Operations Research Letters 50 (2022) 322–328

Contents lists available at ScienceDirect

Operations Research Letters

www.elsevier.com/locate/orl

Online load balancing with general reassignment cost

Sebastian Berndt [a], Franziska Eberle [b,*], Nicole Megow [c]

[a] Institute for IT Security, University of Lübeck, Germany
[b] Department of Mathematics, The London School of Economics and Political Science, UK
[c] Faculty of Mathematics and Computer Science, University of Bremen, Germany

ARTICLE INFO

Article history:
Received 25 June 2021
Received in revised form 25 March 2022
Accepted 29 March 2022
Available online 1 April 2022

Keywords:
Online load balancing
Migration
Recourse
Competitive analysis

ABSTRACT

We investigate a semi-online variant of load balancing with restricted assignment. In this problem, we are given n jobs, which need to be processed by m machines with the goal to minimize the maximum machine load. Since strong lower bounds rule out any competitive ratio of $o(\log n)$, we may reassign jobs at a certain job-individual cost. We generalize a result by Gupta, Kumar, and Stein (SODA 2014) by giving a $\mathcal{O}(\log\log mn)$-competitive algorithm with constant amortized reassignment cost.

1. Introduction

Online algorithms are a classical, well-studied way to model problems with an inherent uncertainty. In the offline scenario, the complete input to an algorithm is given right from the start. For many dynamic real-world systems, such a knowledge is often not feasible, as the input only arrives over time. Hence, one aims to produce a solution for these partial inputs that is not too far from an optimal solution produced by an algorithm knowing the complete input. While online algorithms with such a bounded *competitive ratio* exist for a surprisingly large class of problems, the inability of online algorithms to reconsider previous decisions allows to show several impossibility results. Clearly, allowing an arbitrary amount of reconsideration would trivially remove the uncertainty aspect of the problem, and this situation can easily be solved by iterative use of the best offline algorithm for the problem. Furthermore, a small amount of reconsideration is often possible in practice.

To overcome these impossibility results, several semi-online models were introduced and studied that allow a bounded amount of reconsideration. In order to bound this amount of reconsideration, we need to define a metric that measures this reconsideration. A natural approach to do so is to associate with each new object added to the instance a *reassignment cost* that needs to be paid if the object is reassigned during such a reconsideration. Now, if one wants to bound the *number* of reassigned objects, this can be modeled as having unit reassignment costs. This model is usually known as the *recourse model*. On the other hand, one might be interested in bounding the *volume* of the reassigned objects by associating the reassignment costs with the size or volume of the objects. This model is usually known as the *migration model*. In our work, we consider the natural generalization of these models by allowing *arbitrary* reassignment costs.

Scheduling problems have played an important role in the design of online algorithms. In general, a certain set of jobs $\mathcal{J}$ needs to be scheduled on a certain set of machines $\mathcal{M}$. Clearly, such problems arise in the design of process schedulers of operating systems, but also in many problems from operations research such as logistics (see e.g. [24,27]). A very general scheduling problem is called *load balancing on unrelated machines*. Here, processing job $j \in \mathcal{J}$ on machine $i \in \mathcal{M}$ takes time $p_{i,j}$, i.e. the processing time depends heavily on the used machine. These processing times model the modern scenario of highly heterogeneous computing platforms (such as CPUs, GPUs, FPGAs,...). The goal is to distribute $\mathcal{J}$ onto $\mathcal{M}$ such that the maximum load $C_{\max}$, often called the *makespan*, is minimized. Here, the load of a machine i is the sum of processing times $p_{i,j}$ of jobs j assigned to i.

In the *online-list* model, jobs arrive one by one and an online algorithm has to irrevocably assign a job to one of the m machines before the next job is revealed. That is, jobs are revealed in the order $1,\dots,n$, and upon arrival of job j, the scheduler learns the processing time $p_{i,j}$ and has to assign j to a machine before job $j+1$ is revealed. The performance of online algorithms is typically assessed by *competitive analysis*. An online algorithm

* Corresponding author.
E-mail addresses: s.berndt@uni-luebeck.de (S. Berndt), f.eberle@lse.ac.uk (F. Eberle), nicole.megow@uni-bremen.de (N. Megow).

https://doi.org/10.1016/j.orl.2022.03.007

is α-*competitive* if, for each instance, its makespan is bounded by $\alpha\, C^\star_{\max}$, where $C^\star_{\max}$ is the minimal makespan for this job set.

In this paper, we consider an important special case of unrelated machine scheduling, called load balancing with *restricted assignment*, where $p_{i,j} \in \{p_j, \infty\}$. A job j can either be performed on machine i (and will take time p_j) or it cannot be performed on this machine at all (taking time ∞). This special case already captures a lot of the complexity of the unrelated machine model. In fact, all known lower bounds on the competitive ratio of online algorithms for load balancing on unrelated machines already hold for the restricted assignment. In particular, Azar, Naor, and Rom [7] give a lower bound of $\Omega(\log n)$ for any deterministic online algorithm, even in the restricted assignment scenario and with unit processing times $p_{i,j} \in \{1, \infty\}$. Even for randomized algorithms, they show a lower bound of $\ln m$, where $\ln m$ denotes the natural logarithm of $m > 0$, the number of machines. Aspnes et al. [5] provide online algorithms with matching upper bounds (up to constants).

To overcome these strong lower bounds, we relax the irrevocability requirement for an online algorithm. As described above, we associate with each job $j \in \mathcal{J}$ a non-negative *assignment cost* c_j that any scheduler has to pay when it (re)assigns j to a particular machine. We refer to the total assignment cost for the jobs $[k] := \{1, 2, \dots, k\}$ by $C_k := \sum_{j=1}^{k} c_j$. An offline optimum solution for n jobs does not reassign any job and has, thus, total assignment cost $C^\star := C_n$. We say that an online algorithm has a *reassignment factor* of β if its *amortized* reassignment cost for online assigning and possibly reassigning the first k jobs is bounded by $\beta \cdot C_k$ for each $k \in [n]$. The goal is to design an α-competitive online algorithm with bounded reassignment factor. We show that an $\mathcal{O}(1)$-competitive algorithm with constant reassignment factor exists for online load balancing unit-size jobs with restricted assignment. For arbitrary job sizes, this problem admits an algorithm with competitive ratio $\mathcal{O}(\log\log mn)$ and reassignment factor $\mathcal{O}(1)$.

