal size) except for at most one job, and swap-neighbors if they assign the jobs to the same machines (up to relabeling of machines or jobs of equal size) except for at most two jobs and, if they differ in exactly two jobs $j_1, j_2$, then they are in swapped machines, i.e., $S(j_1) = S'(j_2)$ and $S(j_2) = S'(j_1)$. The weight of a solution is defined through a two-dimensional vector having the minimum load of the schedule as the first coordinate and the number of non-least loaded machines as the second one. We compare the weight of two solutions lexicographically.\(^1\) In other words, a solution is jump-optimal (respectively, swap-optimal) if the migration of a single job (respectively, the migration of a job or the swapping of two jobs) does not increase the minimum load and, if it maintains the minimum load, then it does not reduce the number of least loaded machines.

**Lemma 3.1.** Given $(\mathcal{J}, \mathcal{M})$ a machine covering instance, a schedule $S$ is jump-optimal if and only if for any machine $i \in \mathcal{M}$ and any job $j \in S^{-1}(i)$, we have that $l_j(S) - p_j \leq \ell_{\min}(S)$.

**Proof.** If $S$ is jump-optimal and there is a job not satisfying the inequality, then moving it to a least loaded machine either increases the minimum load of the schedule or reduces the number of least loaded machines, which is a contradiction.

On the other hand, if $S$ is not jump-optimal, then there is a job $j$ whose migration improves the weight of the solution or, if not, a job whose migration to a least loaded machine decreases the number of least loaded machines. Consider first the case in which moving $j$ from a machine $i$

\(^1\)Just using the minimum load does not lead to good approximation ratios: think, for example, of $m > 2$ machines and $m$ jobs of size 1; it would be swap-optimal to assign all of them to the same machine.
increases the minimum load. This means that the new load of machine \( i, \ell_i(S) - p_j, \) is at least the new minimum load, which is strictly larger than \( \ell_{\text{min}}(S), \) proving the needed inequality. Consider now the case in which moving \( j \) from a machine \( i \) to machine \( i' \) maintains the minimum load while reducing the number of least loaded machines. Then, \( i' \) must have been a non-unique least loaded machine in \( S. \) Furthermore, machine \( i \) cannot become a least loaded machine in the new schedule, since, otherwise, the number of least loaded machines would not change. This means that \( \ell_i(S) - p_j \) is strictly larger than \( \ell_{\text{min}}(S). \) \( \square \)

Chen et al. [3] proved tight bounds for the approximability of jump-optimal solutions. Their result is stated in a game theoretical framework, where jump-optimal solutions are equivalent to pure Nash equilibria for the machine covering game (see, for example, Ref. [19]). In this game, each job is a selfish agent trying to minimize the load of its own machine and the minimum load is the welfare function to be maximized. Through a small modification, these bounds can be generalized to swap-optimal solutions as well (notice that a swap-optimal solution is jump-optimal by definition). We summarize the result in the following theorem, which will be useful for our purposes.

**Theorem 3.2 (Chen et al. [3]).** Any locally optimal solution with respect to jump (respectively, swap) for machine covering is 1.7-approximate. Moreover, there are instances showing that the approximation ratio of jump-(respectively, swap-)optimality is at least 1.7.

### 3.1 Online Jump-Optimality

Using the rounding procedure developed in Section 2.2, jump-optimality can be adapted to the online context using migration factor \( O(\frac{1}{\epsilon}) \). In order to do this, we need an auxiliary algorithm called Push (Algorithm 1) to assign a job \( j \) to a given machine. This procedure inserts a given job to a given machine, and then iteratively removes the jobs that break jump-optimality according to Lemma 3.1, storing them in a special set \( Q \), which is part of its output. This algorithm is the base of the Push neighborhood analyzed by Schuurman and Vredeveld [17].

**ALGORITHM 1:** Push

**Input:** Schedule \( S \) for \((J, M)\), \( i \in M, j \notin J \)

**Output:** \( Q \subseteq J \), schedule \( S' \) for \(((J \cup \{j\}) \setminus Q, M)\)

1: \( Q \leftarrow \emptyset \).
2: \( S' \leftarrow S. \)
3: assign \( j \) to machine \( i \) in \( S' \).
4: for \( k \in S^{-1}(i) \) do
5: if \( \ell_i(S') - \tilde{p}_k > \ell_{\text{min}}(S'), \) then
6: take out \( k \) from \( i \) in \( S' \).
7: \( Q \leftarrow Q \cup \{k\}. \)
8: end if
9: end for
10: return \( Q, S'. \)

Our algorithm, described in detail in Algorithm 2, is called every time a new job \( j^* \) arrives to the system, and receives as input the current solution \( S \) for \((J, M)\), initialized as empty if \( J = \emptyset \). It will output a \((k, 2k)\)-relaxed version of a jump-optimal solution for some \( k \leq 4 \). We use the concept of the list-scheduling algorithm, that refers to assigning jobs iteratively (in any order) to some machine of minimum load. Given a schedule \( S, S_B \) will denote the restriction of schedule \( S \) to big jobs.
The general idea of Algorithm 2 is to first round the instance, and assign the incoming job to a least-loaded machine using Algorithm 1. Jobs removed by Algorithm 1 need to be reassigned, which we do by iteratively applying Algorithm 1 on each one of them that is big until only small jobs are left to be assigned. At each iteration, jump-optimality is preserved in a relaxed way, and as a last step, all the remaining small jobs are reassigned using list-scheduling. Notice that since Algorithm 1 only removes jobs of size strictly smaller than the inserted job, each job is migrated at most once.

**ALGORITHM 2:** Online jump-optimality

**Input:** Instances \((J', M)\) and \((J', M)\) such that \(J' = J \cup \{j^*\}\); a schedule \(S(J, M)\).

1. run LPT on input \(J'\) and let \(\tau\) be the minimum load. Set \(UB \leftarrow 2\tau\). Define \(P, \ell, \) and \(u\) based on this upper bound \(UB\) using Equations (1) and (2).

2. set \(S' \leftarrow S\)

3. if \(\hat{p}_{j^*} < 2^\ell\) then.

4. assign \(j^*\) to a least loaded machine in \(S'\).

5. else

6. set \(Q_B \leftarrow \{j^*\}\). \(\triangleright\) Set with unassigned big jobs.

7. set \(Q_s \leftarrow \emptyset\). \(\triangleright\) Set with unassigned small jobs.

8. while \(Q_B \neq \emptyset\) do

9. let \(j\) be the largest job in \(Q_B\). Set \(Q_B \leftarrow Q_B \setminus \{j\}\).

10. in \(S_B'\), use Push (Algorithm 1) to assign \(j\) to a least loaded machine \(m^*\), obtaining its output set \(Q\). Update \(S_B'\) to be the output solution of this procedure.

11. reassign jobs in \(S'\) such that the assignment of (big) jobs in \(S'\) and \(S_B'\) coincides.

12. while \(m^*\) contains a small job w.r.t. \(UB\) and \(\ell_{m^*}(S') > \ell_{\min}(S') + 2^\ell\) do

13. remove the smallest job in \(S'^{-1}(m^*)\) and add it to \(Q_s\).

14. end while

15. \(Q_B \leftarrow Q_B \cup Q\).

16. end while

17. assign the jobs in \(Q_s\) to \(S'\) using list-scheduling.

18. end if

19. return \(S'\).

**Lemma 3.3.** For any \(h \in \mathbb{R}^+\), Algorithm 2 has a robust structure at level \(h\). Furthermore, Algorithm 2 is \((1.7 + O(\varepsilon))\)-competitive and has polynomial running time.

**Proof.** First of all, Algorithm 2 has, for any \(h \in \mathbb{R}^+\), a robust structure at level \(h\) because each time that Push is called, it moves a total load of jobs smaller than the processing time of the inserted job (otherwise, the machine would not be a least loaded machine), and if the job is small, then nothing is migrated. This also directly implies that the running time of the algorithm is polynomial because every job is migrated at most once, so the while loop is executed only a polynomial number of times.

In order to show that the competitive ratio is \((1.7 + O(\varepsilon))\), we just need to show that the schedule constructed by Algorithm 2 is a \((k_1, k_2)\)-relaxed version of a jump-optimal solution for some constants \(k_1, k_2\). Having that, the result follows from Theorem 3.2 and Lemma 2.4.

