Online Algorithms with Advice for Bin Packing and Scheduling Problems

Marc P. Renault[a,1,2,*], Adi Rosén[a,2], Rob van Stee[b]

[a] *CNRS and Université Paris Diderot, France*
[b] *University of Leicester, Department of Computer Science, University Road, Leicester, LE1 7RH, United Kingdom*

Abstract

We consider the setting of online computation with advice and study the bin packing problem and a number of scheduling problems. We show that it is possible, for any of these problems, to arbitrarily approach a competitive ratio of 1 with only a constant number of bits of advice per request. For the bin packing problem, we give an online algorithm with advice that is $(1+\varepsilon)$-competitive and uses $O\left(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon}\right)$ bits of advice per request. For scheduling on m identical machines, with the objective function of any of makespan, machine covering and the minimization of the ℓ_p norm, $p>1$, we give similar results. We give online algorithms with advice which are $(1+\varepsilon)$-competitive ($(1/(1-\varepsilon))$-competitive for machine covering) and also use $O\left(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon}\right)$ bits of advice per request. We complement our results by giving a lower bound that shows that for any online algorithm with advice to be optimal, for any of the above scheduling problems, a non-constant number (namely, at least $\left(1-\frac{2m}{n}\right)\log m$, where n is the number of jobs and m is the number of machines) of bits of advice per request is needed.

Keywords: online algorithms, online computation with advice, competitive analysis, bin packing, machine scheduling

1. Introduction

Online algorithms are algorithms that receive their input one piece at a time. An online algorithm must make an irreversible decision on the processing of the current piece of the input before it receives the next piece, incurring a cost for this processing. The method of choice to analyze such algorithms is *competitive*

*Corresponding author

Email addresses: `marc.renault@lip6.fr` (Marc P. Renault), `adiro@liafa.univ-paris-diderot.fr` (Adi Rosén), `rob.vanstee@leicester.ac.uk` (Rob van Stee)

[1]Present address: Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6, F-75005, Paris, France

[2]Research supported in part by ANR project NeTOC.

analysis [1]. In this framework, the decisions of the online algorithm must be taken with no knowledge about future pieces of input. In competitive analysis, one measures the quality of an online algorithm by analyzing its *competitive ratio*, i.e. the worst-case ratio, over all possible finite request sequences, of the cost of the online algorithm and the cost of an optimal offline algorithm that has full knowledge of the future. In general, there are no computational assumptions made about the online algorithm, and thus competitive analysis is concerned with quantifying the difference between no knowledge of the future and full knowledge of the future.

In many situations, however, an algorithm with no knowledge of the future is unreasonably restrictive [1, 2]. Furthermore, "classical" competitive analysis, as described above, is only concerned with one point on the spectrum of the amount of information about the future available to the online algorithm (i.e. no information at all). In order both to address the lack of a general model for situations of partial information about the future, and to try to quantify the interplay between the amount of information about the future and the achievable competitive ratio, a framework for a more refined analysis of online algorithms, which attempts to analyze online algorithms with partial information about the future, has been proposed and studied in recent years, e.g. [3, 4, 5, 6, 7, 8, 9, 10].

This framework was dubbed *online computation with advice* and, roughly speaking (see Section 2.1 for a formal definition), works as follows. The online algorithm, when receiving each piece of input r_i, can query the adversary about the future by specifying a function u_i going from the universe of all input sequences to a universe of all binary strings of length b, for some $b \geq 0$. The adversary must respond with the value of the function on the whole input sequence (including the parts not yet revealed to the online algorithm). Thus, the online algorithm receives, with each piece of input, b bits of information about the future. We call these *bits of advice.* The decisions of the online algorithm can now depend not only on the input seen so far, but also on the advice bits received so far which reveal some information about the future. The online algorithm can thus improve its competitive ratio. We are typically interested in the interplay between the amount of information received about the future and the achievable competitive ratio. This model was introduced by Emek et al. [4]. Another variant of the setting of online algorithms with advice was proposed by Böckenhauer et al. [3] (see Section 1.1). Recent years have seen an emergence of works on online computation with advice in both variants of the model, e.g. studying problems such as the k-server problem [4, 5, 8, 11], metrical task systems [4], the knapsack problem [7], the bin packing problem [10], 2 value buffer management [9], reordering buffer management problem [12] and more.

In this paper, we study bin packing, and scheduling on m identical machines with the objective functions of the makespan, machine covering, and minimizing the ℓ_p norm in the framework of online computation with advice. All of these problems have been widely studied in the framework of online algorithms (without advice), and in the framework of offline approximation algorithms, e.g. [13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25]. For all of these problems, we show that it is possible to arbitrarily approach a competitive ratio of 1 with a constant number of bits of advice per request, i.e. we give $(1+\varepsilon)$-competitive deterministic algorithms with advice that use $f(1/\varepsilon)$ bits of advice per request (for some polynomial function f). It is worthwhile noting that this is certainly not the case for all online problems, as non-constant lower bounds on the amount of advice required to have a competitive ratio arbitrarily close to 1 are known for some online problems (e.g. for metrical task systems [4]). Furthermore, for all the problems we study, lower bounds bounded away from 1 are known for the competitive ratio achievable by online algorithms without advice. We further show, for the scheduling problems, that a non-constant number of bits of advice is needed for an online algorithm with advice to be optimal (a similar result for bin packing has been given in [10]).

1.1. Related Work.

The model of online computation with advice that we consider in the present paper was introduced by Emek et al [4]. In the model of [4], the advice is a fixed amount that is revealed in an online manner with every request. This model is referred to as the *online advice model*. Another variant of the model of online algorithms with advice was proposed by Böckenhauer et al. [3]. In this variant, the advice is not given to the algorithm piece by piece with each request, but rather a single tape of advice bits is provided to the algorithm. This model is termed the *semi-online advice model* since the algorithm can read from the advice tape at will and, therefore, could read all the advice at the beginning prior to receiving any requests. For the semi-online advice model, one then analyzes the total number of advice bits read from the tape as a function of the length of the input (and the competitive ratio). A number of works have analyzed various online problems in the framework of online algorithms with advice (in both variants). For example: the k-server problem has a competitive ratio of at most $\left\lceil \frac{\lceil \log k \rceil}{b-2} \right\rceil$ [8], where b is the number of bits of advice per request; the metrical task system problem has tight bounds on the competitive ratio of $\Theta(\log N/b)$ [4]; the unweighted knapsack problem has a competitive ratio of 2 with 1 bit of advice in total and $\Omega(\log(n))$ bits are required to further improve the competitive ratio [7], the 2 value buffer management problem has a competitive ratio of 1 with $\Theta((n/B)\log B)$ bits of advice (n is the length of the request sequence and B is the size of the buffer) [9], and the reordering buffer problem, for any $\varepsilon > 0$, has a $(1+\varepsilon)$-competitive algorithm which uses only a constant (depending on ε) number of advice bits per input item [12].

To the best of our knowledge, the only scheduling problems studied to date in the framework of online computation with advice is a special case of the job shop scheduling problem [3, 6] , and, makespan scheduling on indentical machines in [26]. In both cases, the semi-online advice model is used. In [26], an algorithm that is $(1+\varepsilon)$-competitive and uses advice of constant size in total is presented. Boyar et al. [10] studied the bin packing problem with advice, using the semi-online advice model of [3] and presented a 3/2-competitive algorithm, using $\log n + o(\log n)$ bits of advice in total, and a $(4/3+\varepsilon)$-competitive algorithm, using $2n + o(n)$ bits of advice in total, where n is the length of the request sequence. As both algorithms rely on reading $O(\log(n))$ bits of advice prior to receiving any requests, they would use $O(\log(n))$ bits of advice per request in the model used in this paper. The 3/2-competitive algorithm can be converted into an algorithm that uses 1 bit of advice per request. We are not aware of a similar simple conversion for the $(4/3+\varepsilon)$-competitive algorithm. It should be noted that in the online advice model, an algorithm receives at least 1 bit of advice per request, i.e. at least linear advice in total. Finally, they show that an online algorithm with advice requires at least $(n-2N)\log N$ bits of advice in total to be optimal, where N is the optimal number of bins. In [27], an algorithm is presented that has a competitive ratio that can be arbitrarily close to 1.47012 and uses constant advice in total. Further, they show that linear advice in total is required for a competitive ratio better than 7/6.

For online bin packing without advice, the best known lower bound on the competitive ratio is 1.54037 due to Balogh et al. [25] and the best known deterministic upper bound on the competitive ratio is 1.58889 due to Seiden [22]. Chandra [28] showed that all known lower bounds can be shown to apply to randomized algorithms.

For online scheduling on m identical machines without advice, Rudin and Chandrasekaran [24] presented the best known deterministic lower bound of 1.88 on the competitive ratio for minimizing the makespan. The best known deterministic upper bound on the competitive ratio for minimizing the makespan, due to Fleischer et al. [21], is 1.9201 as $m \to \infty$. The best known randomized lower bound on the competitive ratio for minimizing the makespan is $1/(1-(1-1/m)^m)$, which tends to $e/(e-1) \approx 1.58$ as $m \to \infty$, and it was proved independently by Chen et al. [15] and Sgall [17]. The best known randomized algorithm, due to Albers [23], has a competitive ratio of 1.916.