Our model of load balancing with reassignment cost generalizes two previously and only independently studied models, the recourse model (where $c_j = 1$) and the migration model (with $c_j = p_j$). We refer to these special reassignment factors by recourse and migration factor, respectively, and we give details on previous results below. Also other online problems have been investigated in semi-online models with recourse and migration, but hardly in a unified model with reassignment cost. We hope to foster further research on this general semi-online model.

1.1. Related work

Westbrook [31] introduced online scheduling with reassignments in a very general model in which jobs may arrive and depart. Here, the optimal makespan may decrease over time. Therefore, he designs algorithms that are α-competitive against the current optimal load, in contrast to the so far observed maximum optimal load. He gives constant competitive algorithms with constant migration factor and constant recourse factor, respectively, for identical as well as related machines. For arbitrary reassignment costs c_i, the algorithm is $\mathcal{O}\big(\log_\delta \frac{\max_j\{c_j/p_j\}}{\min_j\{c_j/p_j\}}\big)$-competitive with reassignment factor $\mathcal{O}(\delta)$ for some parameter δ with $1 \le \delta \le \frac{\max_j\{c_j/p_j\}}{\min_j\{c_j/p_j\}}$. Andrews, Goemans, and Zhang [3] improve upon these results giving algorithms that are constant competitive against the current optimal load with constant reassignment factor for identical and related machines. We are not aware of any previous work on scheduling with reassignment cost for unrelated machines.

Subsequent work considered either the recourse or the migration model. Both models have been analyzed from an amortized as well as from a worst-case point of view. In the latter, the reassignment cost in round k is required to be bounded by βc_k. Clearly, any worst-case bound translates to a bound in the amortized setting while the reverse is not necessarily true.

Sanders, Sivadasan, and Skutella [29] consider online load balancing on identical parallel machines with migration. Without reassignments, there is a lower bound of $\sqrt{3} \approx 1.88$ on the competitive ratio by Rudin and Chandrasekaran [28] while the best known algorithm achieves a competitive ratio of 1.92 [2]. Sanders et al. [29] improve upon this lower bound when using migration. More precisely, they obtain a $\frac{3}{2}$-competitive algorithm with worst-case migration factor $\frac{4}{3}$. Moreover, they design a family of $(1+\varepsilon)$-competitive algorithms with worst-case migration factor $\beta(\varepsilon)$ allowing for a fully scalable tradeoff between the quality of a solution and its migration cost. In the online setting, they call such a family of algorithms *robust PTAS*. Also for identical parallel machines, Skutella and Verschae [30] develop a robust PTAS for two problems, minimizing the maximum load and maximizing the minimum load on any machine, with an amortized bound on the migration factor. When jobs can be preempted, Epstein and Levin [16] give a 1-competitive, i.e., optimal, algorithm with worst-case migration factor $1 - \frac{1}{m}$.

Awerbuch et al. [6] investigate (among other problems) load balancing on unrelated machines and give an $\mathcal{O}(\log m)$-competitive algorithm reassigning each job at most $\mathcal{O}(\log m)$ times. For the special case where $p_{i,j} \in \{1, \infty\}$ for each job j and each machine i, their algorithm is 16-competitive using $\mathcal{O}(\log m)$ recourse if the optimal makespan is at least $\Omega(\log m)$.

Most relevant for our work is the work by Gupta, Kumar, and Stein [21], who give an online algorithm for the general restricted assignment problem that is $\mathcal{O}(\log\log mn)$-competitive with constant recourse. We give the details in the next section.

For restricted assignment with unit-size jobs, Bernstein et al. [11] give an 8-competitive online algorithm with constant recourse that simultaneously achieves the competitive ratio for every ℓ_q-norm for $q \in [1, \infty]$. That is, if $l = (l_1, \dots, l_m)$ is the load vector of a given job-to-machine assignment, then the ℓ_q-norm of l is defined by $\sqrt[q]{\sum_{i=1}^{m} l_i^q}$ for $q < \infty$ and ℓ_∞ is $\max_i\{l_i\}$. They achieve this by following an optimal assignment with machine loads $(l_1^\star, \dots, l_m^\star)$ such that $l_i \le 8 l_i^\star$ after each job arrival.

Other problems that have been studied in semi-online models with reassignments include matching problems [4,9–11,20,21,25], minimum Steiner tree problems [18,22,26], the traveling salesperson problem [26] as well as packing [8,14,15,17,23] and covering problems [19]. As already mentioned, these problems are typically studied in either in the recourse or the migration model. The only previous work on online optimization with general reassignment cost that we are aware of is on load balancing on identical and related machines [3,31] and bin-packing [17].

1.2. The approach by Gupta, Kumar, and Stein [21]

Besides results on online flows and online matching problems, the authors design an online algorithm for unit-size jobs with constant competitive ratio and constant recourse cost. Further, they describe a high-level idea on how to generalize this algorithm to unit-size jobs with arbitrary reassignment costs.

Using these two algorithms, the authors give an $\mathcal{O}(\log\log mn)$-competitive algorithm for load balancing that incurs constant recourse. To this end, they partition the set of jobs into *big* and *small* jobs. The big jobs are further partitioned into $\mathcal{O}(\log\log mn)$ many classes of roughly equal size, and, after rounding, each class can be solved using the algorithm for unit-size jobs. Every small job j is split into p_j unit-size jobs with the same set of feasible machines as j, for which the algorithm for unit-size jobs is used to determine an assignment. The assignment of the unit-size jobs corresponding to job j is then treated as a probability dis-

tribution over possible assignments for j and rounded carefully, to obtain an assignment for j itself. The algorithm for small jobs achieves a competitive ratio of $\mathcal{O}(1)$ using an expected number of $\mathcal{O}(n)$ reassignments. Combining these two algorithms gives an online algorithm with competitive ratio $\mathcal{O}(\log\log mn)$ and constant recourse.

1.3. Our framework and our contribution

To generalize this result to arbitrary reassignment costs, we follow the same approach as in [21], but adapt the algorithm as well as its analysis at certain key points. First, for scheduling the classes of big jobs, we replace the $\mathcal{O}(1)$-recourse algorithm with its generalization to arbitrary reassignment costs. This immediately guarantees an online algorithm with competitive ratio of $\mathcal{O}(\log\log mn)$ and constant reassignment cost. For the small jobs, we employ again the randomized algorithm in [21], but in this case, we have to adapt the analysis in order to show that this randomized algorithm is still $\mathcal{O}(1)$-competitive and has constant expected reassignment costs.

For the sake of completeness, we additionally give a detailed analysis of the involved algorithms whenever details are missing in [21].

