Symmetry Exploitation for Online Machine Covering with Bounded Migration

Waldo Gálvez
IDSIA, USI-SUPSI
Lugano, Switzerland
waldo@idsia.ch

José A. Soto
Departamento de Ingeniería Matemática & CMM, Universidad de Chile
Santiago, Chile
jsoto@dim.uchile.cl

José Verschae
Facultad de Matemáticas & Escuela de Ingeniería, Pontificia Universidad Católica de Chile
Santiago, Chile
jverschae@uc.cl

Abstract

Online models that allow recourse are highly effective in situations where classical 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.

By rounding the processing times, which yields 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 $\tilde{O}(1/\varepsilon^3)$. 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.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms, Theory of computation → Online algorithms

Keywords and phrases Machine Covering, Bounded Migration, Online, Scheduling, LPT

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.32

Related Version A full version of the paper is available at [9], `https://arxiv.org/abs/1612.01829`.

Funding This work was partially supported by SNSF Grant APXNET 200021_159697/1 and CONICYT-Chile through projects FONDECYT 1181527 and 1181180, PCI PII 20150140 and PIA AFB170001.

26th Annual European Symposium on Algorithms (ESA 2018).
Editors: Yossi Azar, Hannah Bast, and Grzegorz Herman; Article No. 32; pp. 32:1–32:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

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) [21]. A well studied algorithm for this problem is the *Largest Processing Time First* rule (LPT), that 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.

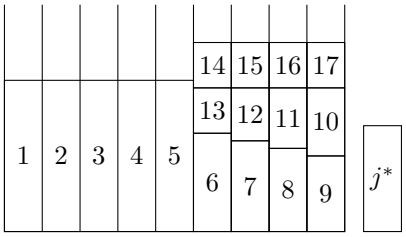
In the online setting jobs arrive one after another, and at the moment of an arrival, we must decide on a machine to assign the arriving job. This natural problem does not admit a constant competitive ratio. Deterministically, the best possible competitive ratio is m [21], while randomization allows for a $\tilde{O}(\sqrt{m})$-competitive algorithm, which is the best possible up to logarithmic factors [1].

Dynamic model. These negative facts motivate the study of a relaxed online scenario 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 βp_j. The value β 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 trade-off between the migration factor β 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) [17].

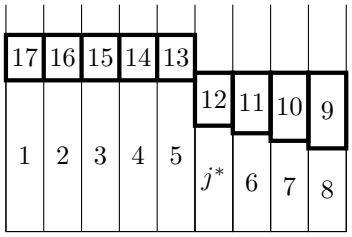
Local search and migration. The local search method has been extensively used to tackle different hard combinatorial problems, and it is closely related to online algorithms where recourse is allowed. This comes from the fact that simple local search neighborhoods allow to get considerably improved solutions while having accurate control over the recourse actions needed, and in some cases even a bounded number of local moves leads to substantially improved solutions (see [15, 10, 14] for examples in network design problems).

Related Work. Sanders et al. [17] develop online algorithms for load balancing problems with migration. For the makespan minimization objective, where the aim is to minimize the maximum load, they give a $(1+\varepsilon)$-competitive algorithm with $2^{\tilde{O}(1/\varepsilon)}$. A mayor open problem in this area is to determine whether a migration factor of $\text{poly}(1/\varepsilon)$ is achievable.

The landscape for the machine covering problem is somewhat different. Sanders et al. [17] 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 [19] 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 very small, not allowing to migrate other jobs. This motivated



(a) LPT for the original instance and arriving job j^*.



(b) LPT for the new instance. Thick items correspond to migrated jobs.

Figure 1 $\Omega(m)$ migration factor needed to maintain LPT at the arrival of j^*.

the study of an amortized version, called *reassignment cost model*, where they develop a $(1+\varepsilon)$-competitive algorithm using a constant reassignment factor. They also show that if all arriving jobs are larger than $\varepsilon \cdot \mathrm{OPT}$, then there is a $(1+\varepsilon)$-competitive algorithm with constant migration factor.

Similar migration models have been studied for other packing and covering problems. For example, Epstein & Levin [6] design a $(1+\varepsilon)$-competitive algorithm for the online bin packing problem using a migration factor of $2^{\tilde{O}(1/\varepsilon^2)}$, which was improved later by Jansen & Klein [12] to $\mathrm{poly}(1/\varepsilon)$ migration factor, and then further refined by Berndt et al. [2]. Also, for makespan minimization with preemption and other objectives, Epstein & Levin [7] design a best-possible online algorithm using a migration factor of $\left(1-\frac{1}{m}\right)$.

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 [18], makespan minimization on restricted related machines [16], and also multi-exchange neighborhoods for makespan minimization on identical parallel machines [8]. For the case of machine covering, Chen et al. [3] study 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 $(4/3+\varepsilon)$-competitive algorithm using $\mathrm{poly}(1/\varepsilon)$ migration factor. This is 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 well studied algorithm in the offline context, directly running this algorithm in each time step in the online context yields an unbounded migration factor; see Figure 1 for an illustrative example.

To overcome this barrier, we first adapt a less standard procedure to round processing times in the online framework. The rounding reduces the possible number of sizes of jobs larger than $\Omega(\varepsilon\mathrm{OPT})$ (where OPT is the offline optimum value) to $\tilde{O}(1/\varepsilon)$ many numbers, and furthermore these values are multiples of a common number $g \in \Theta(\varepsilon^2\mathrm{OPT})$. 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. [13]), our rounding is well suited for online algorithms and helps simplifying 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 and a greedy assignment for small jobs on top of that. 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 constant many possible processing times for large jobs.