For machine covering, Woeginger [18] proved tight $\Theta(m)$ bounds on the competitive ratio for deterministic algorithms, and Azar and Epstein [20] showed a randomized lower bound of $\Omega(\sqrt{m})$ and a randomized upper bound of $O(\sqrt{m}\log m)$. Also, Azar and Epstein considered the case where the optimal value is known to the algorithm and showed that, for $m \geq 4$, no deterministic algorithm can achieve a competitive ratio better than 1.75.

In the offline case, Fernandez de la Vega and Lueker [13] presented an asymptotic polynomial time approximation scheme (APTAS) for the bin packing problem. Hochbaum and Shmoys [14] developed a polynomial time approximation scheme (PTAS) for the makespan minimization problem on m identical machines. Subsequently, Woeginger [18] presented a PTAS for the machine covering problem on m identical machines and Alon et al. [16] presented a PTAS for the ℓ_p norm minimization problem on m identical machines.

1.2. Our Results.

We give a deterministic online algorithm with advice for bin packing that, for $0 < \varepsilon \leq 1/2$, achieves a competitive ratio of $1+\varepsilon$, and uses $O\left(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon}\right)$ bits of advice per request. For scheduling on m identical machines, we consider the objective functions of *makespan*, *machine covering* and *minimizing the ℓ_p norm* for $p > 1$. For any of these, we give online algorithms with advice that, for $0 < \varepsilon < 1/2$, are $(1 + \varepsilon)$-competitive $((1/(1 - \varepsilon))$-competitive for machine covering) and use $O\left(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon}\right)$ bits of advice per request.

We complement our results by showing that, for any of the scheduling problems we consider, an online algorithm with advice needs at least $\left(1 - \frac{2m}{n}\right)\log m$ bits of advice per request to be optimal, where n is the number of jobs and m is the number of machines. This lower bound uses techniques similar to those used by the analogous lower bound for bin packing found in [10]. We note that with $\lceil \log m \rceil$ bits a trivial algorithm that indicates for each job on which machine it has to be scheduled is optimal.

1.3. Our Techniques.

Common to all our algorithms for the packing and scheduling problems is the technique of classifying the input items, according to their size, into a constant number of classes, depending on ε. For the bin packing problem, there are a constant number of groups of a constant number of items with both of these constants depending on ε. For the scheduling problems, the sizes of the items in one class differ only by a multiplicative constant factor, depending on ε. We classify all the items except the smallest ones in this way, where the bound on the size of the items not classified again depends on ε. This classification is done explicitly in the scheduling algorithms, and implicitly in the bin packing algorithms. We then consider an optimal packing (resp. schedule) for the input sequence and define *patterns* for the bins (the machines) that describe how the critically sized items (jobs) are packed (scheduled) into the bins (machines). The advice bits indicate with each input item into which bin (machine) pattern it should be packed (scheduled). For the bin packing problem, all but the largest classified items can be packed into the optimal number of bins, according to the assigned pattern. The remaining items cause an ε multiplicative increase in the number of bins used. For the scheduling problems, since items in the same class are "similar" in size, we can schedule the items such that the class of the items of each machine matches the class of those in the optimal schedule while being within an ε factor of the optimal. For both the bin packing problem and the scheduling problems, the very small items (jobs) have to be treated separately; in both cases, the items (jobs) are packed (scheduled) while remaining within an ε multiplicative factor of the optimal.

Our techniques for these algorithms are similar to those of [13, 14, 18, 16]. In particular, we use the technique of rounding and grouping the items. The main difficulty in getting our algorithms to work stems from the fact that we must encode the necessary information using only a constant number of advice bits per request. In particular, the number of advice bits per request cannot depend on the size of the input or the size of the instance (number of bins/machines). Further, for the online advice mode, the advice is received per request and this presents additional challenges as the advice has to be presented sequentially per request such that the algorithm will be able to schedule the items in an online manner.

The scheduling objective functions that we consider are all a function of the loads of the machines. This relates closely to the bin packing problem. The main differences are that the bins in the bin packing problem have a maximum capacity and the goal is to minimize the number of bins used. For scheduling on m identical machines, we have no such capacity constraint (i.e. there is no maximum load per machine) but can use at most m machines. This changes the nature of the problem and requires similar but different ideas for the approximation schemes for scheduling as compared to the approximation schemes for bin packing. This is also the case for the online algorithms with advice presented in this paper. The difference is most noticeable in the nature of the grouping of the items that are done implicitly in the case of bin packing based on a ranking of the size of the items and explicitly in the case of scheduling based on a threshold value.

2. Preliminaries

Throughout this paper, we denote by log the logarithm of base 2. For simplicity of presentation, we assume that $1/\varepsilon$ is a natural number.

2.1. Online Advice Model.

We use the model of online computation with advice introduced in [4]. A deterministic online algorithm with advice is defined by the sequence of pairs (g_i, u_i), $i \geq 1$. The functions $u_i : R^* \to U$ are the query functions where R^* is the set of all finite request sequences, and U is an advice space of all binary strings of length b, for some $b \geq 0$. For a given request sequence $\sigma \in R^*$, the advice received with each request $r_i \in \sigma$ is the value of the function $u_i(\sigma)$. The functions $g_i : R^i \times U^i \to A_i$ are the action functions, where A_i is the action space of the algorithm at step i. That is, for request r_j, the action of the online algorithm with advice is $a_j = g_j(r_1, \ldots, r_j, u_1, \ldots, u_j)$, i.e. a function of the requests and advice received to date.

2.2. Competitive Analysis.

Let $\text{ALG}(\sigma)$ be the cost for an online algorithm ALG to process σ and let $\text{OPT}(\sigma)$ be the optimal cost. For a minimization problem, an online algorithm is *c-competitive* if, for all finite request sequences σ, $\text{ALG}(\sigma) \leq c \cdot \text{OPT}(\sigma) + \zeta$, where ζ is a constant that does not depend on σ. For a maximization problem, an algorithm ALG is c-competitive if $\text{ALG}(\sigma) \geq \frac{1}{c}\text{OPT}(\sigma) - \zeta$.

2.3. Bin Packing.

An instance of the *online bin packing problem* consists of a request sequence σ, and an initially empty set B of bins of capacity 1. Each $r_i \in \sigma$ is an item with size $0 < s(r_i) \leq 1$. The goal is to assign all the items of σ to bins such that, for each bin $b_j \in B$, $\sum_{r_i \in b_j} s(r_i) \leq 1$ and $|B|$ is minimized. The optimal number of bins ($|B^{OPT}|$) is denoted by N. An item *fits* into a bin if its size plus the size of previously packed items in that bin is at most 1. For an item $r_i \in b_j$, where b_j is a bin in the packing B, we will write $r_i \in B$. In order to define part of the advice used by our algorithms, we use a common heuristic for bin packing, NEXT FIT [29]. For completeness, we indicate here that the heuristic NEXT FIT packs the item into the current bin if it fits. Else, it closes the current bin, opens a new bin and packs the item in it.

2.4. Scheduling on m Identical Machines.

An instance of the *online scheduling problem on m identical machines* consists of m identical machines and a request sequence σ. Each $r_i \in \sigma$ is a job with a processing time $v(r_i) > 0$. An assignment of the jobs to the m machines is called a *schedule.* For a schedule S, $L_i(S) = \sum_{r_j \in M_i} v(r_j)$ denotes the *load* of machine i in S, where M_i is the set of jobs assigned to machine i in S. In this paper, we focus on the following objective functions:

- *Minimizing the makespan:* minimizing the maximum load over all the machines;
- *Machine cover:* maximizing the minimum load;
- *ℓ_p norm:* minimizing the ℓ_p norm, $1 < p \leq \infty$, of the load of all the machines. For a schedule S, the ℓ_p norm is defined to be $\|L(S)\|_p = \left(\sum_{i=1}^{m}(L_i(S))^p\right)^{1/p}$. Note that minimizing the ℓ_∞ norm is equivalent to minimizing the makespan.

3. Online Algorithms with Advice for Bin Packing

Presented in this section is an algorithm for the online bin packing problem called BPA. The algorithm BPA is inspired by the APTAS algorithms for offline bin packing problem and is $(1+\varepsilon)$-competitive, using $O\left(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon}\right)$ bits of advice per request.

The advice for the algorithm BPA is based on a $(1+2\varepsilon)$-competitive packing of the request sequence, denoted by S. The packing of S is based on the APTAS of Fernandez de la Vega and Lueker [13] but S is created in an online manner so that BPA can produce the same schedule. All the items with size at least ε are grouped, based on their size, into $1/\varepsilon^2$ groups. The groups are numbered sequentially and each item is assigned an *item type* that corresponds to its group number. The packing of the these items uses $(1+\varepsilon)N$ bins and items smaller than ε can be packed using no more than an additional εN bins. The advice indicates the item type and the packing of the bin in which the item is packed. The packing of the bin is described by the types of the items in the bin of S. This allows BPA to reproduce the packing of S.