Let \(k = \frac{\ell_{\text{OPT}}}{\ell_{\text{OPT}}}\). We will prove that the constructed schedule is a \((k, 2k)\)-relaxed version of a jump-optimal schedule by induction on \(|J|\) (notice that \(k\) depends on \(\text{OPT}'\) and \(\ell\), hence, on the instance \((J, M)\)). The base case when \(J = \emptyset\) is trivial. Let \(\ell^{(1)}\) be the lower bound computed for \(\text{OPT}\) and \(k^{(1)} = \frac{\ell^{(1)}}{\ell_{\text{OPT}}}\), and let us assume that \(S\) is a \((k^{(1)}, 2k^{(1)})\)-relaxed version of some jump-optimal
solution \(S^*\) for \((\mathbb{J}, \mathcal{M})\) (recall that \(OPT \leq OPT'\) and \(\ell^{(1)} \leq \ell\)). This means that \(S\) and \(S^*\) assign to the same machines jobs of size at least \(2^{\ell^{(1)}}\), and machines in \(S\) containing at least one job of size smaller than \(2^{\ell^{(1)}}\) have load at most \(\ell_{\min}(S) + 2 \cdot 2^{\ell^{(1)}}\). Our goal is to prove that the output \(S'\) of Algorithm 2, when run on \((\mathbb{J}, \mathcal{M})\) plus an arriving job \(j^*\), is a \((k, 2k)\)-relaxed version of some jump-optimal solution \(S''\) for \((\mathbb{J} \cup \{j^*\}, \mathcal{M})\).

Notice first that for this \(k\), it holds that big jobs have processing time at least \(ktOPT'\). If \(\tilde{p}_{j^*} < 2^\ell\), it is easy to see that the conditions are fulfilled since it is assigned to a least loaded machine. Assume from now on that \(j^*\) is big. Suppose that we run Algorithm 2 on \(S^*(\mathbb{J}, \mathcal{M})\) and arriving job \(j^*\), getting a solution \(S^*_\text{aux}\). First of all, it is not difficult to see that the minimum load does not decrease when applying Algorithm 2. Thanks to the jump-optimality of \(S^*\), we have that, for every machine \(i\) where no job was assigned using Push and any job \(j\) assigned to \(i\), \(\ell_i(S^*_\text{aux}) - p_j < \ell_{\min}(S^*_\text{aux})\); hence, the jobs breaking jump-optimality in \(S^*_\text{aux}\) can only belong to the remaining machines. In these machines, we either have only big jobs or they have load at most \(\ell_{\min}(S^*_\text{aux}) + 2^\ell\), implying that the jobs breaking jump-optimality are small thanks to Lemma 3.1. If we take out from the solution such jobs and reassign them using Push until no job is left to be assigned (i.e., reassigning also the jobs that are pushed out), we get a jump-optimal solution \(S''\). Since this procedure moves only small jobs (as pushed jobs are always smaller than the assigned job), the assignment of big jobs in \(S'\) and \(S''\) is the same, proving the first part of being a \((k, 2k)\)-relaxed version of some jump-optimal solution.

We will now prove that if a machine has at least one job of size at most \(2^\ell\), then its load is at most \(\ell_{\min}(S') + 2 \cdot 2^\ell\). To this end, we will consider three cases:

- If \(i\) is a machine where no job was assigned using Push and it has a job of size smaller than \(2^{\ell^{(1)}}\), since \(S\) is a \((k^{(1)}, 2k^{(1)})\)-relaxed version of some jump-optimal solution, the load of \(i\) is at most \(\ell_{\min}(S) + 2 \cdot 2^{\ell^{(1)}} \leq \ell_{\min}(S') + 2 \cdot 2^\ell\).
- If \(i\) is a machine where no job was assigned using Push, it has only jobs of size at least \(2^{\ell^{(1)}}\) and has at least one job of size smaller than \(2^\ell\) (implying that \(\ell^{(1)} < \ell\)), since \(S\) is a \((k^{(1)}, 2k^{(1)})\)-relaxed version of some jump-optimal solution \(S^*\), the load of \(i\) is at most \(\ell_{\min}(S) + 2 \cdot 2^{\ell^{(1)}} \leq \ell_{\min}(S^*) + 2^\ell\). From the proof of Lemma 2.4, we have that \(\ell_{\min}(S^*) \leq \ell_{\min}(S') + 2 \cdot 2^{\ell^{(1)}}\). Putting everything together, we have that
  \[
  \ell_i(S') \leq \ell_{\min}(S^*) + 2^\ell \\
  \leq \ell_{\min}(S') + 2 \cdot 2^{\ell^{(1)}} + 2^\ell \\
  \leq \ell_{\min}(S') + 2 \cdot 2^\ell,
  \]
  where the last inequality comes from the fact that \(\ell^{(1)} < \ell\).
- If \(i\) is a machine where some job was assigned using Push and it has at least one job of size smaller than \(2^\ell\), the algorithm enforces its load to be at most \(\ell_{\min}(S^*) + 2^\ell\).

This proves that \(S'\) is a \((k, 2k)\)-relaxed version of some jump-optimal solution, and we conclude the proof by noticing that \(1 \leq k = 2^\ell/(\epsilon OPT') \leq 2^{1UB/(\epsilon OPT')} \leq 4\).

Now, we will bound the migration factor and also construct an instance showing that the analysis of the migration factor is essentially tight.

**Lemma 3.4.** Algorithm 2 has migration factor \(O(1/\epsilon)\).

**Proof.** To analyze the migration factor, we define the migration tree of the algorithm as a node-weighted tree \(G = (V, E)\), where \(V\) is the set of migrated jobs together with the incoming job \(j^* \notin \mathbb{J}\), and the weight of each \(v \in V\) is the processing time of the corresponding job \(\tilde{p}_v\). The tree
is constructed by first adding $j^*$ as root. For each node (job) $v$ in the tree, its children are defined as all the jobs migrated at the insertion of $v$. It is easy to see that this process does not create any loops as each job is migrated at most once. By definition, the leaves of the tree are the jobs not inducing migration; thus, any small job in the tree is a leaf. In the context of local search, the number of nodes in the tree representing big jobs corresponds to the number of iterations of the specific local search procedure.

Let $w_i$ be the total processing time of nodes corresponding to big jobs in level $i$ of the migration tree. Recall that the processing times of these jobs are rounded according to the procedure described in Section 2.2; hence, they belong to the set $\tilde{P} = \{2^j + k r 2^j : i \in \{\ell, \ldots, u\}, k \in \{0, 1, \ldots, (1/\epsilon) - 1\}\}$. We will relabel this set as $\tilde{P} = \{q_1, q_2, \ldots, q_{|\tilde{P}|}\}$, where $q_1 > q_2 > \cdots > q_{|\tilde{P}|}$. Assume that $q_k = 2^y + 2^y$ for some $\kappa \in \{1, \ldots, |\tilde{P}|\}$, $g \in \{\ell, \ldots, u\}$, and $h \in \{0, 1, \ldots, \frac{1}{\epsilon} - 1\}$.

Every time a job $j$ is inserted using Push, the total load of jobs in the output $Q$ of the algorithm is strictly less than $\hat{p}_j$, which means that $w_i$ is strictly decreasing, and also that at each level $i$ of the tree, there are at most $\frac{w_i}{\epsilon}$ nodes corresponding to big jobs. Since the second condition of being a $(k_1, k_2)$-relaxed version (Definition 2.3) of a jump-optimal solution is maintained through the iterations, the small jobs that need to be migrated because of insertion of a big job $j$ have a total load at most $\hat{p}_j + 2^\ell$. This implies that the total load of small jobs at each level $i \geq 1$ of the tree is at most $w_{i-1} + \frac{w_i}{|\tilde{P}|} \cdot 2^\ell = 2 w_{i-1}$; hence, the total processing time of nodes corresponding to small jobs is at most twice the total processing time of nodes corresponding to big jobs. Because of that, from now on, we will assume that the migration tree contains only nodes corresponding to big jobs.

We categorize each level $i \geq 1$ of the migration tree according to the following two cases: if there is a node in level $i - 1$ having at least two children, we say that level $i$ falls in case 1, and it falls in case 2, otherwise. Let $l_1$ (respectively, $l_2$) be the number of levels of the tree falling in case 1 (respectively, case 2). We start by noticing that $l_1 \leq \frac{\hat{p}_j}{2^\ell} \leq 1/\epsilon$: indeed, because of the way the migration tree is constructed, the total weight of the leaves is at most $\hat{p}_j$ (this property is maintained inductively through the executions of Algorithm Push). This implies that, since each big job has processing time at least $2^\ell$, every migration tree has at most $\frac{\hat{p}_j}{2^\ell}$ leaves, which is also an upper bound for the number of nodes that have more than one child in the tree (each one of them induces at least one extra leaf), and hence for $l_1$.

Unfortunately, it might happen that $l_2 > 1/\epsilon$, but we will show that $w_i$ is significantly smaller than $w_{i-1}$ if level $i$ falls in case 2, based on the following claim.

Claim: Let $q_{i_1}, \ldots, q_{i_k} \in \tilde{P}$ such that $\sum_{j=1}^k q_{i_j} \in (q_{s+1}, q_s)$ for some $s \in \{1, \ldots, |\tilde{P}|\}$. Then, $\sum_{j=1}^k q_{i_j} \leq \sum_{j=1}^k q_{i_j} - \frac{2}{3} q_{s+1}$ (assuming $q_{|\tilde{P}|+1} = 0$).