In the second part of the paper 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 time step. 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 time step) is constant. This finally yields a migration factor that only grows polynomially in $1/\varepsilon$. Finally, we give a lower bound of 17/16 for the best competitive ratio achievable by an algorithm with constant migration, improving the bound on [19].

Due to space constraints, we defer most of the proofs to the full version [9].

2 Preliminaries

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

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

2.1 Algorithms with robust structure

An important fact used in the robust PTAS for makespan minimization from Sanders et al. [17] 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. [19] or [9]. A way to avoid this inconvenience is to develop algorithms that are oblivious to the arrival of small jobs, that is, algorithms where the assignment of big jobs is unaffected by arriving small job.

▶ **Definition 1.** Let $h\in\mathbb{R}_+$. An algorithm $\mathcal{A}$ has **robust structure at level h** if, for any instance $(\mathcal{J},\mathcal{M})$ and $j^*\notin\mathcal{J}$ such that $p_{j^*}<h$, $\mathcal{S}_{\mathcal{A}}(\mathcal{J},\mathcal{M})$ and $\mathcal{S}_{\mathcal{A}}(\mathcal{J}\cup\{j^*\},\mathcal{M})$ assign to the same machines all the jobs in $\mathcal{J}$ 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.

▶ Remark. For any $h\in\mathbb{R}_+$, LPT has robust structure at level h.

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.** Let $\mathcal{A}$ be an α-approximation algorithm for the machine covering problem, with α constant, $k_1,k_2\in\mathbb{R}_+$ constants, $1\le k_1\le k_2$ and $\varepsilon>0$. Given a machine covering instance $(\mathcal{J},\mathcal{M})$, a schedule $\mathcal{S}$ is a **(k_1,k_2)-relaxed version of $\mathcal{S}_{\mathcal{A}}$** if:

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

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 3.** *Let $\mathcal{A}$ be an α-approximation, $\alpha \geq 1$ constant, $k_1, k_2 \in \mathbb{R}_+$ constants, $1 \leq k_1 \leq k_2$, $0 < \varepsilon < \frac{1}{2k_2\alpha}$ and $(\mathcal{J}, \mathcal{M})$ a machine covering instance. Every (k_1, k_2)-relaxed version of $\mathcal{S}_\mathcal{A}$ is an $(\alpha + O(\varepsilon))$-approximate solution.*

The described results allow us to significantly simplify the analysis of the designed algorithms. For example, consider LPT and suppose that at the arrival of jobs with processing time at least some specific value $h = \Theta(\varepsilon\text{OPT})$ 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, almost preserving then its approximation ratio. It is important to remark that this approach is useful only if the algorithm has 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.

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. 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 [11]. For any job j, let $e_j \in \mathbb{Z}$ be such that $2^{e_j} \leq p_j < 2^{e_j+1}$. We then round down p_j to the previous number of the form $2^{e_j} + k\varepsilon 2^{e_j}$ for $k \in \mathbb{N}$, that is, we define $\tilde{p}_j := 2^{e_j} + \left\lfloor \frac{p_j - 2^{e_j}}{\varepsilon 2^{e_j}} \right\rfloor \varepsilon 2^{e_j}$.

Observe that $p_j \geq \tilde{p}_j \geq p_j - \varepsilon 2^{e_j} \geq (1-\varepsilon)p_j$. Hence, an α-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 $\text{OPT} \leq \text{UB} \leq 2\text{OPT}$. This can be computed by any 2-approximation for the problem such as LPT. Consider the index set

$$\tilde{I}(\text{UB}) := \left\{ i \in \mathbb{Z} : \varepsilon\text{UB} \leq 2^i < \text{UB} \right\} = \{\ell, \ldots, u\}. \tag{1}$$

We classify jobs as *small* if $\tilde{p}_j < 2^\ell$, *big* if $\tilde{p}_j \in [2^\ell, 2^{u+1})$, and *huge* otherwise. Notice that small jobs have size at most $2\varepsilon\text{UB}$ and huge jobs have size at least UB. As we will see, our main difficulty will be given by big jobs; small and huge jobs are easy to handle. Notice that in every solution $\mathcal{S}$ constructed using LPT, if we ignore small jobs, huge jobs are assigned to a machine on their own and every machine $i \in \mathcal{M}$ 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 $\ell_{\min}(\mathcal{S}) + \tilde{p}_j \leq 2\ell_{\min}(\mathcal{S}) \leq 2\text{UB}$, where j is the smallest job assigned to i. Let

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

be the set of all (rounded) processing times that a big job may take. The next lemma highlights the main properties of our rounding procedure.

▶ **Lemma 4.** *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 $h \in \mathbb{N}_0$.*

Unlike other standard rounding techniques (e.g. [19, 13]), 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 corresponds to the sum of the migrated jobs in each local move, and for simplified instances (rounded, for example) the number of local moves until a locally optimal solution is found 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 $\mathcal{S}, \mathcal{S}'$ 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., $\mathcal{S}(j_1) = \mathcal{S}'(j_2)$ and $\mathcal{S}(j_2) = \mathcal{S}'(j_1)$. The *weight* of a solution is defined through a two-dimensional vector having the minimum load of the schedule as first coordinate and the number of non-least loaded machines as 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 (resp. 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. The following lemma characterizes jump-optimal solutions for machine covering.