For the $(1+\varepsilon)$-competitive algorithm, the advice is defined based on an optimal packing of the request sequence. That is, the offline oracle must solve an NP-hard problem. This is possible in this model as no computational restrictions are placed on the oracle. However, it should be noted that the algorithm presented here creates a packing that is $(1+\varepsilon)$-competitive with respect to some packing S^* which does not necessarily have to be an optimal packing. If the computational power of the oracle were restricted, a $(1+\varepsilon)$-competitive (asymptotic) algorithm could be achieved by defining the advice based on the $(1+\varepsilon')$-approximate packing S^* created via a bin packing APTAS, e.g. the scheme of Fernandez de la Vega and Lueker [13], (albeit requiring slightly more bits of advice as ε would have to be adjusted according to ε').

The main result of this section is the following.

Theorem 1. *Given ε, $0 < \varepsilon \leq 1/2$, the competitive ratio for* BPA *is at most $1+3\varepsilon$, and* BPA *uses at most $\frac{1}{\varepsilon}\log\left(\frac{2}{\varepsilon^2}\right)+\log\left(\frac{2}{\varepsilon^2}\right)+3$ bits of advice per request.*

Initially, we will present an algorithm, ABPA, that uses less than $\frac{1}{\varepsilon}\log\left(\frac{2}{\varepsilon^2}\right)+\log\left(\frac{2}{\varepsilon^2}\right)+3$ bits of advice per request and is asymptotically (in the number of optimal bins) $(1+2\varepsilon)$-competitive. Then, with a small modification to ABPA, we will present BPA, an algorithm that is $(1+3\varepsilon)$-competitive for any number of optimal bins and uses 1 more bit per request than ABPA. That is, regardless of the optimal cost, BPA always has a cost that is at most $(1+3\varepsilon)$ times the number of optimal bins.

3.1. Asymptotic $(1+2\varepsilon)$-Competitive Algorithm.

We begin by creating a rounded input σ' based on σ using the scheme of Fernandez de la Vega and Lueker [13]. That is, we will group items based on their size into a finite number of groups and round the size of all the items of each group up to the size of the largest item in the group (see Figure 1).

An item is called *large* if it has size larger than ε. Items with size at most ε are called *small* items. Let the number of large items in σ be L. Sort the large items of σ in order of nonincreasing size. Let $h = \lceil \varepsilon^2 L \rceil$. For $i = 0, \ldots, 1/\varepsilon^2 - 1$, assign the large items $ih+1, \ldots, ih+\lceil \varepsilon^2 L \rceil$ to group $i+1$. A large item of *type* i denotes a large item assigned to group i. The last group may contain less than $\lceil \varepsilon^2 L \rceil$ items. For each item in group i, $i = 1, \ldots, 1/\varepsilon^2$, round up its size to the size of the largest element in the group.

Let σ' be the subsequence of σ restricted to the large items with their sizes rounded up as per the scheme of Fernandez de la Vega and Lueker. We now build a packing S'. The type 1 items will be packed one item per bin. Let B_1 denote this set of bins. By definition, $|B_1| = \lceil \varepsilon^2 L \rceil$. Since large items have size at least ε, $N \geq \varepsilon L$. This implies the following fact.

Fact 1. $|B_1| \leq \lceil \varepsilon N \rceil$

For the remaining *large* items, i.e. types 2 to $1/\varepsilon^2$, in σ', a packing, B_2' that uses at most N bins can be found efficiently [13]. The packing of each bin $b_i \in B_2'$ can be described by a vector of length at most $1/\varepsilon$, denoted $\mathbf{p_i}$, where each value in the vector ranges from 1 to $1/\varepsilon^2$ representing the type of each of the at most $1/\varepsilon$ large items in b_i. This vector will be called a *bin pattern*. Let B_2 be a set of bins such that $|B_2| = |B_2'|$ and each $b_i \in B_2$ is assigned the bin pattern $b_i \in B_2'$. The items of σ' can be assigned sequentially to the bins of B_2, using the following procedure. Initially, the bins of B_2 are all closed. For each $r_i \in \sigma'$, assign r_i with type t_i to the oldest open bin, b_j, such that there are less items of type t_i packed in the bin than are described in $\mathbf{p_j}$. If no such bin

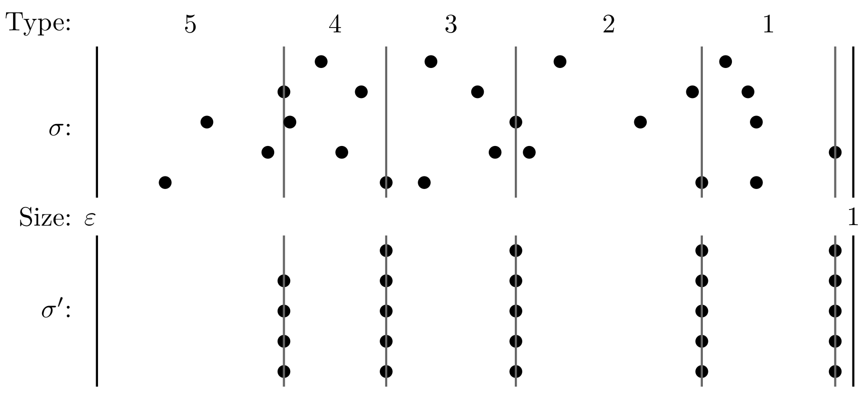

Figure 1: An example of grouping and rounding of large items for $\varepsilon = \sqrt{1/5}$. The top illustration shows the size of 24 large items, denoted by the black dots, from σ grouped into 5 groups of 5 items (except for the last group that contains 4 items), according to a sorting of the items by size. The bottom illustration denotes the same items in the same grouping with their sizes rounded up as in σ'. Note that in the illustration the dots are placed at different heights to be able to clearly distinguish each point and has no other significance.

exists, open a closed bin with a pattern that contains the type t_i and pack r_i in this bin. Note that such a bin must exist by the definition of B_2.

The packing S' is defined to be $B_1 \cup B_2$ with the original (non-rounded up) sizes of the packed large items. The bins of S' are numbered from 1 to $|S'|$ based on the order that the bins would be opened when σ' is processed sequentially. That is, for $i < j$ and every $b_i, b_j \in S'$, there exists an $r_p \in b_i$ such that, for all $r_q \in b_j$, $p < q$. From Fact 1 and $|B_2| \leq N$, we have the following fact.

Fact 2. $|S'| \leq (1 + \varepsilon)N + 1$

We now extend S' to include the small items and define S. Sequentially, by the order that the small items arrive, for each small item $r_i \in \sigma$, pack r_i into S', using NEXT FIT. Additional bins are opened as necessary. The following lemma shows that S is a near-optimal packing. Note the this bound implies that S may pack one more bin than $(1 + 2\varepsilon)$ times the optimal, making it an asymptotically $(1 + 2\varepsilon)$-competitive packing.

Lemma 1. $|S| \leq (1 + 2\varepsilon)N + 1$

Proof. After packing the small items, if no new bins are opened then the claim follows from Fact 2. If there are additional bins opened, all the bins of S, except possibly the last one, are filled to at least $(1 - \varepsilon)$. Since the total size of the items is at most N, we have $(|S| - 1)(1 - \varepsilon) \leq N$ and, therefore, $|S| \leq \frac{N}{1-\varepsilon} + 1 \leq (1 + 2\varepsilon)N + 1$. □

We now define ABPA. It is defined given a σ, and an ε, $0 < \varepsilon \leq 1/2$. ABPA uses two (initially empty) sets of bins L_1 and L_2. L_1 is the set of bins that pack small items and 0 or more large items. L_2 is the set of bins that pack only large items. ABPA and the advice will be defined such that the items are packed exactly as S.

With the first N items, the advice bits indicate a bin pattern. These N bin patterns will be the patterns of the bins in order from S. As the bin patterns are received, they will be queued. Also, with each item, the advice bits indicate the type of the item. Small items will be of type -1. If the item is large, the bits of advice will also indicate if it is packed in S in a bin that also includes small items or not.

During the run of ABPA, bins will be opened and assigned bin patterns. The bins in each of the sets of bins are ordered according to the order in which they are opened. When a new bin is opened, it is assigned an empty bin pattern if the current item is small. If the current item is of type 1, the bin is assigned a type 1 bin pattern. Otherwise, the current item is of type 2 to $1/\varepsilon^2$, and the next pattern from the queue of bin patterns is assigned to the bin. Note that, by the definition of S, this pattern must contain an entry for an item of the current type.

For each $r_i \in \sigma$, the items are packed and bins are opened as follows:

Small Items. For packing the small items, BPA maintains a pointer into the set L_1 indicating the bin into which it is currently packing small items. Additionally, the advice for the small items includes a bit (the details of this bit will be explained subsequently) to indicate if this pointer should be moved to the next bin in L_1. If this is the case, the pointer is moved prior to packing the small item and, if there is no next bin in L_1, a new bin with an empty pattern is opened and added to L_1. Then, the small item is packed into the bin referenced by the pointer.

Large Items. BPA receives an additional bit y as advice that indicates if r_i is packed in a bin in S that also includes small items.

Type 1 *items:* If the item r_i is packed into a bin with small items ($y = 1$), r_i is packed in the oldest bin with an empty pattern. If no such bin exists, then r_i is packed into a new bin that is added to L_1. If r_i is packed into a bin without small items ($y = 0$), then r_i is packed into a new bin that is added to L_2. In all the cases, the bin into which r_i is packed is assigned a type 1 bin pattern.