Notice that the claim implies that for a level $i$ falling in case 2, if $w_{i-1} \in (q_{s+1}, q_s)$ for some $s \in \{1, \ldots, |\tilde{P}|\}$, then $w_i \leq w_{i-1} - \frac{2}{3} q_{s+1}$ (this holds because in this case, if algorithm Push inserts a job of size $q_s$, it removes at most one big job of size at most $q_{t+1}$). To compute the total processing time of the nodes in the migration tree, we will bound the total weight of the levels corresponding to each case separately. Since $l_1 \leq 1/\epsilon$ and each level falling in case 1 has total weight at most $\hat{p}_j$, we can bound the total weight of those levels by $\frac{1}{\epsilon} \hat{p}_j$. Let us now relabel the levels of the tree where the second case occurs by just $\{1, 2, \ldots, l_2\}$ (i.e., we ignore the levels falling in case 1). Now, for every $i \in \{1, 2, \ldots, l_2\}$, we can apply the claim four times and show that if $w_{i-1} \in (q_{s+1}, q_s)$ for some $s \in \{1, \ldots, |\tilde{P}|\}$, then $w_{i+3} \leq w_{i-1} - \epsilon q_{s+1} \leq q_s - \epsilon q_{s+1} \leq s_{i+3}$ (as $w_{i+2} < w_{i+1} < w_i < w_{i-1} \leq q_s$); this argument can then be iterated starting at level $i + 3$ with the correct bound $q_{s'} \leq q_{s+1}$. If we argue like this starting with $w_0 \in (q_{k+1}, q_k)$, we can conclude that $\sum_{j=0}^{l_2} w_i \leq 4 \sum_{j=k}^{|\tilde{P}|} q_{i_j}$, which,
machines, is constructed in the following way: Consider the possible processing times $\forall k \in \mathbb{N}$.

There are instances for which Algorithm 2 uses a migration factor of at least $\Omega(\frac{1}{\ell})$.

Proof. Consider an instance with $OPT = 2^u+1$ and $\epsilon \cdot OPT = 2^{\ell}$ for some integers $\ell, u$, and assume for simplicity that $UB = OPT$. This way, $I(UB) = \{\ell, \ldots, u\}$. The instance, consisting of $m \in O\left(\frac{1}{\ell} \log \frac{1}{\epsilon}\right)$ machines, is constructed in the following way: Consider the possible processing times sorted non-increasingly $t_1, \ldots, t_h$. For each $i$ such that $t_i < 2^a$, the schedule has a machine with a job of size $t_i$ assigned, and it is completed with jobs until having load $2^a+1$: if $t_i = 2^k + j \epsilon 2^k$, this can be done by adding a job of size $2^k + (\frac{1}{\ell} - j) \epsilon 2^k$, a job of size $2^k$, and for each $k' = k + 2, \ldots, u$, a job of size $2^{k'}$ (if $i = u - 1$, the machine will not have any of these last jobs). By doing so, the load of the machine is

$$2^i + j \epsilon 2^i + 2^i + \left(\frac{1}{\ell} - k\right) \epsilon 2^i + 2^i + \sum_{i'=i+2}^u 2^{i'} = 2^{i+2} + 2^{u+1} - 2^{i+2} = 2^{u+1}.$$ 

Now, if a job of size $2^a$ arrives to the system, it can be inserted using Push in the machine with the largest job of size less than $2^a$ (i.e., with processing time $2^a-1 + (\frac{1}{\ell} - 1) \epsilon 2^a-1$), taking out such job because it breaks jump-optimality. If Algorithm 2 takes the decision in the same way iteratively, then at least one job of each possible size $t_i < 2^a$ is migrated, being then the total migrated load at least

$$\sum_{i=\ell}^{u-1} \sum_{j=1}^{i-1} 2^i + j \epsilon 2^i = \left(\frac{1}{\ell} - 1\right) \left(2^u - 2^{\ell+1}\right) + \frac{1}{2} \left(\frac{1}{\ell} - 1\right) 2^u \in \Omega\left(\frac{1}{\ell} 2^u\right),$$

and, hence, the migration factor needed for this instance is $\Omega(\frac{1}{\ell})$.

By putting together Lemmas 3.3, 3.4, and 3.5, we can conclude the following result.

Theorem 3.6. Given $\epsilon > 0$, Algorithm 2 is a polynomial time $(1.7 + \epsilon)$-competitive algorithm and uses migration factor $O(1/\epsilon)$. Moreover, there are instances for which this factor is $\Omega(1/\epsilon)$.

4 LPT ONLINE WITH MIGRATION $\tilde{O}(1/\epsilon^3)$

In this section, we present our main contribution, which is an approximate online adaptation of LPT using poly(1/\epsilon) migration factor. In order to analyze it, we will first show some structural properties of the solutions constructed by LPT and how they behave when the instance is perturbed by a new job.

Algorithm 2 presented in Section 3 already gives some of the features and properties that our online version of LPT fulfills. However, now, in the analysis, we will crucially exploit the symmetry of instances rounded according to the procedure described in Section 2.2, in particular, the facts that the load of each machine is a multiple of some fixed value and that the set of possible loads for a machine has size $\text{poly}(1/\epsilon)$ (see Section 4.4 for a discussion about this last property and the fact...
that the usual geometric rounding procedure does not satisfy it). Since LPT takes decisions based solely on the machine loads, having a bounded number of values for them allows us to accurately control the set of machines where the assignment of big jobs can be kept unchanged after the arrival of a big job while maintaining the structure of the solution. Unless stated otherwise, for the rest of this section, machine loads are considered with respect to the rounded processing times $\hat{p}_j$.

### 4.1 Load Monotonicity of LPT

Here, we describe in more detail the useful structural properties of solutions constructed using LPT.

**Definition 4.1.** Given a schedule $S$, its **load profile**, denoted by $\text{load}(S)$, is an $\mathbb{R}_{\geq 0}^m$-vector $(t_1, \ldots, t_m)$ containing the load of each machine sorted so that $t_1 \leq t_2 \leq \cdots \leq t_m$.

The following lemma shows that after the arrival of a job, the load profile of solutions constructed using LPT can only increase. This property only holds if the vector of loads is sorted, as it can be seen in Figure 1. This monotonicity property is essential for our analysis. To show the mentioned property, the following rather technical lemma will help.

**Lemma 4.2.** Let $x, y \in \mathbb{R}_+^n$, $x = (x_1, x_2, \ldots, x_n)$, $y = (y_1, y_2, \ldots, y_n)$ such that $x_1 \leq x_2 \leq \cdots \leq x_n$, $y_1 \leq y_2 \leq \cdots \leq y_n$ and $x \leq y$ coordinate-wise, and $\alpha, \beta \in \mathbb{R}$ such that $\alpha \leq \beta$. If we consider the new vectors defined by replacing $x_i$ by $x_i + \alpha$ in $x$ and $y_i$ by $y_i + \beta$ in $y$ for some $i \in \{1, 2, \ldots, n\}$, and then we sort the coordinates non-decreasingly of the new vectors, obtaining $x'$ and $y'$, then $x' \leq y'$ coordinate-wise.

**Proof.** Let $s$ be the coordinate such that $x'_s = x_1 + \alpha$ and $t$ such that $y'_t = y_t + \beta$. For each coordinate $k < \min(s, t)$ or $k > \max(s, t)$, we have that $x'_k = x_k$ and $y'_k = y_k$, thus satisfying the desired inequality by hypothesis. For the remaining coordinates, we have two cases:

- Assume that $s < t$, and let $k \in \{s, s + 1, \ldots, t - 1\}$. Then, we have that
  
  $$x'_k \leq x'_{k+1} = x_{k+1} \leq y_{k+1} = y'_k,$$
  
  where the first inequality holds due to the monotonicity of the vectors and the second one because of the hypothesis. Similarly, for $k = t$, we have that
  
  $$x'_k = x_k \leq y_k = y'_{k-1} \leq y'_k,$$
  
  where the first inequality follows from the hypothesis.

- Assume that $s \geq t$, and let $k \in \{t, t + 1, \ldots, s\}$. Then,
  
  $$x'_k \leq x'_t \leq y'_t \leq y'_k,$$
  
  where the first and third inequalities follow from the monotonicity of the vectors, and the second one from the fact that $x_1 + \alpha \leq y_t + \beta$.

**Lemma 4.3.** Let $(\mathcal{J}, M)$ be a machine covering instance and $j^* \notin \mathcal{J}$ a job. Then, it holds that $\text{load}(S_{\text{LPT}}(\mathcal{J}, M)) \leq \text{load}(S_{\text{LPT}}(\mathcal{J}', M))$, where the inequality is considered coordinate-wise and $\mathcal{J}' = \mathcal{J} \cup \{j^*\}$.