▶ **Lemma 5.** *Given $(\mathcal{J}, \mathcal{M})$ a machine covering instance, a schedule $\mathcal{S}$ is jump-optimal if and only if for any machine $i \in \mathcal{M}$ and any job $j \in \mathcal{S}^{-1}(i)$, we have that $\ell_i(\mathcal{S}) - p_j \leq \ell_{\min}(\mathcal{S})$.*

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 [20]). 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.

[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 is swap-optimal to assign all of them to the same machine.

Algorithm 1 Online jump-optimality.

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

run LPT on input $\mathcal{J}'$ and let τ be the minimum load. Set UB $\leftarrow 2\tau$. Define $\tilde{P}, \ell$, and u based on this upper bound UB using (1) and (2).
set $\mathcal{S}' \leftarrow \mathcal{S}$
if $\tilde{p}_{j^*} < 2^\ell$ **then**. ▷ `Arriving job is small.`
    assign j^* to a least loaded machine in $\mathcal{S}'$.
else
    set $Q_B \leftarrow \{j^*\}$. ▷ `Set with unassigned big jobs.`
    set $Q_s \leftarrow \emptyset$. ▷ `Set with unassigned small jobs.`
    while $Q_B \neq \emptyset$ **do**
        let j be the largest job in Q_B. Set $Q_B \leftarrow Q_B \setminus \{j\}$.
        in $\mathcal{S}'_B$, use Push to assign j to a least loaded machine m^*, obtaining its output set Q. Update $\mathcal{S}'_B$ to be the output solution of this procedure.
        reassign jobs in $\mathcal{S}'$ such that the assignment of (big) jobs in $\mathcal{S}'$ and $\mathcal{S}'_B$ coincides.
        while m^* contains a small job w.r.t. UB and $\ell_{m^*}(\mathcal{S}') > \ell_{\min}(\mathcal{S}') + 2^\ell$ **do**
            remove the smallest job in ${\mathcal{S}'}^{-1}(m^*)$ and add it to Q_s.
        end while
        $Q_B \leftarrow Q_B \cup Q$.
    end while
    assign the jobs in Q_s to $\mathcal{S}'$ using list-scheduling.
end if
return $\mathcal{S}'$.

▶ **Theorem 6** (from [3]). *Any locally optimal solution with respect to Jump (resp. Swap) for Machine Covering is* 1.7*-approximate. Moreover, there are instances showing that the approximation ratio of jump-(resp. swap-)optimality is at least* 1.7*.*

3.1 Online jump-optimality.

Using the rounding procedure from Section 2.2, jump-optimality can be adapted to the online context using a migration factor of $O\left(\frac{1}{\varepsilon}\right)$. Our algorithm, described in detail in Algorithm 1, is called every time a new job j^* arrives to the system, and receives as input the current solution $\mathcal{S}$ for $(\mathcal{J}, \mathcal{M})$, initialized as empty if $\mathcal{J} = \emptyset$. It will output a (k, k)-relaxed version of a jump-optimal solution for some $k \leq 4$. We use the concept of a *list-scheduling* algorithm, that refers to assigning jobs iteratively (in any order) to some machine of minimum load. Given a schedule $\mathcal{S}$, $\mathcal{S}_B$ denotes the restriction of schedule $\mathcal{S}$ to big jobs.

The general idea of Algorithm 1 is to first round the instance, and assign the incoming job to a least loaded machine using an auxiliary algorithm called *Push* (see Algorithm 2). Push assigns a given job j into a given machine i and then iteratively removes the jobs in i that break jump-optimality according to Lemma 5, storing them in a set Q which is part of the output. Jobs removed by Push need to be reassigned, which we do by iteratively applying Push on each one of them which is big to assign them to the current least loaded machine 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 unassigned small jobs are reassigned using list-scheduling. Notice that, since Push only removes jobs of size strictly smaller than the inserted job, each job is migrated at most once.

Algorithm 2 Push.

Input: Schedule $\mathcal{S}$ for $(\mathcal{J}, \mathcal{M})$, $i \in \mathcal{M}$, $j \notin \mathcal{J}$
Output: $Q \subseteq \mathcal{J}$, schedule $\mathcal{S}'$ for $((\mathcal{J} \cup \{j\}) \setminus Q, \mathcal{M})$

$Q \leftarrow \emptyset$.
$\mathcal{S}' \leftarrow \mathcal{S}$.
assign j to machine i in $\mathcal{S}'$.
for $k \in \mathcal{S}^{-1}(i)$ **do**
    if $\ell_i(\mathcal{S}') - \tilde{p}_k > \ell_{\min}(\mathcal{S}')$ **then**
        take out k from i in $\mathcal{S}'$.
        $Q \leftarrow Q \cup \{k\}$.
    end if
end for
return Q, $\mathcal{S}'$.

▶ **Lemma 7.** *For any $h \in \mathbb{R}^+$, Algorithm 1 has robust structure at level h. Furthermore, Algorithm 1 is $(1.7 + O(\varepsilon))$-competitive and has polynomial running time.*

Proof idea. Robust structure of Algorithm 1 comes from the fact that Push removes jobs that are only smaller than the inserted job. We can then show that our solution is a $(k, 2k)$-relaxed version of a jump-optimal solution for $k = 2^\ell/(\varepsilon \mathrm{OPT}') \leq 4$, and we can conclude the first part of the result by using Theorem 6 and Lemma 3. Polynomial running time is implied by the fact that each job is migrated at most once. ◀

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 $j^* \notin \mathcal{J}$, 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, and thus any small job in the tree is a leaf. In the context of local search, the number of nodes in the tree corresponds to the number of iterations of the specific local search procedure. By analyzing the migration tree level by level, and together with the already discussed ideas, we can show the following result.