2. Online load balancing

In this section we give our main result, a randomized online algorithm that achieves a competitive ratio of $\mathcal{O}(\log\log mn)$ while having constant expected reassignment cost. First, we describe an online flow problem with rerouting that generalizes online load balancing with unit-size jobs and reassignment cost and recall the algorithm for that problem by [21]. Next, we precisely describe the algorithm for unit-size jobs. Then, we adapt the randomized algorithm for small jobs with constant recourse [21] and give the details of the new analysis before we describe the main result of this section.

2.1. Online flows with rerouting

We consider the following online flow problem. We are given a directed graph $G = (V, A)$ with vertices V and arcs A. Each arc $a \in A$ has a capacity $u_a \in \mathbb{Z}_+$ and an assignment cost $c_a \geq 0$. Moreover, there is a unique source vertex $s \in V$. In round t, vertex $v_t \in V$ is specified as sink and the task is to (unsplittably) send one unit of flow from s to v_t, in addition to the unit flows already being routed from the source to the vertices $v_1, \dots, v_{t-1}$, without violating the arc capacities u_a. Throughout the paper we assume that the underlying offline problem admits a feasible solution fulfilling all capacity constraints, while an online algorithm may violate some capacity constraints, which is necessary for deterministic online algorithms with bounded competitive ratio.

For determining the quality of an algorithm, we are interested in two properties: (i) the factor by which any arc capacity is violated and (ii) the total assignment cost of the flow. In round t, that is *after* satisfying the demand of vertices $v_1, \dots, v_t$, let $(f_a(t))_{a\in A} \in \mathbb{N}^A$ denote the flow found by the online algorithm. We say that the algorithm is *α-competitive* if $f_a(t) \leq \alpha u_a$ holds for each arc a and each round t, which seems orthogonal to the classical definition. However, this different notion is helpful for our use case, online load balancing with restricted assignment.

Let us next describe the relationship between the problem of load balancing unit-size jobs with restricted assignment and the flow problem. In the offline load balancing problem, we create for each machine and for each job one vertex and add one vertex s as source. Given the optimal makespan $C^\star_{\max}$, the source connects to each machine-vertex i by an arc with capacity $u_{s,i} = C^\star_{\max}$ and assignment cost $c_{s,i} = 0$. Further, between each machine-vertex i and each job-vertex j, we draw an arc (i, j) with capacity 1 and assignment cost c_j if and only if j can be scheduled by machine i, i.e., if $p_{i,j} = 1$. By specifying each job-vertex as sink with unit demand, we obtain an instance of the offline version of the above introduced flow problem. The online flow problem assumes that the graph is known upfront while online load balancing is characterized by having the jobs, i.e., in the reduction the job-vertices, revealed one by one. We emphasize that the graph we created has a very special structure. Before a job-vertex is specified as a sink, sending flow along its incident arcs violates the flow conservation at this vertex since all incident arcs are entering this node. Thus, any algorithm that always maintains a feasible solution to the flow problem will not use any of these arcs. The shortest-augmenting-path algorithm designed by Gupta, Kumar and Stein [21] satisfies this condition.

The just developed reduction implies that the lower bound of $\Omega(\log m)$ on the competitive ratio for any online algorithm without reassignment for load balancing with restricted assignment also holds for the online flow problem using the above definition of competitiveness for this problem. To beat this strict lower bound, we allow the online algorithm to reroute flow at a certain assignment cost. More precisely, every time the flow sent along an arc a is decreased or increased by one unit, the assignment cost c_a has to be paid. Let $C^\star_t$ be the assignment cost of an optimal solution after the first t rounds. We aim at developing algorithms that violate the arc capacities by at most a constant factor and simultaneously reroute flow at a reassignment cost bounded by $\mathcal{O}(C^\star_t)$.

To this end, we have a closer look at the shortest path algorithm in [21]. Let f be the flow in graph G after round t. We define the residual network G_t on the vertex set V as follows: For every arc $a \in A$ let $\bar{a}$ be its backward arc, i.e., if $a = (v, w)$, then $\bar{a} = (w, v)$. Set $u^t_a = \alpha u_a - f_a$ and $u^t_{\bar{a}} = f_a$, where α is the competitive ratio we are aiming for. Moreover, let $c^t_a = c^t_{\bar{a}} = c_a$. That is, in contrast to the classical shortest-augmenting-path algorithm, the backward arc of every arc with positive flow has assignment cost identical to its forward arc. If vertex v_t is specified as sink in round t, we use a shortest path algorithm to find P, a shortest path from s to v_t in the residual network G_t. We augment the flow f along P by one unit, i.e., if $a \in P$, then the flow along a is increased by one unit, while $\bar{a} \in P$ implies that f_a is decreased by one unit.

Gupta, Kumar, and Stein [21] show that this algorithm maintains a $(2 + \varepsilon)$-competitive flow while the assignment cost of rerouting the flow is at most $\left(1 + \frac{2}{\varepsilon}\right)$ times the assignment cost of an offline optimum.

Theorem 1 *(Theorem 6.1 in [21]). If there is a feasible solution $f^\star$ to the flow instance G with source s and sinks $v_1, \dots, v_t$ of assignment cost $C^\star_t$, the total assignment cost of augmentations performed by the adapted shortest-augmenting-path algorithm on instance G is at most $\left(1+\frac{2}{\varepsilon}\right)C^\star_t$. The capacities on the arcs are violated by at most a factor $(2 + \varepsilon)$.*

As pointed out already by Bernstein et al. [11], the original proof of the above theorem was erroneous (Lemma 5.4 in [21]) but has been fixed by the authors.

2.2. Unit-size jobs

In this section, we give the details for the usage of the algorithm described in the previous section to solve online load balancing with unit-size jobs with constant competitive ratio and constant recourse. As discussed above, this problem can directly be translated to the online flow problem assuming that $C^\star_{\max}$, the optimal makespan, is known in advance. This assumption is not a

restriction as we can employ a standard guess-and-double framework at the cost of losing an additional factor of 4 in the competitive ratio. Specifically, we start by guessing $C^\star_{\max} = 1$, i.e., we assign the arcs (s, i) for $i \in [m]$ a capacity of $(2+\varepsilon)$, where $\varepsilon > 0$ is the parameter that describes the trade off between competitive ratio and reassignment cost in Theorem 1. That is, our algorithm will be $4(2+\varepsilon)$-competitive with reassignment cost at most $\left(1+\frac{2}{\varepsilon}\right)$. In general, let *round* t refer to the point in time when job j_t is revealed. If $C^\star_{\max}$ is the guess of the optimal makespan in round t, then the arcs (s, i) for $i \in [m]$ have capacity $(2+\varepsilon)C^\star_{\max}$. If the shortest augmenting path algorithm does not find a feasible flow in this network, then Theorem 1 implies that the true optimum is strictly greater than $C^\star_{\max}$. Hence, we double $C^\star_{\max}$ and rerun the shortest augmenting path algorithm on the residual network G_t with the updated capacities $u_{s,i} = (2+\varepsilon)C^\star_{\max}$. As the failure of the shortest augmenting path algorithm before doubling gives a lower bound on the optimal makespan, we obtain the following corollary; see also Section 7 in [21].