Type $i > 1$ *items:* Let t_i be the type of r_i. If r_i is packed with small items ($y = 1$), then r_i is packed into the oldest bin of L_1 such that the bin pattern specifies more items of type t_i than are currently packed. If no such bin exists, then r_i is packed in the first bin with an empty bin pattern and the next bin pattern from the queue is assigned to this bin. If there are no empty bins, a new bin is added to pack r_i. If r_i is not packed with small items ($y = 0$), r_i is packed analogously but into the bins of L_2.

The advice bit used to move the pointer for packing small items (see Section 3.1.1 for a formal definition) is defined so that BPA will schedule the same number of small items on each bin as S. Further, BPA schedules both the small and large jobs in the order the arrive on the least recently opened bin just as S (see Figure 2) which implies the following fact.

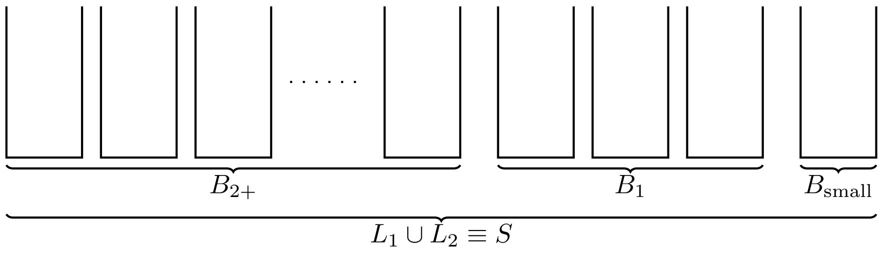

Figure 2: An illustration of the packing produced by ABPA, $L_1 \cup L_2$, that is equivalent to the packing S. B_{2+} packs items of type 2 to $1/\varepsilon^2$ into N bins. B_1 represents the set of εN bins dedicated to packing type 1 items and B_{small} represents the (possibly empty) set of at most $\varepsilon N + 1$ bins dedicated to packing the overflow of small items from the NEXT FIT packing of the small items into the bins of $B_1 \cup B_{2+}$.

Fact 3. *$L_1 \cup L_2$ is the same packing as S.*

Therefore, $|L_1 \cup L_2| \leq (1 + 2\varepsilon)N + 1$ by Lemma 1.

3.1.1. Formal Advice Definition.

Bin Patterns. Instead of sending the entire vector representing a bin pattern, we enumerate all the possible vectors and the advice will be the index of the vector from the enumeration encoded in binary. The bin pattern vectors have a length of at most $1/\varepsilon$ and there are at most $1/\varepsilon^2$ different possible values. To ensure that the all vectors have the same length, a new value $\perp$ is used to pad vectors to a length of $1/\varepsilon$. The increase the number of possible values per entry to $1/\varepsilon^2 + 1$.

The algorithm requires less than $\left\lceil \frac{1}{\varepsilon} \log\left(\frac{1}{\varepsilon^2} + 1\right) \right\rceil < \frac{1}{\varepsilon} \log\left(\frac{2}{\varepsilon^2}\right) + 1$ bits of advice per request to encode the index of the bin pattern from an enumeration of all possible bin patterns in binary.

Advice per Request. In order to define the advice, for each bin $b_i \in S$, we define a value κ_i that is the number of small items packed in b_i.

Per request, the advice string will be xyz, where x is $\left\lceil \log\left(1/\varepsilon^2 + 1\right) \right\rceil < \log\left(2/\varepsilon^2\right) + 1$ bits in length to indicate the type of the item; y is 1 bit in length to indicate whether the large items are packed with small items, or to indicate to small items whether or not to move to a new bin; z has a length less than $\frac{1}{\varepsilon} \log\left(\frac{2}{\varepsilon^2}\right) + 1$ to indicate a bin pattern. xyz is defined as follows for request r_i:

x**:**		The type of r_i encoded in binary.
y**:**	r_i is a small item:	Let s be the number of small jobs in $\langle r_1, \ldots, r_{i-1}\rangle$. If there exists an integer $1 \leq j \leq N$ such that $\sum_{k=1}^{j} \kappa_k = s$, then the first bit is a 1. Otherwise, the first bit is a 0.
	r_i is a large item:	1, if $\kappa_i > 0$, where b_i is the bin in which r_i is packed in S, i.e. b_i packs small items. Otherwise, 0.
z**:**	$i \leq N$	The bits of z encode a number in binary indicating the vector representing the bin pattern of the i-th bin opened by S'.
	$i > N$	Not used. All zeros.

3.2. Strict $(1 + 3\varepsilon)$*-Competitive Algorithm.*

BPA is defined such that it will behave in two different manners, depending on N (i.e. the number of bins in an optimal packing) and ε. One bit of advice per request, denoted by w, is used to distinguish between the two cases. The two cases are as follows.

Case 1: $N > 1/\varepsilon$ *(*$w = 0$*).* BPA will run ABPA as described previously. The only difference is that the advice per request for ABPA is prepended with an additional bit for w. Since $N > 1/\varepsilon$, a single bin is at most εN bins. Therefore, we get the following corollary to Lemma 1.

Corollary 1. $|S| \leq (1 + 3\varepsilon)N$

Case 2: $N \leq 1/\varepsilon$ *(*$w = 1$*).* In this case, for each $r_i \in \sigma$, after w, the next $\lceil \log(1/\varepsilon) \rceil$ bits of advice per request define the bin number in which r_i is packed in an optimal packing. BPA will pack r_i into the bin as specified by the advice. This case requires less than $\log(1/\varepsilon) + 2 < \frac{1}{\varepsilon} \log\left(\frac{2}{\varepsilon^2}\right) + \log\left(\frac{2}{\varepsilon^2}\right) + 3$ (the upper bound on the amount of advice used per request in case 1) bits of advice per request and the packing produced is optimal.

The definition of the algorithm and the advice, Fact 3 and Corollary 1 prove Theorem 1.

4. Online Algorithms with Advice for Scheduling

In this section, we present a general framework for the online scheduling problem on m identical machines. This framework depends on a positive $\varepsilon < 1/2$, $U > 0$, and the existence of an optimal schedule S^*, where all jobs with a processing time greater than U are scheduled on a machine without any other jobs. The framework will produce a schedule S such that, up to a permutation of the machines of S, the load of machine i in S is within $\varepsilon L_i(S^*)$ of the load of machine i in S^*, where $L_i(S)$ denotes the load of machine i in the schedule S. This is done using $O\left(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$ bits of advice per request. We show that this nearly optimal schedule is sufficient for $(1 + \varepsilon)$-competitive algorithms for the makespan and minimizing the ℓ_p norm objectives, and a $(1/(1-\varepsilon))$-competitive algorithm for the machine cover objective.

As is the case with the $(1+\varepsilon)$-competitive algorithm for bin packing problem presented in Section 3, the algorithms with a competitive ratio of $(1+\varepsilon)$ (resp. $(1/(1-\varepsilon))$) presented in this section could use advice that was based on a schedule produced by a PTAS for the desired objective function as opposed to an optimal schedule.

If $\log m$ is less than the number of bits of advice per request to be given to the algorithm, then the trivial algorithm with advice that encodes, for each job, the machine number to schedule that job could be used to obtain a 1-competitive algorithm instead of the framework presented in this section.

4.1. General Framework

The machines are numbered from 1 to m. Given an ε, $0 < \varepsilon < 1/2$, and $U > 0$, the requested jobs will be classified into a constant number of *types*, using a geometric classification. U is a bound which depends on the objective function of the schedule. Formally, a job is of type i if its processing time is in the interval $(\varepsilon(1+\varepsilon)^i U, \varepsilon(1+\varepsilon)^{i+1} U]$ for $i \in [0, \lceil \log_{1+\varepsilon} \frac{1}{\varepsilon} \rceil)$. These jobs will be called *large jobs* (see Figure 3). Jobs with processing times at most εU will be considered *small jobs* and have a type of -1. Jobs with processing times greater than U will be considered *huge jobs* and have a type of $\lceil \log_{1+\varepsilon} \frac{1}{\varepsilon} \rceil$. The online algorithm does not need to know the actual value of the threshold U.

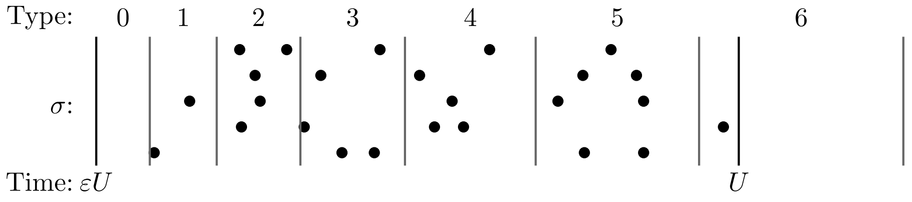

Figure 3: An example of the geometric grouping of large jobs, jobs with a processing time in the range $(\varepsilon U, U]$, for $\varepsilon = 1/4$. The illustration shows the processing time of the 25 large jobs, denoted by the black dots, from σ grouped into 7 groups, according to a sorting of the jobs by processing time. The i-th group consists of jobs with a processing time in the interval $(\varepsilon(1+\varepsilon)^i U, \varepsilon(1+\varepsilon)^{i+1} U]$. Even though the range for type 6 is greater than U, only jobs with a processing time at most U will be assigned type 6.