▶ **Lemma 8.** *Algorithm 1 uses migration factor $O((1/\varepsilon)\log(1/\varepsilon))$.*

Proof idea. Let w_i be the total processing time of jobs in level i of the migration tree. Every time a job j is inserted using Push, the total load of removed jobs in Q is strictly less than $\tilde{p}_j$, which means that w_i is strictly decreasing. Since w_i is strictly decreasing and jobs of size at most 2^ℓ do not induce migration, the tree has at most $|\tilde{P}| \in O((1/\varepsilon)\log(1/\varepsilon))$ levels, each of them having total load at most $\tilde{p}_{j^*}$. This implies that the total load of migrated big jobs is at most $O((1/\varepsilon)\log(1/\varepsilon)\tilde{p}_{j^*})$ and hence the migration factor is at most $O((1/\varepsilon)\log(1/\varepsilon))$. ◀

The analysis of the migration factor can be further refined to get a tight bound of $O(1/\varepsilon)$. The details can be found in the full version [9].

▶ **Theorem 9.** *Given $\varepsilon > 0$, Algorithm 1 is a polynomial time $(1.7+\varepsilon)$-competitive algorithm with migration factor $O(1/\varepsilon)$. Moreover, there are instances for which this factor is $\Omega(1/\varepsilon)$.*

4 LPT online with migration $\tilde{O}(1/\varepsilon^3)$.

In this section we present our main contribution which is an approximate online adaptation of LPT using poly$(1/\varepsilon)$ 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 1 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 fact that the load of each machine is a multiple of some fixed value. 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 $\tilde{p}_j$.

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

▶ **Definition 10.** Given a schedule $\mathcal{S}$, its **load profile**, denoted by $\text{load}(\mathcal{S})$, is an $\mathbb{R}^m_{\geq 0}$-vector $(t_1, \ldots, t_m)$ containing the load of each machine sorted so that $t_1 \leq t_2 \leq \ldots \leq t_m$.

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.

▶ **Lemma 11.** *Let $(\mathcal{J}, \mathcal{M})$ be a machine covering instance and $j^* \notin \mathcal{J}$ a job. Then, it holds that $\text{load}(\mathcal{S}_{\text{LPT}}(\mathcal{J}, \mathcal{M})) \leq \text{load}(\mathcal{S}_{\text{LPT}}(\mathcal{J}', \mathcal{M}))$, where the inequality is considered coordinate-wise and $\mathcal{J}' = \mathcal{J} \cup \{j^*\}$.*

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 12.** *Consider two instances $(\mathcal{J}, \mathcal{M})$ and $(\mathcal{J}', \mathcal{M})$ with $\mathcal{J}' = \mathcal{J} \cup \{j^*\}$, where $\mathcal{J}'$ contains only big or huge jobs w.r.t* UB. *Then the vectors $\text{load}(\mathcal{S}_{\text{LPT}}(\mathcal{J}, \mathcal{M}))$ and $\text{load}(\mathcal{S}_{\text{LPT}}(\mathcal{J}', \mathcal{M}))$ differ in at most $\frac{\tilde{p}_{j^*}}{\varepsilon 2^\ell} \in O(1/\varepsilon^2)$ many coordinates.*

Proof. Due to Lemma 11, we have that $\text{load}(\mathcal{S}_{\text{LPT}}(\mathcal{J}, \mathcal{M})) = (t_1, \ldots, t_m) \leq (t'_1, \ldots, t'_m) = \text{load}(\mathcal{S}_{\text{LPT}}(\mathcal{J}', \mathcal{M}))$. Also, if $t_i < t'_i$ for some i, then $t'_i \geq t_i + \varepsilon 2^\ell$ since all values $t_j, t_{j'}$ are integer multiples of $\varepsilon 2^\ell$ because of Lemma 4. Since $||\text{load}(\mathcal{S}_{\text{LPT}}(\mathcal{J}', \mathcal{M})) - \text{load}(\mathcal{S}_{\text{LPT}}(\mathcal{J}, \mathcal{M}))||_1 = \tilde{p}_{j^*}$, we obtain that the number of coordinates in which the load profiles differ is at most $\frac{\tilde{p}_{j^*}}{\varepsilon 2^\ell}$. Finally, recalling that j^* is big, then $\tilde{p}_{j^*} \leq 2^u \leq \text{UB} \leq 2^\ell/\varepsilon$, and we can bound the number of different coordinates by $\frac{\tilde{p}_{j^*}}{\varepsilon 2^\ell} \leq 1/\varepsilon^2$. ◀

Description of Online LPT. Consider two instances $(\mathcal{J}, \mathcal{M})$ and $(\mathcal{J}', \mathcal{M})$ such that $\mathcal{J}' = \mathcal{J} \cup \{j^*\}$, 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 UB on OPT' by computing an LPT-solution and duplicating the value of its minimum load. For this upper bound, we compute its respective set $\tilde{P}$ with (1) and (2). In the algorithm, we will label elements in $\tilde{P} = \{q_1, \dots, q_{|\tilde{P}|}\}$ such that $q_1 > q_2 > \cdots > q_{|\tilde{P}|}$. Let $\mathcal{J}_h \subseteq \mathcal{J}$ (respectively $\mathcal{J}'_h \subseteq \mathcal{J}'$) be the set of jobs of size q_h in $\mathcal{J}$ (respectively $\mathcal{J}'$), for $q_h \in \tilde{P}$. Similarly, we define $\mathcal{J}_0$ (resp. $\mathcal{J}'_0$) to be the set of jobs in $\mathcal{J}$ (resp. $\mathcal{J}'$) of sizes larger than q_1, that is, all huge jobs in $\mathcal{J}$ (resp. $\mathcal{J}'$). Also, let $\mathcal{S}_h$ (resp. $\mathcal{S}'_h$) be the solution $\mathcal{S}$ (resp. $\mathcal{S}'$) restricted to jobs of size q_h or larger. Finally, $\mathcal{S}_0$ and $\mathcal{S}'_0$ are the respective solutions restricted to jobs in $\mathcal{J}_0$.