Corollary 1. *Let $0 < \varepsilon \leq 1$. If there is a feasible solution with makespan $C^\star_{\max}$ and assignment cost $C^\star$ to the (offline) load balancing problem with restricted assignment and unit-size jobs, then the shortest augmenting path algorithm combined with a guess-and-double framework maintains a schedule with makespan at most $4(2+\varepsilon)C^\star_{\max}$ and reassignment cost at most $\left(1+\frac{2}{\varepsilon}\right)C^\star$.*

We note that this result may overestimate the actual reassignment cost due to the following observation: In the online flow problem, increasing or decreasing the flow along an arc a by one unit costs c_a. When balancing load online with reassignment, the reassignment of job j costs c_j. However, the reduction we use implies that reassigning one unit-size job j from machine i to machine i' is equivalent to decreasing the flow along the arc (i, j) by one unit while simultaneously increasing the flow along the arc (i', j) by one unit. This implies that the cost for rerouting the unit-flow associated with job j is $2c_j$.

2.3. Small jobs

Our algorithm classifies jobs as *big* and *small* depending on the current guess of the optimal makespan and the total number of jobs. Let us assume that we know n, the number of jobs, and $C^\star_{\max}$, the optimal makespan. We justify this assumption later when designing the complete algorithm in Section 2.4. Let $\gamma = \log(mn)$. We say a job j is *big* if $p_j \geq \frac{C^\star_{\max}}{\gamma}$, and otherwise, the job is *small*. Our algorithm treats these jobs differently, and we start by only considering the small jobs, $\mathcal{J}_S$, of the instance. We prove the following.

Theorem 2. *There is a randomized online algorithm maintaining an assignment of the small jobs $\mathcal{J}_S$ with expected makespan at most $\mathcal{O}(1)C^\star_{\max}$ while incurring an expected reassignment cost of at most $\mathcal{O}(1)\sum_{j\in\mathcal{J}_S} c_j$.*

For simplicity, we assume that the set $\mathcal{J}_S$ of small jobs is indexed in the order of the arrival of jobs, i.e., $\mathcal{J}_S = \{1, \dots, n_S\}$, where $n_S = |\mathcal{J}_S|$. For scheduling these jobs, we first generate a fractional assignment of the jobs to machines which we then interpret as probability distribution of the jobs over the machines. By using the rounding scheme of [21], we obtain an integral assignment.

Formally, for job j with processing time p_j and assignment cost c_j, we generate p_j unit-size jobs with reassignment cost $\frac{c_j}{p_j}$ and consider them as an input to online load balancing with unit-size jobs as solved in Section 2.2. The set of machines that are able to process a unit-size job associated with j is identical to the set of eligible machines for job j. Then, the assignment of the associated unit-size jobs gives a fractional assignment of the original job.

Consider round t, i.e., the assignment *after* job t has arrived and was fractionally assigned by the algorithm in p_t steps, one part per step. We are only interested in the final assignment (of all unit-size jobs) and discard the intermediate assignments while job t was only partially assigned. Let $x_{i,j}(t)$ be the number of unit-size jobs of job j that are assigned to machine i at time t. Then, the total (fractional) load on machine i at time t is given by $\ell_i^f(t) = \sum_{j=1}^t x_{i,j}(t)$. Consider a machine i with $x_{i,j}(t) = x_{i,j}(t-1)$. Then, no unit-size job is moved from or to machine i. Hence, the reassignment cost for such a machine is equal to zero. For machine i with $x_{i,j}(t-1) > x_{i,j}(t)$, exactly $x_{i,j}(t-1) - x_{i,j}(t)$ unit-size jobs are moved from machine i to machines i' with $x_{i',j}(t-1) < x_{i',j}(t)$. By definition, reassigning one unit-size job associated with j has actual cost $\frac{c_j}{p_j}$. However, as observed in Section 2.2, the transformation to the online flow problem implies that reassigning one unit-size job from i to i' costs us $2\frac{c_j}{p_j}$ as it involves decreasing the flow on the edge between j and i and increasing the flow on the edge between j and i'. Hence, the assignment cost $c(t)$ incurred due to the arrival of job t is given by

$$c(t) := \sum_{i=1}^{m}\sum_{j=1}^{t} \frac{c_j}{p_j}\left|x_{i,j}(t-1) - x_{i,j}(t)\right|. \tag{1}$$

If there is a schedule with makespan $C^\star_{\max}$, the algorithm maintains a fractional schedule with makespan at most $12C^\star_{\max}$ and reassignment cost at most $\sum_{s=1}^t c(s) \leq 3\sum_{j=1}^t c_j$ by Corollary 1.

Since we are interested in an assignment of the original jobs j, we need to transform the fractional assignment $(x_{i,j}(t))_{i,j}$ at time t to an integral assignment without significantly increasing the reassignment cost. To this end, let $X_j(t) \in [m]$ be the random variable dictating the assignment of j with distribution $\left(\frac{x_{i,j}(t)}{p_j}\right)_{i=1}^m$, i.e., $\mathbb{P}[X_j(t) = i] = \frac{x_{i,j}(t)}{p_j}$. Since the unit-size jobs associated with j have the same set of feasible machines, $x_{i,j}(t) = 0$ if $p_{i,j} = \infty$. Hence, the assignment given by $X_j(t)$ for $1 \leq j \leq t$ is feasible.

However, simply drawing the random variables $X_j(t)$ according to the distribution given by $\left(\frac{x_{i,j}(t)}{p_j}\right)_{i=1}^m$ does not allow us to bound the reassignment cost of the actual jobs in terms of the bound $c(t)$ given in (1). Therefore, we use the rounding approach developed by [21] that takes the realization of $X_j(t-1)$, i.e., the assignment of job j in round $t-1$, into account when drawing the new assignment $X_j(t)$.