**Proof.** Let us first relabel the jobs in $\mathcal{J}$ so that $\hat{p}_1 \geq \hat{p}_2 \geq \cdots \geq \hat{p}_n$. To simplify the argument, we assume that both runs of LPT assign jobs in the order given by the labeling above 1, 2, \ldots, $n$, where in the run for $\mathcal{J}'$, the new job $j^*$ is inserted to the list in any position consistent with LPT. We can assume this without loss of generality as different tie breaking do not affect the load profiles.
Consider the set of instances \((J|k, M)\) for \(k = r, \ldots, n\), where \(J|k \subseteq J\) is the set of the \(k\) largest jobs in \(J\), and \(r\) is the maximal index such that \(p_r \leq \hat{p}_r\). Similarly, let \(J'|k = J|k \cup \{j^*\}\) for any \(k \in \{r, \ldots, n\}\). We will show, by induction, that the lemma is true for each pair \((J|k, M)\) and \((J'|k, M)\). The base case \(k = r\) follows easily from Lemma 4.2 since \(S_{LPT}(J|k, M)\) and \(S_{LPT}(J'|k \setminus \{j^*\}, M)\) assign to the same machines all jobs \(\{1, \ldots, r\}\), and adding \(j^*\) to the least loaded machine in \(S_{LPT}(J|k, M)\) (a job of size 0 to the least loaded machine in \(S_{LPT}(J'|k \setminus \{j^*\}, M)\)), and then the inequality holds.

Suppose now that \(\text{load}(S_{LPT}(J|k, M)) \leq \text{load}(S_{LPT}(J'|k, M))\). The fact that the inequality holds for \(k + 1\) is equivalent to the following claim: if job \(k + 1\) is assigned to a least loaded machine in \(S_{LPT}(J|k, M)\) and in \(S_{LPT}(J'|k, M)\), the resulting load profiles satisfy the inequality. This is exactly the statement of Lemma 4.2 with \(\alpha = \beta = \hat{p}_{k+1}\), \(x = \text{load}(S_{LPT}(J|k, M))\), \(y = \text{load}(S_{LPT}(J'|k, M))\) and \(i = 1\).

This lemma together with our rounding procedure allow us to show that the difference (in terms of the Hamming distance) of the load profiles of two consecutive solutions consisting purely of big jobs, is bounded by a small constant. This property will be important to obtain a poly\((1/\varepsilon)\) migration factor and, here, we crucially exploit the fact that the load of the machines is always multiple of a fixed value.

**Lemma 4.4.** Consider two instances \((J, M)\) and \((J', M)\) with \(J' = J \cup \{j^*\}\), where \(J'\) contains only big or huge jobs w.r.t. UB. Then, the vectors load\(S_{LPT}(J, M)\) and load\(S_{LPT}(J', M)\) differ in at most \(\frac{\hat{p}_{j^*}}{\varepsilon^2}\) many coordinates.

**Proof.** Thanks to Lemma 4.3, we have that \(\text{load}(S_{LPT}(J, M)) = (t_1, \ldots, t_m) \leq (t'_1, \ldots, t'_m) = \text{load}(S_{LPT}(J', M))\). Also, if \(t_i < t'_i\) for some \(i\), then \(t'_i \geq t_i + \varepsilon^2 t\) since all values \(t_i, t'_i\) are integer multiples of \(\varepsilon^2 t\) because of Lemma 2.6. Since \(\|\text{load}(S_{LPT}(J', M)) - \text{load}(S_{LPT}(J, M))\|_1 = \hat{p}_{j^*}\), we obtain that the number of coordinates in which the load profiles differ is at most \(\frac{\hat{p}_{j^*}}{\varepsilon^2}\). Finally, recalling that \(j^*\) is big, then \(\hat{p}_{j^*} \leq 2^m \leq \text{UB} \leq 2^\varepsilon t\), and we can bound the number of different coordinates by \(\frac{\hat{p}_{j^*}}{\varepsilon^2} \leq 1/\varepsilon^2\). \(\square\)

### 4.2 Description of Online LPT

Consider two instances \((J, M)\) and \((J', M)\) such that \(J' = J \cup \{j^*\}\), and let OPT and OPT’ be their optimal values, respectively. In what follows, for a given list-scheduling algorithm, we will refer to a tie-breaking rule as a rule that decides a particular machine for assigning a job when faced with multiple least loaded machines. We say that an assignment is an LPT-solution if there is some tie-breaking rule such that LPT yields such assignment. We will compute an upper bound on OPT’ by computing an LPT-solution and duplicating the value of its minimum load. For this upper bound, we compute its respective set \(\hat{P}\) with Equations (1) and (2). In the algorithm, we will label elements in \(\hat{P} = \{q_1, \ldots, q_{|\hat{P}|}\}\) such that \(q_1 > q_2 > \cdots > q_{|\hat{P}|}\). Let \(J_h \subseteq J\) (respectively, \(J'_h \subseteq J'\)) be the set of jobs of size \(q_h\) in \(J\) (respectively, \(J'\)), for \(q_h \in \hat{P}\). Similarly, we define \(J_0\) (respectively, \(J'_0\)) to be the set of jobs in \(J\) (respectively, \(J'\)) of sizes larger than \(q_1\), that is, all huge jobs in \(J\) (respectively, \(J'\)). Also, let \(S_h\) (respectively, \(S'_h\)) be the solution \(S\) (respectively, \(S'\)) restricted to jobs of size \(q_h\) or larger. Finally, \(S_0\) and \(S'_0\) are the respective solutions restricted to jobs in \(J_0\).

In what follows, \(x_+\) denotes the positive part of \(x \in \mathbb{R}\), i.e., \(x_+ = \max\{x, 0\}\). To better understand the algorithm, it is useful to have the following observation in mind.
OBSERVATION 4.5. Consider a solution $S$ for jobs in $J$ and let $K$ be a set of jobs with $J \cap K = \emptyset$, and all jobs in $K$ have the same size $p$. Consider a solution $S_{LS}$ constructed by adding the jobs from $K$ into $S$ using list-scheduling, and let $\lambda = \ell_{\min}(S_{LS})$. Notice that $\lambda$ is independent of the tie-breaking rule used in list-scheduling. Consider any solution $S'$ that is constructed starting from $S$ and adding jobs in $K$ in some arbitrary way. Then, $S'$ corresponds to a solution obtained by adding jobs from $K$ with a list-scheduling procedure (for some tie-breaking rule) if and only if the number of jobs in $K$ added to each machine $i$ is: $\left\lceil \frac{(\lambda - \ell_i(S))_p}{p} \right\rceil$ if $\frac{(\lambda - \ell_i(S))_p}{p}$ is not an integer, and either $\frac{(\lambda - \ell_i(S))_p}{p}$ or $1$ if $\frac{(\lambda - \ell_i(S))_p}{p}$ is a non-negative integer.

Our main procedure is called every time that we get a new job $j^*$ (where $J' = J \cup \{j^*\}$) and receives as input the current solution $S$ for $(J, M)$. If $J = \emptyset$, then $S$ is trivially initialized as empty. The exact description is given in Algorithm 3.

Broadly speaking, the algorithm works in phases $h \in \{0, \ldots, |\tilde{P}|\}$, where for each $h$, it assigns jobs in $J'_h$. First, we assign jobs exactly as in $S_h$ for machines in which the assignment of $S_{h-1}$ and $S'_{h-1}$ coincide. The set of such machines is denoted by $M_{h-1}^\equiv$ and the set of remaining machines is denoted by $M_h^\neq$. As we will see, this is consistent with LPT by the previous observation and Lemma 4.3. The remaining jobs in $J'_h$ are assigned using list-scheduling. Crucially, we will break ties in favor of machines where the assignment of $S_{h-1}$ and $S'_{h-1}$ differ. This is necessary to avoid creating new machines with different assignments. After assigning huge and big jobs, small jobs are added exactly as in $S$ to machines where the assignment of big jobs in $S$ and $S'$ coincides. The rest of small jobs are added greedily. In the last part, the algorithm rebalances small jobs by moving them from machines of load higher than $\ell_i(S') + 2\ell$ to the least loaded machines.

**ALGORITHM 3: Online LPT**

**Input:** Instances $(J, M)$ and $(J', M)$ such that $J' = J \cup \{j^*\}$; a schedule $S(J, M)$.

1. run LPT on input $J'$ and let $\tau$ be the minimum load of the constructed solution. Set $\text{UB} \leftarrow 2\tau$.