In what follows, x_+ denotes the positive part of $x \in \mathbb{R}$, i.e., $x_+ = \max\{x, 0\}$. To understand the algorithm, it is useful to have the following observation in mind.

▶ **Observation 13.** *Consider a solution $\mathcal{S}$ for jobs in $\mathcal{J}$ and let $\mathcal{K}$ be a set of jobs with $\mathcal{J} \cap \mathcal{K} = \emptyset$ and all jobs in $\mathcal{K}$ have the same size p. Consider a solution $\mathcal{S}_{LS}$ constructed by adding the jobs from $\mathcal{K}$ in $\mathcal{S}$ using list-scheduling, and let $\lambda = \ell_{\min}(\mathcal{S}_{LS})$. Notice that λ is independent of the tie-breaking rule used in list-scheduling. Consider any solution $\mathcal{S}'$ that is constructed starting from $\mathcal{S}$ and adding jobs in $\mathcal{K}$ in some arbitrary way. Then, $\mathcal{S}'$ corresponds to a solution obtained by adding jobs from $\mathcal{K}$ with a list-scheduling procedure (for some tie-breaking rule) if and only if the number of jobs in $\mathcal{K}$ added to each machine i is: (i) $\left\lceil \frac{(\lambda - \ell_i(\mathcal{S}))_+}{p} \right\rceil$ if $\frac{(\lambda - \ell_i(\mathcal{S}))_+}{p}$ is not an integer, and either $\frac{(\lambda - \ell_i(\mathcal{S}))_+}{p}$ or $\frac{(\lambda - \ell_i(\mathcal{S}))_+}{p} + 1$ if $\frac{(\lambda - \ell_i(\mathcal{S}))_+}{p}$ is a non-negative integer.*

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

Broadly speaking, the algorithm works in phases $h \in \{0, \dots, |\tilde{P}|\}$, where for each h it assigns jobs in $\mathcal{J}'_h$. First, we assign jobs exactly as in $\mathcal{S}_h$ for machines in which the assignment of $\mathcal{S}_{h-1}$ and $\mathcal{S}'_{h-1}$ coincide. The set of such machines is denoted by $\mathcal{M}^{=}_{h-1}$ and the set of remaining machines is denoted by $\mathcal{M}^{\neq}_{h-1}$. As we will see, this is consistent with LPT by the previous observation and Lemma 11. The remaining jobs in $\mathcal{J}'_h$ are assigned using list-scheduling. Crucially, we will break ties in favor of machines where the assignment of $\mathcal{S}_{h-1}$ and $\mathcal{S}'_{h-1}$ 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 $\mathcal{S}$ in machines where the assignment of big jobs in $\mathcal{S}$ and $\mathcal{S}'$ 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 $\ell_i(\mathcal{S}') + 2^{\ell}$ to the least loaded machines.

We can prove the following lemma in a very similar way to Lemma 7.

▶ **Lemma 14.** *Algorithm 3 is $(4/3 + O(\varepsilon))$-competitive.*

Bounding the migration factor. To analyze the migration factor of the algorithm, we will show that $|\mathcal{M}^{\neq}_{|\tilde{P}|}|$ is upper bounded by a constant. This will be done inductively by first bounding $|\mathcal{M}^{\neq}_{h} \setminus \mathcal{M}^{\neq}_{h-1}|$ for each h and then using the fact that $|\tilde{P}| \in O((1/\varepsilon)\log(1/\varepsilon))$. 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 $\mathcal{J}'_0$). Notice that all these jobs are larger than $\mathrm{OPT}' \geq \mathrm{OPT}$, and hence in $\mathcal{S}'_0$ each one is assigned alone to one machine. The same situation happens in solution $\mathcal{S}$ restricted to jobs in $\mathcal{J}_0$. 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.

Algorithm 3 Online LPT.