In round t, the newly arrived job t is always assigned according to the probabilities $\left(\frac{x_{i,t}(t)}{p_t}\right)_{i=1}^m$ since there is no previous assignment that needs to be taken into account.

We fix a small job $j \in \mathcal{J}_S$ with $j < t$ and construct the following complete bipartite directed graph $G(t)$ with vertex set $V(t-1) \cup V(t)$ and arc set $V(t-1) \times V(t)$, denoted by $A(t)$. The two vertex sets $V(t-1)$ and $V(t)$ contain one vertex for each machine, i.e., $V(s) = \{i(s) : i \in [m]\}$ for $s \in \{t-1, t\}$. An arc $a = (i(t-1), i'(t))$ has cost $c_a = 0$ if $i = i'$. Otherwise, the unit flow cost for arc $a \in A$ equals $c_a := \frac{c_j}{p_j}$. Each vertex $i(t-1)$ is a source with demand $d_{i(t-1)} = -x_{i,j}(t-1)$, while each vertex $i(t)$ is a sink with demand $d_{i(t)} = x_{i,j}(t)$. Since $\sum_i x_{i,j}(t-1) = p_j = \sum_i x_{i,j}(t)$, we can solve the min-cost transportation problem for the p_j units of flow from $V(t-1)$ to $V(t)$; for details please refer to, e.g., [1]. Consider now the integral assignment $X_j(t-1) = i$ of j at time t. Then, pick one of the $x_{i,j}(t-1)$ units placed at i uniformly at random independently of other jobs $j' \neq j$. If this unit is sent to node $i'(t)$ by the solution of the transportation problem, set $X_j(t) = i'$. The

following lemma gives some useful properties of the random variables that enable us to bound the reassignment cost of the integral assignment. As these properties are only mentioned but not proven in [21], we provide a full proof here.

Lemma 1. *The random variables $X_j(t)$ for $1 \le j \le t \le n_S$ satisfy the following properties:*

1. *$X_j(t)$ and $X_{j'}(t)$ are independent for $j \neq j'$,*
2. *$\mathbb{P}[X_j(t) = i] = \frac{x_{i,j}(t)}{p_j}$, and*
3. *$\mathbb{P}[X_j(t-1) \neq X_j(t)] = \sum_{\substack{i \in [m]: \\ \mathbb{P}[X_j(t-1)=i]>0}} \frac{1}{p_j}(x_{i,j}(t-1) - x_{i,j}(t))^+$,*

where $x^+ = \max\{x, 0\}$.

Proof. We fix a time t.

Ad 1 Solving the transportation problem independently for each job implies Property 1.

Ad 2 We prove this by induction on round t. Let $j = 1$ be the first small job that arrived. Clearly, $\mathbb{P}[X_1(1) = i] = \frac{x_{i,1}(1)}{p_1}$ by definition. Suppose now that Property 2 holds for all jobs $1 \le j \le t-1$ in round $t-1$. Consider the fractional assignment $(x_{i,j}(t))_{i,j}$ after job t arrived. Let $f_{i,i'}$ denote the flow from machine vertex $i(t-1)$ to vertex $i'(t)$ as given by the optimal solution to the min-cost transportation problem. If $X_j(t-1) = i$, then the probability that $X_j(t) = i'$ is $\frac{f_{i,i'}}{x_{i,j}(t-1)}$. By the Law of Total Probability and by the induction hypothesis,

$$\begin{aligned}
&\mathbb{P}\big[X_j(t) = i'\big] \\
&\quad = \sum_{\substack{i \in [m]: \\ \mathbb{P}[X_j(t-1)=i]>0}} \mathbb{P}\big[X_j(t) = i' \mid X_j(t-1) = i\big] \\
&\qquad\qquad \cdot \mathbb{P}\big[X_j(t-1) = i\big] \\
&\quad = \sum_{\substack{i \in [m]: \\ \mathbb{P}[X_j(t-1)=i]>0}} \frac{f_{i,i'}}{x_{i,j}(t-1)} \frac{x_{i,j}(t-1)}{p_j} = \frac{x_{i',j}(t)}{p_j},
\end{aligned}$$

where the last equality follows from $f_{i,i'}$ being a feasible solution to the transportation problem.

Ad 3 Recall that $c_{(i(t-1),i(t))} = 0$. For a machine i with $x_{i,j}(t-1) > x_{i,j}(t)$, the optimal solution to the transportation problem sends $x_{i,j}(t-1) - x_{i,j}(t)$ unit jobs to other machines. Thus, $\mathbb{P}\left[X_j(t) \neq X_j(t-1) \mid X_j(t-1) = i\right] = \frac{x_{i,j}(t-1) - x_{i,j}(t)}{x_{i,j}(t-1)}$. For i with $x_{i,j}(t-1) \le x_{i,j}(t)$ the optimal solution to the transportation problem sends $x_{i,j}(t-1)$ unit jobs from $i(t-1)$ to $i(t)$. Thus, $\mathbb{P}\big[X_j(t) \neq X_j(t-1) \mid X_j(t-1) = i\big] = 0$. Therefore,

$$\begin{aligned}
&\mathbb{P}\big[X_j(t) \neq X_j(t-1)\big] \\
&= \sum_{\substack{i \in [m]: \\ \mathbb{P}[X_j(t-1)=i]>0}} \mathbb{P}\big[X_j(t) \neq X_j(t-1) \mid X_j(t-1) = i\big] \\
&\qquad\qquad \cdot \mathbb{P}\big[X_j(t-1) = i\big] \\
&= \sum_{\substack{i \in [m]: \\ \mathbb{P}[X_j(t-1)=i]>0}} \frac{\big(x_{i,j}(t-1) - x_{i,j}(t)\big)^+}{x_{i,j}(t-1)} \frac{x_{i,j}(t-1)}{p_j} \\
&= \sum_{\substack{i \in [m]: \\ \mathbb{P}[X_j(t-1)=i]>0}} \frac{\big(x_{i,j}(t-1) - x_{i,j}(t)\big)^+}{p_j},
\end{aligned}$$

where the first equality holds because of the Law of Total Probability and the second equality follows from Property 2 and the observation discussed above. □

Proof of Theorem 2. We first show that the above described algorithm incurs a total cost of at most $3\sum_{j=1}^{n_S} c_j$ while maintaining a solution that has a small load on each machine in expectation. To this end, let $L_i(t) := \sum_{j:X_j(t)=i} p_j$ denote the random load on machine i at time t. We start with showing that $\mathbb{E}[L_i(t)] \le 12C^\star_{\max}(t)$ for all $1 \le i \le m$. Here, $C^\star_{\max}(t)$ denotes the optimal makespan in round t. As a bound on the expected load per machine is not sufficient to bound the expected maximum, i.e., $\mathbb{E}[\max_i L_i(t)]$, we then show how to guarantee a makespan less than $72C^\star_{\max}$ with probability one at the loss of a constant factor in the reassignment cost.