2. Define $\tilde{P}$, $\lambda$, and $u$ based on this upper bound $\text{UB}$ using Equations (1) and (2).

3. for $h = 0, 1, \ldots, |\tilde{P}|$ do  
   $\triangleright$ Assignment of big and huge jobs
   4. for each machine $i \in M_{h-1}^\equiv$, assign all jobs in $J_h \cap S^{-1}(i)$ to $i$ in $S'$.
   5. for jobs in $J'_h$ still not assigned in $S'$, apply list-scheduling (with an arbitrary order of jobs). If there is more than one least loaded machine break ties in favor of machines in $M_{h-1}^\equiv$.
   6. define $M_h^\equiv$ as the set of machines $i$ such that $S_{h-1}^{-1}(i) = S_{h-1}^{-1}(i)$ and $M_h^\neq \leftarrow M \setminus M_h^\equiv$.
   7. end for

8. for machines $i \in M_{|\tilde{P}|}^\equiv$ do  
   $\triangleright$ Assignment of small jobs
   9. assign all small jobs w.r.t. to $\text{UB}$ in $J \cap S^{-1}(i)$ to $i$ in $S'$.
   10. end for

11. assign the remaining jobs using list-scheduling.

12. set $\tilde{M}$ to be the set of machines containing a small job w.r.t. $\text{UB}$.

13. while there exists $i \in \tilde{M}$ s.t. $\ell_i(S') > \ell_{\min}(S') + 2\ell$ do
   14. consider a machine $i \in \tilde{M}$ of maximum load. Reassign the smallest job in $S^{-1}_{\tau-1}(i)$ to any least loaded machine.
   15. update $\tilde{M}$ to be the set of machines containing a small job w.r.t. $\text{UB}$.
   16. end while

17. return $S'$.

We can bound the competitive ratio of the algorithm in a very similar way to Lemma 3.3. First, we prove the following auxiliary lemma.
Lemma 4.6. If $S'$ is the output of the algorithm, then $S'_{|P|}$ is an LPT-solution.

Proof. We show the proof inductively. Consider a run of the algorithm with input assignment $S$. If $S$ is empty, then it is clearly an LPT-solution. Otherwise, $S$ is the output of a run of the algorithm. We can assume inductively that $S'_{|P|}$ is an LPT-solution (and, thus, also any restriction of $S'_{|P|}$ to jobs of sizes at least $p$, for any $p \geq 0$). Notice that $UB_0 \leq UB$, by Lemma 4.3, and thus min $P_0 \leq min \bar{P}$ and max $P_0 \leq max \bar{P}$.

We use a second induction to show that for every $h \in \{0, \ldots, |P|\}$, $S'_h$ is an LPT-solution.

To show the base case ($h = 0$), consider jobs in $J'_0$, which are all larger than $UB \geq OPT'$. Hence, there are at most $m$ of them, and the algorithm assigns them each to a different machine (this, again, follows inductively). Thus, the base case holds.

Consider $h \geq 1$ and let us assume that $S'_{h-1}$ is an LPT-solution. Let $S_{LPT,h}$ be an LPT-solution for jobs in $J_0 \cup \ldots \cup J_h$, and, similarly, $S'_{LPT,h}$ for jobs in $J'_0 \cup \ldots \cup J'_h$. First, observe that the load profile vector load($S'_{LPT,h}$) is independent of the tie-breaking rule. Consider the target value $\lambda = \ell_{\min}(S_{LPT,h})$ and $\lambda' = \ell_{\min}(S'_{LPT,h})$. Notice that by Lemma 4.3, $\lambda \leq \lambda'$.

Since $S_{h-1}$ is an LPT-solution, then $S'_h$ is an LPT-solution if jobs in $J'_h$ are added using list-scheduling. By Observation 4.5, the following characterizes this fact: for all machines $i \in M$, the number of jobs assigned in $S'_h$ to $i$ is $\lceil (\lambda' - \ell_i(S'_{h-1}))_+ / q_h \rceil$ if $(\lambda' - \ell_i(S'_{h-1}))_+ / q_h$ is not an integer, and either $(\lambda' - \ell_i(S'_{h-1}))_+ / q_h$ or $(\lambda' - \ell_i(S'_{h-1}))_+ / q_h + 1$ if $(\lambda' - \ell_i(S'_{h-1}))_+ / q_h$ is an integer. Since $\lambda \leq \lambda'$, and $S_h$ is an LPT-solution, then the number of jobs assigned in Step 4 is never more than $\lceil (\lambda' - \ell_i(S'_{h-1}))_+ / q_h \rceil$ if $(\lambda' - \ell_i(S'_{h-1}))_+ / q_h$ is not an integer, and never more than $(\lambda' - \ell_i(S'_{h-1}))_+ / q_h + 1$ if $(\lambda' - \ell_i(S'_{h-1}))_+ / q_h$ is an integer. Hence, after adding jobs in Step 5, we obtain an LPT-solution.

Now, we can conclude about the approximation guarantee of the output.

Lemma 4.7. When considering instances of machine covering such that $|M| = m$, Algorithm 3 is $(\frac{4m-2}{3m-1} + O(\varepsilon))$-competitive.

Proof. We will use the previous lemma to show that $S'$ is a $(k,k)$-relaxed version of $S_{LPT}(J', M)$ for some $k \leq 4$, which is enough to conclude the claim due to Lemma 2.4 and the result from Csirik et al. [4, Theorem 3.5]. Indeed, let $k = 2^\ell / (\varepsilon OPT')$. Then, by the previous lemma, all jobs larger than $k\varepsilon OPT' = 2^\ell$ are assigned with LPT. Also, the while loop at Step 13 ensures that the output of the algorithm is a $(k,k)$-relaxed version of $S_{LPT}(J', M)$. The lemma follows since $k \leq 4$ as shown in Lemma 3.3.

4.3 Bounding the Migration Factor

To analyze the migration factor of the algorithm, we will show that $|M^e_{|P|}|$ is upper bounded by a constant. This will be done inductively by first bounding $|M^e_h \setminus M^e_{h-1}|$ for each $h$ and then using the fact that $|\bar{P}| \in O((1/\varepsilon) log(1/\varepsilon))$. A description of the overall idea can be found in Figure 2.

Let us consider huge jobs w.r.t. UB (i.e., jobs in $J'_0$). Notice that all these jobs are larger than OPT' \geq OPT; hence, in $S'_0$, each one is assigned to some machine as its unique job. The same situation happens in solution $S$ restricted to jobs in $J_0$. Thus, none of these jobs are migrated. Hence, we can assume w.l.o.g. for the sake of the analysis of the migration that all jobs are big or small w.r.t. UB (including $j^*$). Additionally, we can assume that $j^*$ is not small, since, otherwise, there is no migration.

Let $J^e_h$ be the set of jobs assigned by Step 5 to machines in $M^e_{h-1}$. Notice that the jobs in $J^e_h$ correspond to the jobs in $J'_h$ that $S'$ assigns to a machine in $M^e_{h-1}$ but $S$ processes in $M^e_{h-1}$. These are the only jobs that can induce new machines with different assignments in $S$ and $S'$;
Fig. 2. Depiction of a possible situation at the end of iteration \( h - 1 \). The machines on the right side correspond to machines in \( M_{h-1}^\neq \) and, therefore, process the same jobs in \( S_{h-1} \) and \( S'_{h-1} \). Assume, possibly erroneously and just as a thought experiment, that the machines in \( M_{h-1}^\neq \) can be sorted non-decreasingly by load for \( S_{h-1} \) and \( S'_{h-1} \) simultaneously. The two solutions are depicted simultaneously in the picture, where the difference of loads on machines in \( M_{h-1}^\neq \) corresponds to the dashed area. The total dashed load equals to \( \tilde{\lambda}_f \), which is spread in only constantly many machines by Lemma 4.4. When assigning jobs in \( J_h \), the algorithm first assigns a number of jobs to each machine in \( M_{h-1}^\neq \) (Step 4), and then fills machines in \( M_{h-1}^\neq \). Notice that as long as the algorithm does not assign another job to a machine in \( M_{h-1}^\neq \), no new machine will enter \( M_{h-1}^\neq \). On the other hand, the number of such jobs can be bounded by a number proportional to \( \tilde{\lambda}_f \) (and \( 1/\epsilon \)), which then also bounds the number of machines in \( M_{h-1}^\neq \). In reality, however, it is not true that the machines in \( M_{h-1}^\neq \) can be sorted non-decreasingly on the loads for \( S_{h-1} \) and \( S'_{h-1} \) simultaneously. This provokes a number of technical difficulties that we avoid by using a different permutation of machines for each solution and invoking Lemma 4.3.

hence, our strategy will consist of bounding the cardinality of set \( J_h^\neq \) in order to upper bound \( |M_{h-1}^\neq \setminus M_{h-1}^\neq | \). First, we prove two auxiliary lemmas that help to upper bound \( |J_h^\neq | \). Also, recall that \( \lambda = \ell_{\min}(S_{\text{LPT},h}) \) and \( \lambda' = \ell_{\min}(S'_{\text{LPT},h}) \).