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

1: run LPT on input $\mathcal{J}'$ and let τ be the minimum load of the constructed solution. Set UB $\leftarrow 2\tau$. Define $\tilde{P}, \ell$, and u based on this upper bound UB using (1) and (2).
2: set $\mathcal{M}^{=}_{-1} \leftarrow \mathcal{M}$ and $\mathcal{M}^{\neq}_{-1} \leftarrow \emptyset$.
3: **for** $h = 0, 1, \ldots, |\tilde{P}|$ **do** ▷ `Assignment of big and huge jobs`
4: for each machine $i \in \mathcal{M}^{=}_{h-1}$, assign all jobs in $\mathcal{J}_h \cap \mathcal{S}^{-1}(i)$ to i in $\mathcal{S}'$.
5: for jobs in $\mathcal{J}'_h$ still not assigned in $\mathcal{S}'$, apply list-scheduling (with an arbitrary order of jobs). If there is more than one least loaded machine break ties in favor of $\mathcal{M}^{\neq}_{h-1}$.
6: define $\mathcal{M}^{=}_h$ as the set of machines i such that $\mathcal{S}_h^{-1}(i) = \mathcal{S}_h'^{-1}(i)$ and $\mathcal{M}^{\neq}_h \leftarrow \mathcal{M} \setminus \mathcal{M}^{=}_h$.
7: **end for**
8: **for** machines $i \in \mathcal{M}^{=}_{|\tilde{P}|}$ **do** ▷ `Assignment of small jobs`
9: assign all small jobs w.r.t to UB in $\mathcal{J} \cap \mathcal{S}^{-1}(i)$ to i in $\mathcal{S}'$.
10: **end for**
11: assign the remaining jobs using list-scheduling.
12: set $\overline{\mathcal{M}}$ to be the set of machines containing a small job w.r.t UB.
13: **while** there exists $i \in \overline{\mathcal{M}}$ s.t. $\ell_i(\mathcal{S}') > \ell_{\min}(\mathcal{S}') + 2^\ell$ **do**
14: consider a machine $i \in \overline{\mathcal{M}}$ of maximum load. Reassign the smallest job in $\mathcal{S}'^{-1}(i)$ to any least loaded machine.
15: update $\overline{\mathcal{M}}$ to be the set of machines containing a small job w.r.t UB.
16: **end while**
17: **return** $\mathcal{S}'$.

Let $\mathcal{J}^{=}_h$ be the set of jobs assigned by Step 5 to machines in $\mathcal{M}^{=}_{h-1}$. Notice that the jobs in $\mathcal{J}^{=}_h$ correspond to the jobs in $\mathcal{J}'_h$ that $\mathcal{S}'$ assigns to a machine in $\mathcal{M}^{=}_{h-1}$ but $\mathcal{S}$ processes in $\mathcal{M}^{\neq}_{h-1}$. The next lemma is the main technical contribution of this section.

▶ **Lemma 15.** *For all $h \in \{1, \ldots, |\tilde{P}|\}$ it holds that $|\mathcal{M}^{\neq}_h \setminus \mathcal{M}^{\neq}_{h-1}| \in O(\frac{\tilde{p}_{j^*}}{\varepsilon 2^\ell})$.*

The strategy to prove this lemma is first to show that $|\mathcal{J}^{=}_h| \in O(\frac{\tilde{p}_{j^*}}{\varepsilon 2^\ell})$; this is the main difficulty and for the proof we use lemmas 11 and 12. Having this, since 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 $\mathcal{S}_h$ and $\mathcal{S}'_h$, then $|\mathcal{M}^{\neq}_h \setminus \mathcal{M}^{\neq}_{h-1}| \leq |\mathcal{J}^{=}_h|$ and the lemma holds.

Let $\mathcal{S}_{\mathrm{LPT},h}$ be an LPT-solution for jobs in $\mathcal{J}_0 \cup \ldots \cup \mathcal{J}_h$, and similarly $\mathcal{S}'_{\mathrm{LPT},h}$ for jobs in $\mathcal{J}'_0 \cup \ldots \cup \mathcal{J}'_h$. Let us fix $h \geq 1$ and consider the target values $\lambda = \ell_{\min}(\mathcal{S}_{\mathrm{LPT},h})$ and $\lambda' = \ell_{\min}(\mathcal{S}'_{\mathrm{LPT},h})$. Notice that by Lemma 11, $\lambda \leq \lambda'$. In order to bound $|\mathcal{J}^{=}_h|$, we first show in the following lemma that, if a job is actually assigned by Step 5 to some machine in $\mathcal{M}^{=}_{h-1}$, then many jobs from the stage must be assigned to machines in $\mathcal{M}^{\neq}_{h-1}$.

▶ **Lemma 16.** *Assume that $\mathcal{J}^{=}_h \neq \emptyset$. For each machine $i \in \mathcal{M}^{\neq}_{h-1}$, if $\lambda - \ell_i(\mathcal{S}'_{h-1}) \geq 0$ solution $\mathcal{S}'_h$ assigns to i at least $\left\lfloor \frac{(\lambda - \ell_i(\mathcal{S}'_{h-1}))_+}{q_h} \right\rfloor + 1$ many jobs from $\mathcal{J}_h$.*

Now we can sketch the proof that $|\mathcal{J}^{=}_h| \in O(\frac{\tilde{p}_{j^*}}{\varepsilon 2^\ell})$ (a detailed proof can be found in [9]).

▶ **Lemma 17.** *It holds that $|\mathcal{J}^{=}_h| \in O(\frac{\tilde{p}_{j^*}}{\varepsilon 2^\ell})$.*

Proof Sketch. Assume w.l.o.g. that $\mathcal{M}^{\neq}_{h-1} = \{1, \ldots, m'\}$ and that $\ell_1(\mathcal{S}'_{h-1}) \leq \ell_2(\mathcal{S}'_{h-1}) \leq \cdots \leq \ell_{m'}(\mathcal{S}'_{h-1})$. Consider also a permutation $\sigma : \mathcal{M}^{\neq}_{h-1} \to \mathcal{M}^{\neq}_{h-1}$ such that $\ell_{\sigma(1)}(\mathcal{S}_{h-1}) \leq$