Let S^* be an optimal schedule for the input at hand. In what follows, we will define a schedule S' from S^* such that, for all i, $L_i(S') \in [L_i(S^*) - \varepsilon U, L_i(S^*) + \varepsilon U]$. Then, based on S', we will define a schedule S such that $L_i(S) \in [(1-\varepsilon)L_i(S^*) - \varepsilon U, (1+\varepsilon)L_i(S^*) + \varepsilon U]$. The advice will be defined so that the online algorithm will produce the schedule S.

The framework makes the following assumption. For each of the objective functions that we consider, we will show that there always exists an optimal schedule for which this assumption holds.

Assumption 1. *S^*, the optimal schedule on which the framework is based, is a schedule such that each huge job is scheduled on a machine that does not contain any other job.*

The general framework is defined given a σ, an ε, a U, and an S^* under Assumption 1. For the schedule S^*, we assume without loss of generality that machines are numbered from 1 to m, according to the following order. Assign to each machine the request index of the first large job scheduled on it. Order the machines by increasing order of this number. Machines on which no large job is scheduled are placed at the end in an arbitrary order.

We define S' by removing the small jobs from S^*. S' can be described by m patterns, one for each machine. Each such pattern will be called a *machine pattern*. For machine i, $1 \leq i \leq m$, the machine pattern indicates that (1) the machine schedules large or huge jobs, or (2) an empty machine (such a machine may schedule only small jobs). In the first case, the machine pattern is a vector with one entry per large or huge job scheduled on machine i in S'. These entries will be the job types of these jobs on machine i ordered from smallest to largest. Let v denote the maximum length of the machine pattern vectors for S'. The value of v will be dependent on the objective function and U. We later show that for all the objective functions we consider, $v \leq 1/\varepsilon + 1$.

We now extend S' to also include the small jobs. Figure 4 depicts the proof of the following lemma.

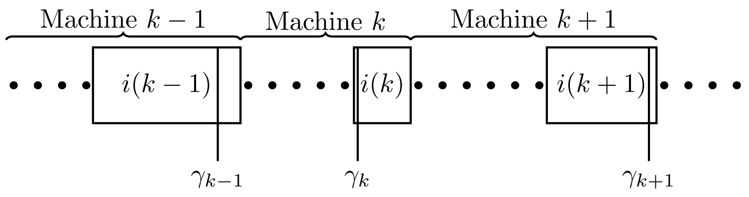

Figure 4: An illustration of the proof of Lemma 2. In the illustration, the rectangles represent the processing time of the small jobs as they are ordered in σ. The value γ_i indicates the total processing time of the small jobs scheduled on the first i machines in an optimal schedule. The difference between γ_i and γ_{i-1} ($\gamma_0 = 0$) being the processing time of the smalls jobs on machine i which is denoted by y_i in Lemma 2. The small jobs are assigned in a next fit manner to the machines of S' such that the small jobs scheduled on machine k include all the small jobs from the first small job immediately after the last small job scheduled on machine $k-1$ ($i(k-1)$) to the small job ($i(k)$) such that the total processing time of all the small jobs prior to and including $i(k)$ is at least γ_k. This ensures that the total processing time of the small jobs assigned to machine k is within εU of the processing time of the small jobs on machine k in the optimal schedule.

Lemma 2. *The small jobs of σ can be scheduled on the machines of S' sequentially in a next fit manner from machine 1 to machine m, such that the load for each machine i will be in $[L_i(S^*) - \varepsilon U, L_i(S^*) + \varepsilon U]$.*

Proof. Consider the small jobs in the order in which they arrive. Denote the processing time of the jth small job in this order by x_j for $j = 1, \ldots$. For

$i = 1, \dots, m$, let y_i be the total processing time of small jobs assigned to machine i in S^*. Let $i(0) = 0$, and for $k = 1, \dots, m$, let $i(k)$ be the minimum index such that $\sum_{j=1}^{i(k)} x_j \geq \sum_{i=1}^{k} y_i$. Finally, for $k = 1, \dots, m$, assign the small jobs $i(k-1)+1, \dots, i(k)$ to machine k. (If $i(k) = i(k-1)$, machine k receives no small jobs.)

By the definition of $i(k)$ and the fact that all small jobs have a processing time at most εU, the total processing time of small jobs assigned to machines $1, \dots, k$ is in $[\sum_{i=1}^{k} y_i, \sum_{i=1}^{k} y_i + \varepsilon U]$ for $k = 1, \dots, m$. By taking the difference between the total assigned processing time for the first $k-1$ and for the first k machines, it immediately follows that the total processing time of small jobs assigned to machine k is in $[y_k - \varepsilon U, y_k + \varepsilon U]$. □

Note that some machines may not receive any small jobs in this process. We will use the advice bits to separate the machines that receive small jobs from the ones that do not, so that we can assign the small jobs to consecutive machines.

We now define the schedule S, using the following procedure. Assign the machine patterns of S' in the same order to the machines of S. For each large or huge job $r_i \in \sigma$, in the order they appear in σ, assign r_i with type t_i to the first machine in S such that the number of jobs with type t_i currently scheduled is less than the number of jobs of type t_i indicated by the machine pattern. After all the large and huge jobs have been processed, assign the small jobs to the machines of S exactly as they are assigned in S' in Lemma 2.

Lemma 3. *For* $1 \leq i \leq m$, $L_i(S) \in [(1-\varepsilon)L_i(S^*) - \varepsilon U, (1+\varepsilon)L_i(S^*) + \varepsilon U]$.

Proof. By Lemma 2 and the fact that jobs of the same type differ by a factor of at most $1 + \varepsilon$, we have $L_i(S) \in \left[\frac{1}{1+\varepsilon} L_i(S^*) - \varepsilon U, (1+\varepsilon)L_i(S^*) + \varepsilon U\right]$. The claim follows since $1/(1+\varepsilon) > 1 - \varepsilon$ for $\varepsilon > 0$. □

We have thus shown that in S the load on every machine is very close to the optimal load (for an appropriate choice of U). Note that this statement is independent of the objective function. This means if we can find such a schedule S online with a good value of U, we can achieve our goal for every function of the form $\sum_{i=1}^{m} f(L_i)$, where f satisfies the property that if $x \leq (1+\varepsilon)y$ then $f(x) \leq (1 + O(1)\varepsilon) f(y)$.

We now define the online algorithm with advice for the general framework, which produces a schedule equivalent to S up to a permutation of the machines. For simplicity of presentation, we assume that this permutation is the identity permutation.

For the first m requests, the general framework receives as advice a machine pattern and a bit y, which indicates whether this machine contains small jobs or not (see Figure 5). For r_j, $1 \leq j \leq m$, if $y = 0$, the framework assigns the machine pattern to the *highest* machine number without an assigned pattern. Otherwise, the framework will assign the machine pattern to the *lowest* machine number without an assigned pattern. For each request r_i in σ, the type of r_i, denoted by t_i, is received as advice. The framework schedules r_i, according to t_i, as follows:

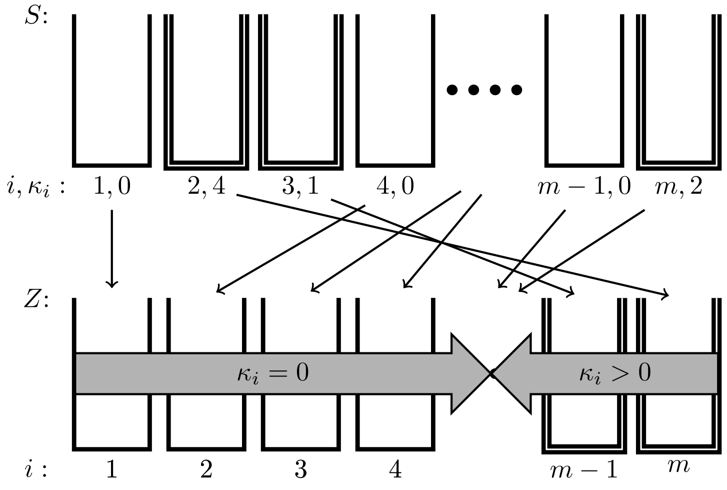
S:
i, κ_i : $1,0$ $2,4$ $3,1$ $4,0$ $m-1,0$ $m,2$
Z:
$\kappa_i = 0$
$\kappa_i > 0$
i : 1 2 3 4 $m-1$ m

Figure 5: In order to produce the exact same schedule as S, the online algorithm must permute the machines of S based on the number of small jobs scheduled per machine (denoted by κ_i for machine i). This figure illustrates such a permutation. Z denotes the schedule produced by the online algorithm. Note that, in Z, machines with no small jobs, single line ($\kappa_i = 0$), are in the same relative order as S and machines with small jobs, double line ($\kappa_i > 0$), are in reverse relative order. This allows the jobs for machines without small jobs to be scheduled from left to right and jobs for machines with small jobs to be scheduled from right to left.