**Lemma 4.8.** Assume that \( J_h^\neq \neq \emptyset \). For each machine \( i \in M_{h-1}^\neq \), if \( \lambda - \ell_i(S_{h-1}) \geq 0 \), then solution \( S_h' \) assigns to \( i \) at least \( \left\lfloor \frac{(\lambda - \ell_i(S_{h-1}'))_i}{q_h} \right\rfloor + 1 \) many jobs from \( J_h \).

**Proof.** Due to Observation 4.5, the number of jobs assigned to machine \( i \) is \( \left\lfloor \frac{(\lambda - \ell_i(S_{h-1}))_i}{q_h} \right\rfloor \) if \( \frac{(\lambda - \ell_i(S_{h-1}))_i}{q_h} \) is not an integer, and either \( \frac{(\lambda - \ell_i(S_{h-1}))_i}{q_h} \) or \( \frac{(\lambda - \ell_i(S_{h-1}))_i}{q_h} + 1 \) if \( \frac{(\lambda - \ell_i(S_{h-1}))_i}{q_h} \) is a non-negative integer. In the first case, we have that \( \left\lfloor \frac{(\lambda - \ell_i(S_{h-1}'))_i}{q_h} \right\rfloor \geq \left\lfloor \frac{(\lambda - \ell_i(S_{h-1}'))_i}{q_h} \right\rfloor + 1 \) since \( \lambda' \geq \lambda \) thanks to Lemma 4.3, and the claim holds. If, instead, \( \frac{(\lambda - \ell_i(S_{h-1}))_i}{q_h} \) is integral, we distinguish two cases:

- If \( \lambda = \lambda' \), then suppose for the sake of contradiction that the algorithm assigns \( \frac{(\lambda - \ell_i(S_{h-1}'))_i}{q_h} \) many jobs to \( i \). In this case, the tie-breaking rule does not assign jobs to \( M_{h-1}^\neq \); hence, \( J_h^\neq = \emptyset \), which is a contradiction.
- If \( \lambda > \lambda' \), then \( \lambda > \ell_i(S_{h-1}') \), and the number of jobs in \( J_h \) assigned to machine \( i \) is at least \( \left\lfloor \frac{(\lambda - \ell_i(S_{h-1}'))_i}{q_h} \right\rfloor \geq \left\lfloor \frac{(\lambda - \ell_i(S_{h-1}'))_i}{q_h} \right\rfloor + 1 \); hence, the claim holds. \( \square \)

**Lemma 4.9.** Let \( x, y \in \mathbb{R}_{+}^n \), \( x = (x_1, x_2, \ldots, x_n) \), \( y = (y_1, y_2, \ldots, y_n) \) such that \( x_1 \leq x_2 \leq \cdots \leq x_n \), \( y_1 \leq y_2 \leq \cdots \leq y_n \), and \( x \leq y \) coordinate-wise. Assume that \( x_i = y_i \) for some indices \( i, j \). If \( x_{-i} \) denotes the \((n-1)\)-dimensional vector obtained by removing the \( i\)-th entry of \( x \), and \( y_{-j} \) is the vector obtained by removing the \( j\)-th entry of \( y \), then \( x_{-i} \leq y_{-j} \).

**Proof.** Notice first that if \( i = j \), then the result is a direct consequence of Lemma 4.2—by taking \( \alpha = \beta = -x_i \) and coordinate \( i \), we get new vectors \( \tilde{x} \) and \( \tilde{y} \) satisfying \( \tilde{x} \leq \tilde{y} \) and \( \tilde{x}_i = \tilde{y}_i = 0 \).
Hence, we can conclude that \( x_{-i} \leq y_{-j} \) because \( x_{-i} \) (respectively, \( y_{-j} \)) corresponds to the last \( n - 1 \) coordinates of \( \tilde{y} \) (respectively, \( \tilde{y} \)).

We now distinguish two cases: if \( i < j \), we have that \( y_j = x_i \leq y_j \); hence, \( x_k = y_j \) for every \( k = i, i + 1, \ldots, j \). This implies that \( x_{-i} = x_{-i(i+1)} = \cdots = x_{-j-1} \) and then we can conclude that \( x_{-j} \leq y_{-j} \) by applying the previous observation for \( x_{-j} \) and \( y_{-j} \). On the other hand, if \( j < i \), we define vector \( z \) equal to \( y \) but replacing coordinates \( j, j + 1, \ldots, i \) by \( y_j \). It is not difficult to see that \( x \leq y \) coordinate-wise, and also \( z_{-j} = z_{-(j+1)} = \cdots = z_i \). If we apply the first observation for \( x \) and \( z \) using coordinate \( i \), we have that \( x_{-i} \leq z_{-i} \), and applying it to \( z \) and \( y \) using coordinate \( j \), we get that \( z_{-j} \leq y_{-j} \). Merging both inequalities and using the fact that \( z_{-i} = z_{-j} \), we conclude that \( x_{-i} \leq y_{-j} \).

**Lemma 4.10.** It holds that \( |\mathcal{J}_h^z| \in O\left( \frac{\eta^p}{r^d} \right) \).

**Proof.** We will assume that \( \mathcal{J}_h^z \neq \emptyset \), as otherwise the claim is trivial. Assume, w.l.o.g., that \( M_{h-1} = \{1, \ldots, m'\} \) and \( \ell_1(S'_{h-1}) \leq \ell_2(S'_{h-1}) \leq \cdots \leq \ell_{m'}(S'_{h-1}) \). Consider also a permutation \( \sigma : M_{h-1} \to M_{h-1} \) such that \( \ell_{\sigma(1)}(S_{h-1}) \leq \ell_{\sigma(2)}(S_{h-1}) \leq \cdots \leq \ell_{\sigma(m')} (S_{h-1}) \). By Lemma 4.3, the sorted vector of loads (over all machines) of solution \( S_{h-1} \) is upper bounded by the sorted vector of loads of \( S_{h-1} \). Applying Lemma 4.9 iteratively to remove machines in \( M_{h-1} \) one by one (which have the same assignment in both solutions), it holds that \( \ell_{i}(S_{h-1}) \leq \ell_{i}(S'_{h-1}) \) for all \( i \in M_{h-1} \).

Let us consider the sets

\[
T_- = \{ i \in M_{h-1} : \ell_i(S'_{h-1}) \leq \lambda \}, \quad \text{and} \quad T_+ = \{ i \in M_{h-1} : \ell_{\sigma(i)}(S_{h-1}) \leq \lambda \ \text{and} \ \ell_i(S'_{h-1}) > \lambda \}.
\]

Lemma 4.8 implies that the total number of jobs from \( \mathcal{J}_h' \) assigned by \( S'_{h} \) to machines in \( M_{h-1} \) is at least

\[
\sum_{i \in T_-} \left( \frac{\ell_i(S'_{h-1})}{q_h} + 1 \right) = \sum_{i \in T_-} \left( \frac{\ell_{\sigma(i)}(S_{h-1})}{q_h} \right) + \sum_{i \in T_-} \left( \frac{\ell_i(S'_{h-1})}{q_h} \right) + \sum_{i \in T_+} \left( \frac{\ell_{\sigma(i)}(S_{h-1})}{q_h} \right) - \sum_{i \in T_+} \left( \frac{\ell_i(S'_{h-1})}{q_h} \right)
\]

Notice that the set \( T_- \cup T_+ \) contains all indices \( i \in M_{h-1} \) such that \( \ell_{\sigma(i)}(S_{h-1}) \leq \lambda \). Hence, the first sum in the last expression upper bounds the number of jobs in \( \mathcal{J}_h \) that solution \( S \) assigns to machines in \( M_{h-1} \). That way, since \( |\mathcal{J}_h' \setminus \mathcal{J}_h| \leq 1 \), it holds that

\[
|\mathcal{J}_h^z| \leq 1 + \sum_{i \in T_-} \left( \frac{\ell_i(S'_{h-1})}{q_h} \right) + \sum_{i \in T_+} \left( \frac{\ell_i(S'_{h-1})}{q_h} \right) - \sum_{i \in T_+} \left( \frac{\ell_i(S'_{h-1})}{q_h} \right)
\]

ACM Transactions on Algorithms, Vol. 16, No. 4, Article 43. Publication date: July 2020.
Let us now consider $T_h = \{ i \in M_{h-1}^\# : \ell_{\sigma(i)}(S_{h-1}) \neq \ell_i(S_{h-1}^{'}) \}$. Thus, the last expression is at most
\[
|\mathcal{J}_h^{\#}| \leq 1 + |T_+| + \sum_{i \in (T_+ \cup T_0) \cap T_s} \left( \frac{(\lambda - \ell_{\sigma(i)}(S_{h-1}))_+}{q_h} - \frac{1}{q_h} \right)
\]
\[
\leq 1 + |T_+| + \sum_{i \in T_s} \left( \frac{(\lambda - \ell_{\sigma(i)}(S_{h-1}))_+}{q_h} - \frac{(\lambda - \ell_i(S_{h-1}^{'}) + 1}{q_h} \right)
\]
\[
\leq 1 + |T_+| + |T_+| + \sum_{i \in T_s} \frac{\ell_i(S_{h-1}^{'}) - \ell_{\sigma(i)}(S_{h-1})}{q_h}
\]
\[
\leq 1 + 2|T_+| + \frac{\bar{p}}{q_h}.
\]

Also, Lemma 4.4 can be applied and, thus, $|T_+| \leq \frac{\bar{p}}{2\varepsilon^2}$. The lemma finally follows since $q_h \geq 2^\ell$ by definition.