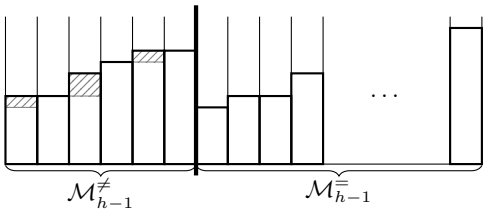


Figure 2 Depiction of a possible situation at the end of iteration $h-1$. The machines on the right side correspond to machines in $\mathcal{M}^{=}_{h-1}$ and therefore process the same jobs in $\mathcal{S}_{h-1}$ and $\mathcal{S}'_{h-1}$. Assume, possibly erroneously and just as a thought experiment, that the machines in $\mathcal{M}^{\neq}_{h-1}$ can be sorted non-decreasingly by load for $\mathcal{S}_{h-1}$ and $\mathcal{S}'_{h-1}$ simultaneously. The two solutions are depicted simultaneously in the picture, where the difference of loads on machines in $\mathcal{M}^{\neq}_{h-1}$ corresponds to the dashed area. The total dashed load equals to $\tilde{p}_{j^*}$, which is spread in only constantly many machines by Lemma 12. When assigning jobs in $\mathcal{J}_h$, the algorithm first assigns a number of jobs to each machine in $\mathcal{M}^{=}_{h-1}$ (Step 4), and then fills machines in $\mathcal{M}^{\neq}_{h-1}$. Notice that while the algorithm does not assign another job to a machine in $\mathcal{M}^{=}_{h-1}$, no new machine will enter $\mathcal{M}^{\neq}_{h} \setminus \mathcal{M}^{\neq}_{h-1}$. On the other hand, the number of such jobs can be bounded by a number proportional to $\tilde{p}_{j^*}$ (and $1/\varepsilon$), which then also bounds the number of machines in $\mathcal{M}^{\neq}_{h} \setminus \mathcal{M}^{\neq}_{h-1}$. In reality, however, it is not true that the machines in $\mathcal{M}^{\neq}_{h-1}$ can be sorted non-decreasingly on the loads for $\mathcal{S}_{h-1}$ and $\mathcal{S}'_{h-1}$ simultaneously. This provokes a number of technical difficulties that we avoid by using a different permutation of machines for each solution and invoking Lemma 11.

$\ell_{\sigma(2)}(\mathcal{S}_{h-1}) \leq \cdots \leq \ell_{\sigma(m')}(\mathcal{S}_{h-1})$. By using Lemma 11, we can show that $\ell_{\sigma(i)}(\mathcal{S}_{h-1}) \leq \ell_i(\mathcal{S}'_{h-1})$ for all $i \in \mathcal{M}^{\neq}_{h-1}$. Let us consider sets

$$\begin{aligned} T_- &= \{i \in \mathcal{M}^{\neq}_{h-1} : \ell_i(\mathcal{S}'_{h-1}) \leq \lambda\}, \text{ and} \\ T_+ &= \{i \in \mathcal{M}^{\neq}_{h-1} : \ell_{\sigma(i)}(\mathcal{S}_{h-1}) \leq \lambda \text{ and } \ell_i(\mathcal{S}'_{h-1}) > \lambda\}. \end{aligned}$$

Lemma 16 implies that the total number of jobs from $\mathcal{J}'_h$ assigned by $\mathcal{S}'_h$ to machines in $\mathcal{M}^{\neq}_{h-1}$ is at least

$$\begin{aligned} &\sum_{i\in T_-} \left(\left\lfloor \tfrac{(\lambda-\ell_i(\mathcal{S}'_{h-1}))_+}{q_h} \right\rfloor + 1 \right) = \sum_{i\in T_-\cup T_+} \left(\left\lfloor \tfrac{(\lambda-\ell_{\sigma(i)}(\mathcal{S}_{h-1}))_+}{q_h} \right\rfloor + 1 \right) \\ &- \sum_{i\in T_+} \left(\left\lfloor \tfrac{(\lambda-\ell_{\sigma(i)}(\mathcal{S}_{h-1}))_+}{q_h} \right\rfloor + 1 \right) + \sum_{i\in T_-} \left\lfloor \tfrac{(\lambda-\ell_i(\mathcal{S}'_{h-1}))_+}{q_h} \right\rfloor - \left\lfloor \tfrac{(\lambda-\ell_{\sigma(i)}(\mathcal{S}_{h-1}))_+}{q_h} \right\rfloor. \end{aligned}$$

Since $T_- \cup T_+$ contains all indices $i \in \mathcal{M}^{\neq}_{h-1}$ such that $\ell_{\sigma(i)}(\mathcal{S}_{h-1}) \leq \lambda$, we have that $\sum_{i\in T_-\cup T_+} \left(\left\lfloor \tfrac{(\lambda-\ell_{\sigma(i)}(\mathcal{S}_{h-1}))_+}{q_h} \right\rfloor + 1 \right) \geq |\mathcal{J}^{=}_h| - 1$. With a bit of work we get that

$$|\mathcal{J}^{=}_h| \leq 1 + |T_+| + \sum_{i\in T_+} \left\lfloor \tfrac{(\lambda-\ell_i(\mathcal{S}'_{h-1}))_+}{q_h} \right\rfloor + \sum_{i\in T_-\cup T_+} \left\lfloor \tfrac{(\lambda-\ell_{\sigma(i)}(\mathcal{S}_{h-1}))_+}{q_h} \right\rfloor - \left\lfloor \tfrac{(\lambda-\ell_i(\mathcal{S}'_{h-1}))_+}{q_h} \right\rfloor,$$