Small Jobs ($t_i = -1$). For scheduling the small jobs, the algorithm maintains a pointer to a machine (initially machine m) indicating the machine that is currently scheduling small jobs. With each small job, the algorithm gets a bit of advice x that indicates if this pointer should be moved to the machine with the preceding serial number. If so, the pointer is moved prior to scheduling the small job. Then, r_i is scheduled on the machine referenced by the pointer.

Large and Huge Jobs ($0 \le t_i \le \lceil \log_{1+\varepsilon} \frac{1}{\varepsilon} \rceil$). The algorithm schedules r_i on a machine where the number of jobs of type t_i is less than the number indicated by its pattern.

4.1.1. Formal advice definition.

Machine Patterns. For the first m requests, a machine pattern is received as advice. Specifically, all possible machine patterns will be enumerated and the id of the pattern, encoded in binary, will be sent as advice for each machine. For large jobs, there are at most v jobs in a machine pattern vector, and each job has one of $\lceil \log_{1+\varepsilon} \frac{1}{\varepsilon} \rceil$ possible types. The machine patterns can be described with the jobs ordered from smallest to largest since the order of the jobs on the machine is not important. This is equivalent to pulling v names out of $\lceil \log_{1+\varepsilon} \frac{1}{\varepsilon} \rceil + 1$ names (one name to denote an empty entry and $\lceil \log_{1+\varepsilon} \frac{1}{\varepsilon} \rceil$ names for each of the large job types), where repetitions are allowed and order is not significant. Therefore, there are

$$\binom{v + \lceil \log_{1+\varepsilon} \frac{1}{\varepsilon} \rceil}{\lceil \log_{1+\varepsilon} \frac{1}{\varepsilon} \rceil} \leq \left\lceil \log_{1+\varepsilon} \frac{1}{\varepsilon} \right\rceil^v$$

different possible machine patterns for the machines scheduling large jobs. Additionally, there is a machine pattern for machines with only small jobs and a machine pattern for machines with only a huge job. Hence, at most $\beta(v) \leq \lceil \log(2 + \lceil \log_{1+\varepsilon} \frac{1}{\varepsilon} \rceil^v) \rceil < v \log\left(\frac{3\log(1/\varepsilon)}{\log(1+\varepsilon)}\right) + 1$ bits are required to encode the index of a machine pattern in an enumeration of all possible machine patterns in binary. As we show in the following, for the cases of makespan, machine cover and ℓ_p norm, $v \leq 1/\varepsilon + 1$ and $\beta(v) < \frac{1+\varepsilon}{\varepsilon} \log\left(\frac{3\log(1/\varepsilon)}{\log(1+\varepsilon)}\right) + 1$.

Advice per Request. In order to define the advice, for each machine $m_i \in S$, we define a value κ_i that is the number of small jobs scheduled on m_i.

Per request, the advice string will be $wxyz$, where w has a length of $\lceil \log(2 + \lceil \log_{1+\varepsilon} \frac{1}{\varepsilon} \rceil) \rceil < \log\left(\frac{3\log(1/\varepsilon)}{\log(1+\varepsilon)}\right) + 1$ bits to indicate the job type, x and y are 1 bit in length (as described above), and z has a length of $\beta(v)$ bits to indicate a machine pattern. $wxyz$ is defined as follows for request r_i:

w:		A number in binary representing the type of r_i.
x:	r_i is a small job:	$x = 1$ if the small job should be scheduled on the next machine. Otherwise, $x = 0$. More formally, let s be the number of small jobs in $\langle r_1, \ldots, r_{i-1} \rangle$. If there exists and an integer $1 \leq j \leq m$ such that $\sum_{k=1}^{j} \kappa_k = s$, then $x = 1$. Otherwise, $x = 0$.
	otherwise:	x is unused and the bit is set to 0.
y:	$i \leq m$:	If $\kappa_i > 0$, $y = 0$. Otherwise, $y = 1$.
	$i > m$:	This bit is unused and set to 0.
z:	$i \leq m$:	z is a number in binary indicating the machine pattern of machine i in S'.
	$i > m$:	z is unused and all the bits are set to 0.

Fact 4. *This framework uses less than* $\log\left(\frac{3\log(1/\varepsilon)}{\log(1+\varepsilon)}\right) + \beta(v) + 3$ *bits of advice per request.*

The following theorem, which follows immediately from the definition of the general framework and Lemma 3, summarizes the main result of this section.

Theorem 2. *For any σ, an ε, $0 < \varepsilon < 1/2$, and a $U > 0$ such that there exists an S^* under Assumption 1, the general framework schedules σ such that for all machines, $1 \leq i \leq m$, $L_i(S) \in [(1-\varepsilon)L_i(S^*) - \varepsilon U, (1+\varepsilon)L_i(S^*) + \varepsilon U]$.*

4.2. Minimum Makespan

For minimizing the makespan on m identical machines, we will define $U = \textsc{opt}$, where $\textsc{opt}$ is the minimum makespan for σ.

Fact 5. *If $U = \textsc{opt}$, there are no huge jobs as the makespan is at least as large as the largest processing time of all the jobs.*

By the above fact, we know that Assumption 1 holds.

Lemma 4. *The length of the machine pattern vector is at most $\frac{1}{\varepsilon}$.*

Proof. This lemma follows from the fact that all large jobs have a processing time greater than $\varepsilon U = \varepsilon\textsc{opt}$ and that a machine in S^* with more than $\frac{1}{\varepsilon}$ jobs with processing times greater than $\varepsilon\textsc{opt}$ is more than the maximum makespan, a contradiction. $\square$

From Lemma 4, $v = \frac{1}{\varepsilon}$. Using this value with Fact 4 of the general framework, gives the following.

Fact 6. *The online algorithm with advice, based on the general framework, uses at most $\frac{2}{\varepsilon}\left(\log\left(\frac{3\log(1/\varepsilon)}{\log(1+\varepsilon)}\right)\right) + 4$ bits of advice per request.*

Theorem 3. *Given a request sequence σ, $U = \textsc{opt}$ and an ε, $0 < \varepsilon < 1/2$, the online algorithm with advice, based on the general framework schedules the jobs of σ such that the online schedule has a makespan of at most $(1 + 2\varepsilon)\textsc{opt}$.*

Proof. By Fact 5, Assumption 1 holds and Theorem 2 applies.

Let j be a machine with the maximum load in S^*. By Theorem 2, $L_i(S) \leq (1 + \varepsilon)L_i(S^*) + \varepsilon U \leq (1 + 2\varepsilon)\textsc{opt}$ as $U = \textsc{opt} = L_j(S^*) \geq L_i(S^*)$ for all $1 \leq i \leq m$. $\square$

4.3. Machine Covering

For maximizing the minimum load, i.e. machine covering, on m identical machines, we will define $U = \textsc{opt}$, where $\textsc{opt}$ is the load of the machine with the minimum load in S^*.

Lemma 5. *There exists an optimal schedule S such that any job with processing time more than that of the minimum load, i.e. a huge job, will be scheduled on a machine without any other jobs.*

Proof. In S, let machine i be the machine with the minimum load. Note that by definition a huge job has a processing time that is more than the minimum load. Therefore, machine i cannot contain a huge job. Assume that scheduled on some machine $j \neq i$ is a huge job and one or more large or small jobs. We will denote the set of non-huge jobs scheduled on machine j by J. We will define another schedule S^* to be the same schedule as S for all the machines but i and j. In S^*, machine i will schedule the same jobs as in S plus all the jobs in J and machine j will only schedule the huge job scheduled on machine j in S. The load on machine j in S^* is greater than $\textsc{opt}$ as it contains a huge job and the load on machine i in S^* is greater than OPT given that it was OPT in S and jobs were added to it in S^*. If the load of machine i in S is a unique minimum, then S^* contradicts the optimality of S. Otherwise, there exists another machine, $k \neq i$ and $k \neq j$, with the same load as i in S. Machine k has the same load in S^* as it does in S. Therefore, S^* is an optimal schedule. This process can be repeated until a contradiction is found or an optimal schedule is created such that no huge job is scheduled on a machine with any other jobs. □

Lemma 6. *There exists an optimal schedule S such that there are at most $1 + \frac{1}{\varepsilon}$ non-small jobs scheduled on each machine and huge jobs are scheduled on a machine without any other jobs.*

Proof. By Lemma 5, we can transform any optimal schedule S to an optimal schedule S', where all the huge jobs are scheduled on machines without any other jobs.

In S', let machine i be the machine with the minimum load and assume that some machine $j \neq i$ has more than $1 + \frac{1}{\varepsilon}$ large jobs. We will define another schedule S^* to be the same schedule as S' for all the machines but i and j. Note that machine i has at most $\frac{1}{\varepsilon}$ large jobs scheduled and, since its load is U, it cannot contain a huge job because huge jobs have processing times more than U. In S^*, machine i will schedule the same jobs as S' plus all the small jobs and the largest job scheduled on machine j in S'. The load on machine j in S^* is greater than OPT as it still has at least $1 + \frac{1}{\varepsilon}$ large jobs scheduled on it and the load on machine i in S^* is greater than OPT given that it was OPT in S' and jobs were added to it in S^*. If the load of machine i in S' is a unique minimum, then S^* contradicts the optimality of S'. Otherwise, there exists another machine, $k \neq i$ and $k \neq j$, with the same load as i in S'. Machine k has the same load in S^* as it does in S'. Therefore, S^* is an optimal schedule. This process can be repeated until a contradiction is found or an optimal schedule is created such that no machine has more than $1 + \frac{1}{\varepsilon}$ non-small jobs scheduled. □

From Lemma 6, $v = 1 + \frac{1}{\varepsilon}$. Using this value with Fact 4 of the general framework gives the following.