With Lemma 1, it follows

$$\mathbb{E}[L_i(t)] = \sum_{j=1}^{t} \mathbb{P}[X_j(t) = i] p_j = \sum_{j=1}^{t} \frac{x_{i,j}(t)}{p_j} p_j = \ell_i^f(t),$$

where $\ell_i^f(t)$ is the fractional load on machine i after having assigned job t. By Corollary 1, we know that $\max_{1 \le i \le m}\{\ell_i^f(t)\} \le 12C^\star_{\max}$ if there exists a feasible solution with makespan $C^\star_{\max}$. Now consider the reassignment cost $C(t)$ our algorithm incurs over the course of the arrival of t small jobs. For $1 \le j \le t$, the algorithm pays c_j whenever $X_j(t-1) \neq X_j(t)$. Thus, with Property 3 of Lemma 1, we have

$$\begin{aligned}
\mathbb{E}[C(t)] &= \sum_{j=1}^{t} \mathbb{P}[X_j(t-1) \neq X_j(t)] c_j \\
&= \sum_{j=1}^{t} \sum_{\substack{i \in [m]: \\ \mathbb{P}[X_j(t-1)=i]>0}} \frac{c_j}{p_j} (x_{i,j}(t) - x_{i,j}(t-1))^+ \\
&\le \sum_{j=1}^{t} \sum_{i=1}^{m} \frac{c_j}{p_j} |x_{i,j}(t) - x_{i,j}(t-1)| = c(t).
\end{aligned}$$

Again, with Corollary 1, the expected total cost of the randomized algorithm is bounded by $\sum_{t=1}^{n} c(t) \le 3\sum_{j=1}^{n} c_j$.

Unfortunately, bounding $\mathbb{E}[L_i(t)]$ does not imply a bound on $\mathbb{E}[\max_{1 \le i \le m} L_i(t)]$ as noted by [21].

We use the fact that we are only considering small jobs in order to get a better bound. Consider a time t and a machine i. Let $Y_{i,j}(t)$ indicate whether or not j is assigned to i at time t. So, $Y_{i,j} = \mathbb{1}_{\{X_j(t)=i\}}$ and $L_i(t) = \sum_{j \in \mathcal{J}_S} p_j Y_{i,j}$. We have $\mathbb{E}\Big[\sum_{j \in \mathcal{J}_S} p_j Y_{i,j}(t)\Big] = \ell_i^f(t) \le 12C^\star_{\max}(t)$ as discussed above. Now, we bound the probability that the makespan of our schedule exceeds $72C^\star_{\max}(t)$ in round t.

$$\begin{aligned}
&\mathbb{P}\Big[\max_i L_i(t) > 72C^\star_{\max}(t)\Big] \\
&= \mathbb{P}\left[\exists i : \sum_{j \in \mathcal{J}_S : X_j(t)=i} p_j > 72C^\star_{\max}(t)\right] \\
&\le \sum_{i=1}^{m} \mathbb{P}\left[\sum_{j \in \mathcal{J}_S : X_j(t)=i} p_j > 72C^\star_{\max}(t)\right] \\
&= \sum_{i=1}^{m} \mathbb{P}\left[\sum_{j \in \mathcal{J}_S} \frac{\gamma p_j Y_{i,j}(t)}{C^\star_{\max}(t)} > 72\gamma\right].
\end{aligned}$$

Fix a machine i and a round t. As the random variables $Y_{i,j}$ only depend on the event $\{X_j(t) = 1\}$ and the variables $X_j(t)$ are

independent by construction, for fixed i, the $Y_{i,j}$ are independent as well. Hence, $\frac{\gamma p_j Y_{i,j}(t)}{C^\star_{\max}(t)}$ are independently distributed in $[0,1]$ with $\mathbb{E}\Big[\sum_{j\in\mathcal{J}_S} \frac{\gamma p_j Y_{i,j}(t)}{C^\star_{\max}(t)}\Big] = \frac{\gamma \ell_i^f(t)}{C^\star_{\max}(t)} \le 12\gamma$. Applying a Chernoff-Hoeffding type bound [13, Theorem 1.1] yields

$$\mathbb{P}\left[\sum_{j\in\mathcal{J}_S} \frac{\gamma p_j Y_{i,j}(t)}{C^\star_{\max}(t)} > 72\gamma\right] \le 2^{-72\gamma} \le \frac{1}{(mt)^{72}}.$$

Inserting this in the bound calculated above gives

$$\mathbb{P}\big[\max_i L_i(t) > 72C^\star_{\max}(t)\big] \le \frac{1}{m^{71}t^{72}}.$$

Hence, for one instance with n_S jobs, the probability that the makespan of our algorithm exceeds $72C^\star_{\max}(t)$ in some round t is bounded by

$$\mathbb{P}\big[\exists t : \max_i L_i(t) > 72C^\star_{\max}(t)\big] \le \frac{1}{m^{71}} \sum_{t=1}^{n_S} \frac{1}{t^{72}} \le \frac{1.01}{m^{71}}.$$

Now, whenever the randomized rounding algorithm incurs a makespan more than $72C^\star_{\max}$, we just restart the algorithm from scratch and fast-forward to time t. Then, we reassign all small jobs accordingly, incurring a reassignment cost of at most $C^\star = \sum_{j\in\mathcal{J}_S} c_j$. If we observe such a failure mode, we run the algorithm independently of all previous runs. Hence, the probability that we observe such a failure mode k times for one instance is bounded by $\frac{1.01}{m^{71}} \le \frac{1}{2^k}$ for $m \ge 2$. Thus, the total expected cost of possible failure modes is bounded by $\sum_{k=1}^{\infty} C^\star k \left(\frac{1}{2}\right)^k = 2C^\star$ if $m \ge 2$. □

2.4. The final algorithm

We prove the following theorem.