which can be simplified even more since $\sum_{i\in T_+} \left\lfloor \tfrac{(\lambda-\ell_i(\mathcal{S}'_{h-1}))_+}{q_h} \right\rfloor = 0$. Finally, if we consider $T_{\neq} = \{i \in \mathcal{M}^{\neq}_{h-1} : \ell_{\sigma(i)}(\mathcal{S}_{h-1}) \neq \ell_i(\mathcal{S}'_{h-1})\}$, the last expression is at most

$$\begin{aligned} |\mathcal{J}^{=}_h| &\leq 1 + |T_+| + \sum_{i\in(T_-\cup T_+)\cap T_{\neq}} \left\lfloor \tfrac{(\lambda-\ell_{\sigma(i)}(\mathcal{S}_{h-1}))_+}{q_h} \right\rfloor - \left\lfloor \tfrac{(\lambda-\ell_i(\mathcal{S}'_{h-1}))_+}{q_h} \right\rfloor \\ &\leq 1 + |T_+| + |T_{\neq}| + \sum_{i\in T_{\neq}} \tfrac{\ell_i(\mathcal{S}'_{h-1})-\ell_{\sigma(i)}(\mathcal{S}_{h-1})}{q_h}, \end{aligned}$$

which concludes the proof since $|T_{\neq}| \leq \frac{\tilde{p}_{j^*}}{\varepsilon 2^{\ell}}$ (Lemma 12) and the last sum is at most $\frac{\tilde{p}_{j^*}}{\varepsilon 2^{\ell}}$. ◀

▶ **Theorem 18.** *Online LPT is a polynomial time* $(4/3 + O(\varepsilon))$*-competitive algorithm with* $O((1/\varepsilon^3)\log(1/\varepsilon))$ *migration factor.*

We complement this result by improving the lower bound on the best possible competitive ratio for an algorithm with constant migration factor (details can be found in [9]).

▶ **Lemma 19.** *For any* $\varepsilon > 0$*, there is no* $\left(\frac{17}{16} - \varepsilon\right)$*-competitive algorithm using constant migration factor for the online machine covering problem with migration.*

References

1. Y. Azar and L. Epstein. On-line machine covering. *J. Sched.*, 1:67–77, 1998.
2. S. Berndt, K. Jansen, and K. Klein. Fully dynamic bin packing revisited. In *APPROX/RANDOM 2015*, pages 135–151, 2015.
3. Xujin Chen, Leah Epstein, Elena Kleiman, and Rob van Stee. Maximizing the minimum load: The cost of selfishness. *Theor. Comput. Sci.*, 482:9–19, 2013.
4. J. Csirik, H. Kellerer, and G. Woeginger. The exact LPT-bound for maximizing the minimum completion time. *Oper. Res. Lett.*, 11:281–287, 1992.
5. B. Deuermeyer, D. Friesen, and M. Langston. Scheduling to maximize the minimum processor finish time in a multiprocessor system. *SIJADM*, 3:190–196, 1982.
6. L. Epstein and A. Levin. A robust APTAS for the classical bin packing problem. *Math. Program.*, 119:33–49, 2009.
7. L. Epstein and A. Levin. Robust algorithms for preemptive scheduling. *Algorithmica*, 69:26–57, 2014.
8. A. Frangioni, E. Necciari, and M. Scutellà. A multi-exchange neighborhood for minimum makespan parallel machine scheduling problems. *J. Comb. Optim.*, 8:195–220, 2004.
9. Waldo Gálvez, José A. Soto, and José Verschae. Symmetry exploitation for online machine covering with bounded migration. *CoRR*, 2016. `arXiv:1612.01829`.
10. A. Gu, A. Gupta, and A. Kumar. The power of deferral: Maintaining a constant-competitive steiner tree online. *SIAM J. Comput.*, 45:1–28, 2016.
11. D. Hochbaum and D. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM J. Comput.*, 17:539–551, 1988.
12. K. Jansen and K. Klein. A robust AFPTAS for online bin packing with polynomial migration. In *ICALP 2013*, pages 589–600, 2013.
13. K. Jansen, K. Klein, and J. Verschae. Closing the gap for makespan scheduling via sparsification techniques. In *ICALP 2016*, pages 1–13, 2016.
14. J. Łacki, J. Oćwieja, M. Pilipczuk, P. Sankowski, and A. Zych. The power of dynamic distance oracles: Efficient dynamic algorithms for the steiner tree. In *STOC 2015*, pages 11–20, 2015.
15. N. Megow, M. Skutella, J. Verschae, and A. Wiese. The power of recourse for online MST and TSP. *SIAM J. Comput.*, 45:859–880, 2016.
16. D. Recalde, C. Rutten, P. Schuurman, and T. Vredeveld. Local Search Performance Guarantees for Restricted Related Parallel Machine Scheduling. *LATIN 2010*, pages 108–119, 2010.
17. P. Sanders, N. Sivadasan, and M. Skutella. Online scheduling with bounded migration. *Math. Oper. Res.*, 34:481–498, 2009.
18. P. Schuurman and T. Vredeveld. Performance guarantees of local search for multiprocessor scheduling. *INFORMS J. Comput.*, 19:52–63, 2007.
19. M. Skutella and J. Verschae. Robust polynomial-time approximation schemes for parallel machine scheduling with job arrivals and departures. *Math. Oper. Res.*, 41:991–1021, 2016.

20 B. Vöcking. Selfish load balancing. In *Algorithmic Game Theory*, pages 517–542. Cambridge University Press, 2007.

21 G. Woeginger. A polynomial-time approximation scheme for maximizing the minimum machine completion time. *Oper. Res. Lett.*, 20:149–154, 1997.