Fact 7. *The online algorithm with advice, based on the general framework, uses at most $\frac{3}{\varepsilon}\left(\log\left(\frac{3\log(1/\varepsilon)}{\log(1+\varepsilon)}\right)\right) + 4$ bits of advice per request.*

Theorem 4. *Given a request sequence σ, $U =$ OPT and an ε, $0 < \varepsilon < 1$, the online algorithm with advice, based on the general framework, schedules the jobs of σ such that the online schedule has a machine cover at least $(1 - 2\varepsilon)$OPT.*

Proof. By Lemma 6, Assumption 1 holds and Theorem 2 applies.

Let j be a machine with the minimum load in S^*. By Theorem 2, $L_i(S) > (1 - \varepsilon)L_i(S^*) - \varepsilon U \geq (1 - 2\varepsilon)\text{OPT}$ as $\text{OPT} = L_j(S^*) \leq L_i(S^*)$ for all $1 \leq i \leq m$. □

4.4. The ℓ_p Norm

For minimizing the ℓ_p norm on m identical machines, we will define $U = \frac{W}{m}$, where W is the total processing time of all the jobs.

For completeness, we first prove the following technical lemma about convex functions.

Lemma 7. *Let f be a convex function, and let $x_0 > y_0 \geq 0$. Let $v < x_0 - y_0$. Then $f(x_0 - v) + f(y_0 + v) < f(x_0) + f(y_0)$.*

Proof. We need to show that $f(x_0) - f(x_0 - v) > f(y_0 + v) - f(y_0)$ for $y_0 < x_0$ and $0 < v < x_0 - y_0$.

Suppose first that $v < (x_0 - y_0)/2$. Due to the mean value theorem, there exist values $\theta_1 \in [y_0, y_0 + v], \theta_2 \in [x_0 - v, x_0]$, such that $f'(\theta_1) = (f(y_0 + v) - f(y_0))/v$ and $f'(\theta_2) = (f(x_0) - f(x_0 - v))/v$. Since f is convex (so $f''(x) \geq 0$) and $\theta_1 \leq y_0 + v < x_0 - v \leq \theta_2$, we have $f'(\theta_1) < f'(\theta_2)$, proving the claim.

If on the other hand $v \geq (x_0 - y_0)/2$, then define $w = x_0 - (y_0 + v) < (x_0 - y_0)/2$ and note that $x_0 - w = y_0 + v$ and $y_0 + w = x_0 - v$. The claim $f(x_0) - f(y_0 + v) > f(x_0 - v) - f(y_0)$ can now be shown exactly as above. □

Lemma 8. *For any schedule S, moving a job from a machine where it is assigned together with a set of jobs of total size at least W/m to a machine with minimum load strictly improves the ℓ_p norm.*

Proof. Denote the schedule after the move by S'. We show that $\sum_{i=1}^m f(L_i(S')) < \sum_{i=1}^m f(L_i(S))$, where $f(x) = x^p$ (for some $p > 1$). Denote the size of the job to be moved by $v > 0$, the current load of its machine by x_0, where $x_0 - v \geq W/m$ by assumption, and the current minimum load by $y_0 < W/m$. Now we can apply Lemma 7. □

The following corollary follows from Lemma 8.

Corollary 2. *For any schedule S, $||S||_p \geq (\sum_{i=1}^m (W/m)^p)^{1/p}$.*

Proof. We apply Lemma 8 repeatedly (if possible, i.e. if the load is not already exactly W/m on every machine) and also allow parts of jobs to be moved (everything that is above a load of W/m on some machine). Eventually we reach a flat schedule with a load of W/m everywhere, and the ℓ_p norm is improved in every step. □

Lemma 9. *In any optimal schedule S, any job with processing time greater than $\frac{W}{m}$, i.e. a huge job, will be scheduled on a machine without any other jobs.*

Proof. There can be at most $m - 1$ huge jobs, else the total processing time of the jobs would be more than W. In a schedule with a huge job, the machine with the minimum load must have a load less than $\frac{W}{m}$ (and cannot contain a huge job), else the total processing time of the jobs would be more than W.

If, in the optimal schedule, there is a huge job scheduled with other jobs, we can move these jobs, one by one, to the machine with minimum load. By Lemma 8, this process decreases the ℓ_p norm, contradicting that we started with an optimal schedule. □

Lemma 10. *In any optimal schedule S, there are at most $\frac{1}{\varepsilon}$ non-small jobs scheduled on each machine.*

Proof. By Lemma 9, in an optimal schedule, any machine with a huge job will have only one job.

In S, let machine i be the machine with the minimum load and assume that some machine $j \neq i$ has more than $\frac{1}{\varepsilon}$ large jobs. The load of j is at least $(1+\varepsilon)\frac{W}{m}$ and, hence, the load of i is strictly less than $\frac{W}{m}$. This implies that i has less than $\frac{1}{\varepsilon}$ large jobs. By Lemma 8, moving a large job from j to i will decrease the ℓ_p norm, contradicting that S is an optimal schedule. □

From Lemma 10, $v = \frac{1}{\varepsilon}$. Using this value with Fact 4 of the general framework, gives the following.

Fact 8. *The online algorithm with advice, based on the general framework, uses at most $\frac{2}{\varepsilon}\left(\log\left(3\frac{\log(1/\varepsilon)}{\log(1+\varepsilon)}\right)\right)+4$ bits of advice per request.*

Theorem 5. *Given a request sequence σ, $U = \frac{W}{m}$ and an ε, $0 < \varepsilon < 1/2$, the general framework schedules the jobs of σ such that the resulting schedule has an ℓ_p norm of at most $(1+2\varepsilon)\text{OPT}$.*

Proof. By Lemma 9, Assumption 1 holds and Theorem 2 applies.

The algorithm schedules the jobs such that

$$\begin{aligned}
\|L(S)\|_p &= \left(\sum_{i=1}^{m} (L_i(S))^p\right)^{1/p} \\
&\leq \left(\sum_{i=1}^{m} \left((1+\varepsilon)L_i(S^*) + \varepsilon\frac{W}{m}\right)^p\right)^{1/p} && \text{by Theorem 2} \\
&\leq \left(\sum_{i=1}^{m} ((1+\varepsilon)L_i(S^*))^p\right)^{1/p} + \left(\sum_{i=1}^{m}\left(\varepsilon\frac{W}{m}\right)^p\right)^{1/p} \\
&\leq (1+\varepsilon)\text{OPT} + \varepsilon\text{OPT} && \text{by Corollary 2} \\
&= (1+2\varepsilon)\text{OPT}\ ,
\end{aligned}$$

where we have used the Minkowski inequality in the third line. □

5. Lower Bound for Scheduling

Boyar et al. [10] showed that at least $(n-2N)\log N$ bits of advice in total (i.e. at least $\left(1-\frac{2N}{n}\right)\log N$ bits per request) are needed for any online bin packing algorithm with advice to be optimal. Using a similar technique, we show that $(n-2m)\log m$ bits of advice in total (at least $\left(1-\frac{2m}{n}\right)\log m$ bits of advice per request) are required for any online scheduling algorithm with advice on m identical machines to be optimal for makespan, machine cover or the ℓ_p norm.

Let

$$\begin{aligned} k &= n - 2m, \\ \sigma_1 &= \left\langle \frac{1}{2^{k+2}}, \frac{1}{2^{k+3}}, \ldots, \frac{1}{2^{k+m+1}}, \frac{1}{2^2}, \ldots, \frac{1}{2^{k+1}} \right\rangle \text{ and} \\ \sigma_2 &= \langle x_1, x_2, \ldots, x_m \rangle \ , \end{aligned}$$

where x_i will be defined later in an adversarial manner. The entire adversarial request sequence will be $\sigma = \langle \sigma_1, \sigma_2 \rangle$. This sequence will be chosen such that the adversary will have a balanced schedule (a load of 1 on each machine) while any algorithm using less than $k \log m$ bits of advice will not. That is, such an algorithm will have at least one machine with load greater than 1, and, hence, at least one machine with load less than 1. Such an algorithm will, therefore, not be optimal for makespan, machine cover or the ℓ_p norm.

Fact 9. *Every subset of the requests of σ_1 has a unique sum that is less than 1/2.*

Let T be the set of all possible schedules on m identical machines for the requests of σ_1. The adversary will schedule each of the first m requests of σ_1 on a distinct machine. This distinguishes the m machines from one another. Let V be the set of all possible schedules of the last k requests of σ_1 onto the m machines, given that the first m requests of σ_1 were each scheduled on a distinct machine. Note that $V \subset T$ and that $|V| = m^k$. Let $S_{\sigma_1}^{\text{ADV}} \in V$ be the adversarial schedule of the jobs of σ_1. Define $x_i = 1 - L_i(S_{\sigma_1}^{\text{ADV}})$. Note that using Fact 9 we have that the m values x_i, $1 \leq i \leq m$, are distinct. Further, note that σ allows for a balanced schedule, where all machines have load 1.