Theorem 3. *There is a randomized online algorithm maintaining an assignment with expected makespan at most $\mathcal{O}(\log\log(mn))C^\star_{\max}$ while incurring an expected reassignment cost of at most $\mathcal{O}(1)\cdot C^\star$.*

As discussed in Section 2.3, we assume that we know n, the number of jobs we will encounter, and $C^\star_{\max}$, the optimal makespan. Based on these two values, we classify each arriving job as big or as small, where a job j is *big* if $p_j \ge \frac{C^\star_{\max}}{\log(mn)}$, and otherwise, the job is *small*. Small jobs are assigned by the randomized algorithm described in Section 2.3. For big jobs, we first partition them into *classes* $\mathcal{C}_k$, where a job j belongs to $\mathcal{C}_k$ if $p_j \in [2^{k-1}, 2^k)$ for $k \in \mathbb{N}$. Rounding the processing time of jobs in $\mathcal{C}_k$ to 2^{k-1} loses at most a factor 2 in the competitive ratio.

We then separately consider each class, which (after the rounding) constitutes an instance of online load balancing with unit-size jobs. Given that there are $\mathcal{O}(\log\log(mn))$ classes of big jobs, we get:

Corollary 2. *There is an online algorithm maintaining an assignment of the big jobs $\mathcal{J}_B$ with makespan at most $\mathcal{O}(\log\log(mn))C^\star_{\max}$ and reassignment cost at most $\mathcal{O}(1)\sum_{j\in\mathcal{J}_B} c_j$.*

Maintaining $C^\star_{\max}$ or n. We still need to argue that we can assume that either the current value of $C^\star_{\max}$ or n are known to us. Recall that the threshold deciding whether a job is big or small is defined as $\frac{C^\star_{\max}}{\log(mn)}$. This implies that a job changes its classification at most once.

We start with the case where n is known, but not $C^\star_{\max}$. As the optimal makespan $C^\star_{\max}$ only increases, the threshold $\frac{C^\star_{\max}}{\log(mn)}$ is monotonously increasing, implying that a class of big jobs might become small at some point. If this happens, the whole class makes the transition from big to small at the same time and we simply ignore their previous assignment; instead we consider these jobs as newly arriving small jobs (without changing n) and invoke the algorithm for small jobs. Per job, this transition happens at most once, which guarantees that the reassignment cost remains bounded by $\mathcal{O}(1)C^\star$. Theorem 2 guarantees that after such a transition the load on machine i because of small jobs is still bounded by $\mathcal{O}(1)C^\star_{\max}$.

The case where $C^\star_{\max}$ is known but not n is slightly more technical as now our threshold decreases, which implies that a subset of small jobs becomes big and leaves the assignment of small jobs. Again, we treat this transition as arrival of new jobs and give the newly big jobs as input to the algorithm for big jobs. Since this transition happens at most once per job, the overall increase in the reassignment jobs is bounded by $\mathcal{O}(1)C^\star$. However, because other jobs remain small and our algorithm does not handle the case where jobs disappear, we *freeze* their assignment at an additional increase of $\mathcal{O}(1)C^\star_{\max}$ in the load on machine i. This means, any small jobs that have arrived so far and remain small, will not be reassigned in the future. Only small jobs arriving later and the big jobs are subject to reassignments. In the end, there are at most $\mathcal{O}(\log\log(mn))$ many frozen blocks of small jobs, each adding a load of at most $\mathcal{O}(1)C^\star_{\max}$ to machine i. Hence, the total increase of the load on machine i is bounded by $\mathcal{O}(\log\log(mn))C^\star_{\max}$.

Proof of Theorem 3. Theorem 2 guarantees that the algorithm maintains a schedule for the small jobs of makespan at most $\mathcal{O}(1)C^\star_{\max}$ while incurring a reassignment cost of at most $\mathcal{O}(1)\sum_{j\in\mathcal{J}_S} c_j$. Corollary 2 implies that the schedule for the big jobs has makespan at most $\mathcal{O}(\log\log(mn))C^\star_{\max}$ with total cost bounded by $\mathcal{O}(1)\sum_{j\in\mathcal{J}_B} c_j$. Hence, the algorithm achieves a makespan of at most $\mathcal{O}(\log\log(mn))C^\star_{\max}$ with total reassignment cost at most $\mathcal{O}(1)C^\star$. □

3. Concluding remarks

It is somewhat surprising that we achieve the same competitive ratios (up to constants) in all three reassignment models. It remains as an interesting open question whether the problem admits a constant-competitive algorithm in any reassignment model with constant reassignment factor or if there even exists an online algorithm allowing for fully scaleable tradeoff between the competitive ratio and the reassignment factor.

We would like to point out that the analysis of the algorithm is almost tight: Choosing $\gamma \in \Omega\big(\frac{\log(mn)}{\alpha}\big)$ is key to guaranteeing that the expected makespan of the small jobs remains within a factor of $\mathcal{O}(\alpha)$ of $C^\star_{\max}$. While choosing α non-constant, e.g., $\alpha \in \Theta(\log\log(mn))$ is possible, the impact on the competitive ratio is bounded; this still creates $\Omega\big(\log\big(\frac{\log(mn)}{\alpha}\big)\big)$ classes of big jobs. The algorithm treats the $\mathcal{O}\big(\log\big(\frac{\log(mn)}{\alpha}\big)\big)$ classes of big jobs independently, which implies that, even given the optimal schedule for each class, the combined schedule might lose this factor. Consider the following example, where the processing volume is identical for every class and each class can be scheduled on only two machines. Of the two machines, one machine is private and the other one is shared among all classes. The class-optimal schedule uses both machines to the same extent while the combined schedule that only uses the private machines is better by a factor of $\Theta\big(\log\big(\frac{\log(mn)}{\alpha}\big)\big)$. In order to improve upon the competitive ratio of $\mathcal{O}(\log\log(mn))$, an algorithm needs to consider multiple classes at once.

Another challenge is to design an algorithm with a non-trivial bound on the reassignment factor for the general unre-

lated machine scheduling problem, in which jobs may have arbitrary machine-dependent processing times. While a best possible $\Theta(\log n)$-competitive algorithm [5] is known for the online setting, the problem is wide open when allowing reassignments, even for unit reassignment costs. Our (and previous) approaches crucially rely on the fact that the processing times are $\{p_j, \infty\}$.

Further interesting research directions include maximizing the minimum load and the fully dynamic setting where jobs might leave as well. The difficulty in both settings is that there might be time points where the optimum is equal to 0 which makes these types of problems notoriously difficult for approximation. One way to overcome these difficulties is to aim for competitive ratios with an additive constant; such an approach is developed, e.g., in [12], for online load balancing on identical machines.