As mentioned before, jobs in $\mathcal{J}_h^{\#}$ are the only jobs assigned in a given iteration $h$ that can cause one new machine to have different assignments in $S_h$ and $S_h'$. Thus, $|M_h^\# \setminus M_{h-1}^\#| \leq |\mathcal{J}_h^{\#}|$, and the following lemma holds.

**Lemma 4.11.** For all $h \in \{1, \ldots, |\hat{P}|\}$, it holds that $|M_h^\# \setminus M_{h-1}^\#| \in O(\frac{\bar{p}}{\varepsilon^2})$.

Putting all the discussed ideas together, we can prove the following result.

**Theorem 4.12.** When considering instances of machine covering such that $|M| = m$, Algorithm Online LPT is a polynomial time $(\frac{\log m - 2}{3m - 1} + O(\varepsilon))$-competitive algorithm with $O((1/\varepsilon^2) \log(1/\varepsilon))$ migration factor.

**Proof.** We first argue that the algorithm runs in polynomial time. Indeed, it suffices to show that the algorithm enters the while loop in Step 13 a polynomial number of times. This follows easily as the quantity $\ell_{\min}(S')$ is non-decreasing; hence, a job can be reassigned to a least loaded machine at most once. Notice that the competitive ratio of the algorithm follows from Lemma 4.7.

Let us now bound the migration factor. We do this in two steps. First, consider solution $S'$ before entering Step 13. We first bound the volume of jobs migrated between $S$ and $S'$, and then bound the total volume of jobs reassigned in the while loop in Step 13.

For the first bound, by Lemma 4.11 and since $M_{h-1}^\# = \emptyset$, it holds that $|M_{h}^\#| \leq |\hat{P}| \cdot O(\frac{\bar{p}}{\varepsilon^2}) \leq O((1/\varepsilon) \log(1/\varepsilon)) \cdot O(\frac{\bar{p}}{\varepsilon^2})$. The load of migrated jobs in $S_{h-1}^\#$ is upper bounded by $\sum_{i \in M_{h}^\#} \ell_i(S_{h-1}^{'}) \leq |M_{h}^\#| \max_{i \in M_{h}^\#} \ell_i(S_{h-1}^{'})$. On the other hand, since we are assuming (w.l.o.g.) that there is no huge job, the total load of each machine is at most $2\bar{u}B$ as argued in Section 2.2. We conclude that the total load of migrated big jobs is at most $2\bar{u}B \cdot |M_{h}^\#| = \bar{u}B \cdot O((1/\varepsilon) \log(1/\varepsilon)) \cdot O(\frac{\bar{p}}{\varepsilon^2})$. Finally, notice that migrated small jobs (before entering Step 13) are the ones assigned to machines in $M_{h}^\#$ by $S$. Since $S$ is the output of Online LPT, then the total load of these jobs is at most $|S_{h-1}^{'}) + 2^\ell \cdot M_{h}^\# \leq 2\bar{u}B \cdot |M_{h}^\#| \leq \bar{u}B \cdot O((1/\varepsilon) \log(1/\varepsilon)) \cdot O(\frac{\bar{p}}{\varepsilon^2})$. We conclude that the total load migrated is at most $\bar{u}B \cdot O((1/\varepsilon) \log(1/\varepsilon)) \cdot O(\frac{\bar{p}}{\varepsilon^2})$.

It remains to bound the volume migrated in the while loop of Step 13. For this, we will show the following claim.

**Claim:** Let $S'$ be the solution constructed before entering Step 13. Then, all reassigned jobs in the while loop, except possibly the one reassigned last, are assigned to a machine in $M_{h-1}^\#$ by $S'$.

ACM Transactions on Algorithms, Vol. 16, No. 4, Article 43. Publication date: July 2020.
Assume the claim holds, and let us consider the solution $S'$ as returned by the algorithm. Then, the total volume of reassigned jobs is bounded by $|M^e_{|\bar{P}|}| \max_{i \in M^e_{|\bar{P}|}} \ell_i(S')$. Since, by construction, the load of a machine that processes a job smaller than $2^\ell$ is at most $\ell_{\min}(S') + 2^\ell \leq 2UB$, the total volume migrated will be at most $UB \cdot O((1/\varepsilon) \log(1/\varepsilon) \frac{\bar{P}^*}{\varepsilon^2})$ as before. Hence, the migration factor is upper bounded by

$$O \left( \frac{UB}{\bar{P}^*} \right) \frac{(1/\varepsilon) \log(1/\varepsilon) \frac{\bar{P}^*}{\varepsilon^2}}{\varepsilon} = O((1/\varepsilon^3) \log(1/\varepsilon)).$$

To show the claim, consider $S'$ before entering Step 11 together with the corresponding set $\bar{M}$ of machines that process some small job. Since $S$ is the output of Online LPT, then the difference between the maximum and minimum loads of machines in $\bar{M} \cap M^e_{|\bar{P}|}$ for solution $S'$ is at most $2^\ell$. We call this property (P1). Also, notice that $\bar{M} \cap M^e_{|\bar{P}|} = \emptyset$; hence, the maximum load difference of two machines in this set is at most $2^\ell$, vacuously. We refer to this property as (P2). Notice that (P1) and (P2) hold iteratively throughout the later steps of the algorithm. Additionally, if some job is assigned to a machine $M^e_{|\bar{P}|}$ in Step 11, the algorithm does not enter the while loop, and we are done. Otherwise, the minimum load is achieved at $M^e_{|\bar{P}|}$. Hence, if there is a job migrated from a machine in $M^e_{|\bar{P}|}$ to $M^e_{|\bar{P}|}$, then the algorithm finishes. The claim follows. □

4.4 A Note on Geometric vs. Arithmetic Rounding

In most of the previous work (e.g., Refs [16] and [18]), the processing times of the jobs are rounded using a geometric rounding procedure, which, in general, works as follows: Given $\varepsilon > 0$, the rounded processing time of job $j$ is $(1 + \varepsilon)^{e_j}$, where $e_j \in \mathbb{Z}$ satisfies that $(1 + \varepsilon)^{e_j} \leq p_j < (1 + \varepsilon)^{e_j+1}$. One of the main reasons to use our rounding procedure to multiples of $\varepsilon 2^\ell$ instead of this geometric rounding is because the same arguments used in this work cannot be applied to geometric rounded instances. It is crucial in the analysis that the number of possible loads is poly $(1/\varepsilon)$, while for geometric rounded instances that is not true as the following lemmas show.

**Lemma 4.13.** Let $\varepsilon \in \mathbb{Q}_+, \varepsilon < 1$. Given a machine covering instance $(J, M)$, let $\tilde{J}$ be the set of jobs obtained by rounding, geometrically, jobs with processing time $p_j \in [\varepsilon OPT, OPT]$. If $C_1, C_2 \subseteq \tilde{J}$ are two different multi-sets of jobs with processing times at least $\varepsilon OPT$ such that $\sum_{j \in C_i} p_j \in [\varepsilon OPT, OPT]$, $i = 1, 2$, then $\sum_{j \in C_1} p_j \neq \sum_{j \in C_2} p_j$.

**Proof.** Assume w.l.o.g. $OPT = 1$. Hence, the possible processing times are $(1 + \varepsilon)^{i}$, with $i$ such that $\varepsilon \leq (1 + \varepsilon)^{i} \leq 1$ (a finite family of such possible values). Suppose, by contradiction, that there are two different non-empty multi-sets $C_1, C_2$ with the same total load, and assume they are minimal, i.e., that there is no other pair of non-empty multi-sets with the same total load but with smaller total load. For $k = 1, 2$, let $C_k(j)$ be the number of jobs with processing time $(1 + \varepsilon)^{i}$ in set $C_k$. Since the pair $C_1, C_2$ is minimal, we have that $C_1(j) = 0$ or $C_2(j) = 0$ for every $j$. $C_1$ and $C_2$ having the same total load means that

$$\sum_{j=-k}^{0} C_1(j)(1 + \varepsilon)^{i} = \sum_{j=-k}^{0} C_2(j)(1 + \varepsilon)^{i},$$

where $k = -\lceil \log_{1+\varepsilon}(\varepsilon) \rceil$. This last equality can be rephrased as the existence of a non-zero polynomial $p(x) = b_0 + b_1 x + \cdots + b_k x^k$, with $|b_j| \in |C_1(j), C_2(j)|$ (i.e., with integer coefficients), that has $(1 + \varepsilon)$ as one of its roots. Since $\varepsilon = \frac{c}{d} > 0$ for some co-primes $c$ and $d$, then $1 + \varepsilon = \frac{c + d}{d}$. Dividing $p(x)$ by $(dx - (c + d))$ leads to a polynomial $q(x) = a_0 + a_1 x + \cdots + a_{k-1} x^{k-1}$ which, thanks to
Gauss lemma, has integer coefficients, too. Let \( b_i \) be the first coefficient of \( p \) different from zero. Then,

\[
|b_i| = |(c + d)a_i + da_{i-1}| = \left| \left(c + \frac{c}{\varepsilon}\right)a_i + \frac{c}{\varepsilon}a_{i-1}\right| > \frac{1}{\varepsilon},
\]

implying, since the size of each job is at least \( \varepsilon \), that the total load of the multi-sets is at least \( b_i \varepsilon > 1 \), which is a contradiction. \( \square \)

**Lemma 4.14.** Given \( 0 < \varepsilon < 1 \), the number of different multi-sets of jobs with processing time at least \( \varepsilon \text{OPT} \) with total load at most \( \text{OPT} \) for a geometrically rounded instance is \( 2^{\Omega(\frac{1}{\varepsilon})} \).

**Proof.** Let \( u = \lfloor \log_2 \text{OPT} \rfloor \) and \( \ell = \lfloor \log_2 (\varepsilon \text{OPT}) \rfloor \). We will give a lower bound on the number of different sets with total load \( 2^u \) when the jobs are rounded to powers of \( 2 \), which implies that for \( 0 < \varepsilon < 1 \), the same bound holds for processing times rounded to powers of \( (1 + \varepsilon) \). Let \( C_i \) be the number of different multi-sets with total load \( 2^{\ell+i} \). This number is characterized by the recurrence

\[
C_0 = 1 \\
C_{i+1} = 1 + \frac{C_i(C_i + 1)}{2}.
\]

This last term comes from the fact that a multi-set with total load \( 2^{\ell+i+1} \) can be constructed using only one job of size \( 2^{\ell+i+1} \), or merging two multi-sets of size \( 2^{\ell+i} \) (there are \( \binom{C_i}{2} \) + \( C_i \) \( \frac{C_i(C_i + 1)}{2} \) such pairs).

Since recurrence \( a_0 = 1, a_1 = \frac{a_1^a}{2^2} \) satisfies \( a_i \geq 2^a \), we conclude that \( C_{u-\varepsilon} \geq 2^{\log \frac{a}{2}} \geq 2^{\Omega(\frac{1}{\varepsilon})}. \) \( \square \)

Because of these two lemmas, if we use geometrically rounded instances, we cannot make sure that, when a new job arrives to the system, the load profile changes only at poly(1/\( \varepsilon \)) coordinates as it was proved in Lemma 4.4, since, in this case, there are \( 2^{\Omega(1/\varepsilon)} \)-many possible different loads.

## 5 AN IMPROVED LOWER BOUND FOR THE COMPETITIVE RATIO WITH CONSTANT MIGRATION FACTOR

In opposition to online makespan minimization with migration, where a competitive ratio arbitrarily close to one can be achieved with a constant migration factor [16], the online machine covering problem does not allow it. Until now, the best lower bound known for this ratio is \( \frac{26}{17} \) [18], which we now improve to \( \frac{17}{16} \) using similar ideas.

**Lemma 5.1.** For any \( \varepsilon > 0 \), there is no \( \left(\frac{17}{16} - \varepsilon\right) \)-competitive algorithm with constant migration factor for the online machine covering problem with migration.

**Proof.** Consider an instance consisting of three machines and six jobs of sizes \( p_1 = p_2 = p_3 = 2, p_4 = p_5 = 3 \), and \( p_6 = \frac{80}{17} \). It is easy to see that the optimal solution is given by Figure 3 (left). Moreover, there is no other \( \left(\frac{17}{16} - \varepsilon\right) \)-approximate solution (up to symmetry).

Suppose, by contradiction, that there exists a \( \left(\frac{17}{16} - \varepsilon\right) \)-competitive algorithm with constant migration factor \( C \). While processing the above instance, the algorithm must construct the optimal solution depicted in Figure 3 (left). Consider now that jobs with processing time smaller than \( 1/C \) arrive to the system, with total processing time \( \frac{26}{17} \). Since the migration factor is \( C \), none of the six previous jobs can be migrated; thus, the best minimum load we can obtain is \( \frac{96}{17} \), while the optimal solution is six, as shown in Figure 3 (right). We conclude by noting that \( \frac{6}{96/17} = \frac{17}{16}. \) \( \square \)

Notice that the instance reaching the lower bound crucially depends on the arrival of jobs with arbitrarily small processing times. These kinds of jobs are, in fact, the problematic ones because
Fig. 3. Left: Unique \((17/16)\)-approximate solution before the arrival of small jobs. Right: Unique \((17/16)\)-approximate solution after small jobs.

under the assumption that, at each iteration, the incoming job is big enough (has processing time at least \(\epsilon \cdot \text{OPT}\)), there is a robust PTAS with constant migration factor \([18]\).

REFERENCES

[1] Y. Azar and L. Epstein. 1998. On-line machine covering. J. Sched. 1 (1998), 67–77.
[2] S. Berndt, K. Jansen, and K. Klein. 2018. Fully dynamic bin packing revisited. Math. Program. (2018).
[3] Xu Jin, Leah Epstein, Elena Kleiman, and Rob van Stee. 2013. Maximizing the minimum load: The cost of selfishness. Theor. Comput. Sci. 482 (2013), 9–19.
[4] J. Csisrîk, H. Kellerer, and G. Woeginger. 1992. The exact LPT-bound for maximizing the minimum completion time. Oper. Res. Lett. 11 (1992), 281–287.
[5] B. Dueermeiyer, D. Friesen, and M. Langston. 1982. Scheduling to maximize the minimum processor finish time in a multiprocessor system. SIJADM 3 (1982), 190–196.
[6] L. Epstein and A. Levin. 2009. A robust APTAS for the classical bin packing problem. Math. Program. 119 (2009), 33–49.
[7] L. Epstein and A. Levin. 2014. Robust algorithms for preemptive scheduling. Algorithmica 69 (2014), 26–57.
[8] A. Frangioni, E. Necciari, and M. Scutellà. 2004. A multi-exchange neighborhood for minimum makespan parallel machine scheduling problems. J. Comb. Optim. 8 (2004), 195–220.
[9] A. Gu, A. Gupta, and A. Kumar. 2016. The power of deferral: Maintaining a constant-competitive Steiner tree online. SIAM J. Comput. 45 (2016), 1–28.
[10] D. Hochbaum and D. Shmoys. 1988. A polynomial approximation scheme for scheduling on uniform processors: using the dual approximation approach. SIAM J. Comput. 17 (1988), 539–551.
[11] K. Jansen and K. Klein. 2013. A robust AFPTAS for online bin packing with polynomial migration. In ICALP 2013. 589–600.
[12] K. Jansen, K. Klein, and J. Verschae. 2016. Closing the gap for makespan scheduling via sparsification techniques. In ICALP 2016. 1–13.
[13] J. Lacki, J. Oświeja, M. Pilipczuk, P. Sankowski, and A. Zych. 2015. The power of dynamic distance oracles: Efficient dynamic algorithms for the steiner tree. In STOC 2015. 11–20.
[14] N. Megow, M. Skutella, J. Verschae, and A. Wiese. 2016. The power of recourse for online MST and TSP. SIAM J. Comput. 45 (2016), 859–880.
[15] D. Recalde, C. Rutten, P. Schuurman, and T. Vredveld. 2010. Local search performance guarantees for restricted related parallel machine scheduling. LATIN 2010 (2010), 108–119.
[16] P. Sanders, N. Sivadasan, and M. Skutella. 2009. Online scheduling with bounded migration. Math. Oper. Res. 34 (2009), 481–498.
[17] P. Schuurman and T. Vredveld. 2007. Performance guarantees of local search for multiprocessor scheduling. INFORMS J. Comput. 19 (2007), 52–63.
[18] M. Skutella and J. Verschae. 2016. Robust polynomial-time approximation schemes for parallel machine scheduling with job arrivals and departures. Math. Oper. Res. 41 (2016), 991–1021.
[19] B. Vöcking. 2007. Selfish load balancing. In Algorithmic Game Theory. 517–542.
[20] G. Woeginger. 1997. A polynomial-time approximation scheme for maximizing the minimum machine completion time. Oper. Res. Lett. 20 (1997), 149–154.

Received April 2019; revised April 2020; accepted April 2020