References

[1] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, Network Flows, Prentice Hall, 1993.
[2] S. Albers, Better bounds for online scheduling, SIAM J. Comput. 29 (2) (1999) 459–473, https://doi.org/10.1137/S0097539797324874.
[3] M. Andrews, M.X. Goemans, L. Zhang, Improved bounds for on-line load balancing, Algorithmica 23 (4) (1999) 278–301, https://doi.org/10.1007/PL00009263.
[4] S. Angelopoulos, C. Dürr, S. Jin, Online maximum matching with recourse, J. Comb. Optim. 40 (4) (2020) 974–1007, https://doi.org/10.1007/s10878-020-00641-w.
[5] J. Aspnes, Y. Azar, A. Fiat, S.A. Plotkin, O. Waarts, On-line routing of virtual circuits with applications to load balancing and machine scheduling, J. ACM 44 (3) (1997) 486–504, https://doi.org/10.1145/258128.258201.
[6] B. Awerbuch, Y. Azar, S.A. Plotkin, O. Waarts, Competitive routing of virtual circuits with unknown duration, J. Comput. Syst. Sci. 62 (3) (2001) 385–397, https://doi.org/10.1006/jcss.1999.1662.
[7] Y. Azar, J. Naor, R. Rom, The competitiveness of on-line assignments, in: SODA, ACM/SIAM, 1992, pp. 203–210, http://dl.acm.org/citation.cfm?id=139404.139450.
[8] S. Berndt, K. Jansen, K. Klein, Fully dynamic bin packing revisited, Math. Program. 179 (1) (2020) 109–155, https://doi.org/10.1007/s10107-018-1325-x.
[9] A. Bernstein, A. Dudeja, Online matching with recourse: random edge arrivals, in: FSTTCS, in: LIPIcs, vol. 182, Schloss Dagstuhl - Leibniz-Zentrum Für Informatik, 2020, pp. 11:1–11:16.
[10] A. Bernstein, J. Holm, E. Rotenberg, Online bipartite matching with amortized O(log ^{2} n) replacements, J. ACM 66 (5) (2019) 37:1–37:23, https://doi.org/10.4230/LIPIcs.ITCS.2017.51.
[11] A. Bernstein, T. Kopelowitz, S. Pettie, E. Porat, C. Stein, Simultaneously load balancing for every p-norm, with reassignments, in: ITCS, in: LIPIcs, vol. 67, Schloss Dagstuhl - Leibniz-Zentrum Für Informatik, 2017, pp. 51:1–51:14.
[12] M. Buchem, L. Rohwedder, T. Vredeveld, A. Wiese, Additive approximation schemes for load balancing problems, in: ICALP, in: LIPIcs, vol. 198, Schloss Dagstuhl - Leibniz-Zentrum Für Informatik, 2021, pp. 42:1–42:17.
[13] D.P. Dubhashi, A. Panconesi, Concentration of Measure for the Analysis of Randomized Algorithms, Cambridge University Press, 2009.
[14] L. Epstein, A. Levin, A robust APTAS for the classical bin packing problem, Math. Program. 119 (1) (2009) 33–49, https://doi.org/10.1007/s10107-007-0200-y.
[15] L. Epstein, A. Levin, Robust approximation schemes for cube packing, SIAM J. Optim. 23 (2) (2013) 1310–1343, https://doi.org/10.1137/11082782X.
[16] L. Epstein, A. Levin, Robust algorithms for preemptive scheduling, Algorithmica 69 (1) (2014) 26–57, https://doi.org/10.1007/s00453-012-9718-3.
[17] B. Feldkord, M. Feldotto, A. Gupta, G. Guruganesh, A. Kumar, S. Riechers, D. Wajc, Fully-dynamic bin packing with little repacking, in: ICALP, in: LIPIcs, vol. 107, Schloss Dagstuhl - Leibniz-Zentrum Für Informatik, 2018, pp. 51:1–51:24.
[18] A. Gu, A. Gupta, A. Kumar, The power of deferral: maintaining a constant-competitive Steiner tree online, SIAM J. Comput. 45 (1) (2016) 1–28, https://doi.org/10.1137/140955276.
[19] A. Gupta, R. Krishnaswamy, A. Kumar, D. Panigrahi, Online and dynamic algorithms for set cover, in: STOC, ACM, 2017, pp. 537–550.
[20] V. Gupta, R. Krishnaswamy, S. Sandeep, Permutation strikes back: the power of recourse in online metric matching, in: APPROX/RANDOM, in: LIPIcs, vol. 176, Schloss Dagstuhl - Leibniz-Zentrum Für Informatik, 2020, pp. 40:1–40:20.
[21] A. Gupta, A. Kumar, C. Stein, Maintaining assignments online: matching, scheduling, and flows, in: SODA, SIAM, 2014, pp. 468–479.
[22] M. Imase, B.M. Waxman, Dynamic Steiner tree problem, SIAM J. Discrete Math. 4 (3) (1991) 369–384, https://doi.org/10.1137/0404033.
[23] K. Jansen, K. Klein, A robust AFPTAS for online bin packing with polynomial migration, SIAM J. Discrete Math. 33 (4) (2019) 2062–2091, https://doi.org/10.1137/17M1122529.
[24] J.K. Lenstra, D.B. Shmoys, Elements of scheduling, CoRR, arXiv:2001.06005 [abs], 2020.
[25] N. Megow, L. Nölke, Online minimum cost matching with recourse on the line, in: APPROX/RANDOM, in: LIPIcs, vol. 176, Schloss Dagstuhl - Leibniz-Zentrum Für Informatik, 2020, pp. 37:1–37:16.
[26] N. Megow, M. Skutella, J. Verschae, A. Wiese, The power of recourse for online MST and TSP, SIAM J. Comput. 45 (3) (2016) 859–880, https://doi.org/10.1137/130917703.
[27] L.P. Michael, Scheduling: Theory, Algorithms, and Systems, Springer, 2018.
[28] J.F. Rudin III, R. Chandrasekaran, Improved bounds for the online scheduling problem, SIAM J. Comput. 32 (3) (2003) 717–735, https://doi.org/10.1137/S0097539702403438.
[29] P. Sanders, N. Sivadasan, M. Skutella, Online scheduling with bounded migration, Math. Oper. Res. 34 (2) (2009) 481–498, https://doi.org/10.1287/moor.1090.0381.
[30] M. Skutella, J. Verschae, Robust polynomial-time approximation schemes for parallel machine scheduling with job arrivals and departures, Math. Oper. Res. 41 (3) (2016) 991–1021, https://doi.org/10.1287/moor.2015.0765.
[31] J.R. Westbrook, Load balancing for response time, J. Algorithms 35 (1) (2000) 1–16, https://doi.org/10.1006/jagm.2000.1074.