Observation 1. *For every $S_{\sigma_1} \in T \setminus S_{\sigma_1}^{\text{ADV}}$, every possible scheduling of the jobs of σ_2 into S_{σ_1} results in a schedule S_σ such that there are at least 2 machines i and j, where $L_i(S_\sigma) < 1$ and $L_j(S_\sigma) > 1$.*

Proof. The sum of the processing times of all jobs of σ_1 is less than 1/2 which implies that the processing time for each x_i is greater than 1/2. Therefore, any machine that schedules more than one job from σ_2 will have a load greater than 1. It follows that such a schedule also has a machine that does not have any job from σ_2, and, hence, has a load less than 1. We therefore consider a schedule S_σ that schedules a single job from σ_2 on each machine.

Since the sum of the processing times of all the jobs of σ is m, note that if we have a machine with a load greater than 1 then there must be a machine with load less than 1. We can therefore assume by contradiction that in S_σ all machines have a load exactly 1. As each job x_i of σ_2 is scheduled on a distinct machine, we have that in S_σ the total processing time of the jobs from σ_1 on the machine that has job x_i is exactly $1 - x_i$. Fact 9 implies that S_{σ_1} equals $S_{\sigma_1}^{\text{ADV}}$, a contradiction. □

We are now ready to prove the main theorem of the section.

Theorem 6. *Any online algorithm with advice needs at least $(n - 2m)\log m$ bits of advice in order to be optimal for the makespan problem, machine cover problem and the ℓ_p norm problem, where m is the number of machines and n is the length of the request sequence.*

Proof. Let ALG be an arbitrary (deterministic) online algorithm with advice for the given scheduling problem. Let S_{σ_1} be the schedule produced by ALG for σ_1. If $S_{\sigma_1} \in T \setminus V$, i.e. S_{σ_1} is such that the first m requests are not scheduled on distinct machines, then $S_{\sigma_1} \neq S^{\text{ADV}}_{\sigma_1}$, and, by Observation 1, S_σ is not balanced. Therefore, we will assume that the algorithm will schedule the first m requests on m distinct machines, i.e. $S_{\sigma_1} \in V$.

Assume that the online algorithm with advice receives all the advice bits in advance. This only strengthens the algorithm and, thus, strengthens our lower bound. Let $\text{ALG}(s, u)$ be the schedule produced by ALG for request sequence s when receiving advice bits u. Since ALG gets less than $k \log m$ bits of advice, it gets as advice some $u \in U$ for some advice space U, $|U| < m^k$. It follows that $|\{\text{ALG}(\sigma_1, u) | u \in U\}| < m^k = |V|$. Therefore, given ALG, $S^{\text{ADV}}_{\sigma_1}$ is chosen by the adversary such that $S^{\text{ADV}}_{\sigma_1} \in T \setminus \{\text{ALG}(\sigma_1, u) | u \in U\}$. Note that this choice defines σ_2.

We now have, by Observation 1, that S_σ has at least 2 machines i and j such that $L_i(S_\sigma) < 1$ and $L_j(S_\sigma) > 1$. Given that there is a balanced schedule with all machines having load 1 for σ, S_σ is not optimal for makespan due to machine j, S_σ is not optimal for machine cover due to machine i, and S_σ is not optimal for the ℓ_p norm by Corollary 2. □

6. Comparison to the Semi-Online Advice Model

For a request sequence of length n, the naïve conversion of the algorithms described previously from the online advice model to the semi-online advice model uses less than a total of $n\left(\frac{1}{\varepsilon}\log\left(\frac{2}{\varepsilon^2}\right) + \log\left(\frac{2}{\varepsilon^2}\right) + 3\right)$ bits of advice for bin packing and $n\left(\log\left(3\log_{1+\varepsilon} 1/\varepsilon\right) + \beta(v) + 3\right)$ for scheduling. It is possible, as we describe below, to do better in the more powerful semi-online model, but the amount of advice is still linear in n. This follows from the observation that only for the first N (m) request does the advice include an actual bin (machine) pattern.

6.1. BPA *in the semi-online advice model*

Initially, a single bit w (as described above) is written to the advice tape to indicate if $N \leq 1/\varepsilon$ or not. If so, $n\log(1/\varepsilon)$ bits are written to the tape to indicate the bin index in which to pack each item.

If $N > 1/\varepsilon$, the optimal number of bins N is encoded, using a self-delimiting encoding scheme [7], and written to the advice tape, using $\lceil\log N\rceil + 2\lceil\log\lceil\log N\rceil\rceil$ bits. Then, for each of the N optimal bins, the bin pattern is written, using $p < (1/\varepsilon)\log(2/\varepsilon^2) + 1$ bits, followed by a bit to indicate if small items are packed in the bin.

For each request, a bit is written to the advice tape to indicate if the item is small or large. If the requested item is small, an additional bit is written to indicate if the small item should be packed in the current bin packing small items or the next. If the requested item is large, the item type is written, using $t < \log(2/\varepsilon^2) + 1$ bits, and an additional bit is written to indicate if the large item is packed in a bin with or without small items.

The total amount of advice used is less than

$$1 + \lceil \log N \rceil + 2\lceil \log \lceil \log N \rceil \rceil + N \left(\frac{1}{\varepsilon} \log \left(\frac{2}{\varepsilon^2} \right) + 1 \right) + N + n \left(\log \left(\frac{2}{\varepsilon^2} \right) + 2 \right) .$$

In [27], it is shown that a linear amount of advice in total is required for an algorithm with advice for the bin packing problem to achieve a competitive ratio of 7/6. Therefore, the algorithms described here uses an optimal amount of advice (up to constant factors) for competitive ratios at most 7/6.

6.2. Scheduling framework in the semi-online advice model

As with BPA, the machine pattern of each machine and the machines that have small items scheduled can be written to the front of the advice tape. In this case, m unlike N is known to algorithm and does not need to be written to the advice tape.

Initially, the m machine patterns, ordered according to the permutation of the machines as described previously, are written on the advice tape using $p < m\beta(v)$ bits. Then, for each request, the type of each job is written to the advice tape, using $t < \log\left(3\log(1/\varepsilon)/\log(1+\varepsilon)\right) + 1$ bits. If the job is small, then an additional bit is written to indicate if the small job should be scheduled on the previous machine or the current machine.

The total amount of advice used is less than

$$m\beta(v) + n \left(\log \left(\frac{3\log(1/\varepsilon)}{\log(1+\varepsilon)} \right) + 2 \right) .$$

The framework that we presented here works for any objective function of the form $\sum_{i=1}^{m} f(L_i)$ such that if $x \leq (1+\varepsilon)y$ then $f(x) \leq (1+O(1)\varepsilon)f(y)$ (which includes makespan, machine cover and the ℓ_p norm). In [26], an algorithm that uses constant advice in total and achieves a competitive ratio of $1+\varepsilon$ is presented for the makespan objective on m identical machines. It remains open whether or not it is possible to improve on the framework here for other objective functions such as machine cover and the ℓ_p norm.

7. Conclusions

We gave online algorithms with advice for bin packing and scheduling problems that, with a constant number of bits per request, achieve competitive ratios arbitrarily close to 1. Since this is not possible for all online problems, it would be interesting to prove similar results for additional online problems. Furthermore, an interesting question is to find the right trade-off between the (constant) number of bits of advice and the achievable competitive ratios for the problems we study and other problems.

Acknowledgements. We would like to thank Shahin Kamali and Alejandro López-Ortiz for useful discussions about the bin packing problem.

References

[1] A. Borodin, R. El-Yaniv, Online computation and competitive analysis, Cambridge University Press, New York, NY, USA, 1998.

[2] R. Dorrigiv, A. López-Ortiz, A survey of performance measures for on-line algorithms, SIGACT News 36 (3) (2005) 67–81.

[3] H.-J. Böckenhauer, D. Komm, R. Královic, R. Královic, T. Mömke, On the advice complexity of online problems, in: ISAAC, 2009, pp. 331–340.

[4] Y. Emek, P. Fraigniaud, A. Korman, A. Rosén, Online computation with advice, Theor. Comput. Sci. 412 (24) (2011) 2642–2656.

[5] H.-J. Böckenhauer, D. Komm, R. Královic, R. Královic, On the advice complexity of the k-server problem, in: ICALP, 2011, pp. 207–218.

[6] D. Komm, R. Královic, Advice complexity and barely random algorithms, RAIRO - Theor. Inf. and Applic. 45 (2) (2011) 249–267.

[7] H. Böckenhauer, D. Komm, R. Královic, P. Rossmanith, The online knapsack problem: Advice and randomization, Theor. Comput. Sci. 527 (2014) 61–72. doi:10.1016/j.tcs.2014.01.027.
URL `http://dx.doi.org/10.1016/j.tcs.2014.01.027`

[8] M. P. Renault, A. Rosén, On online algorithms with advice for the k-server problem, Theory Comput. Syst. 56 (1) (2015) 3–21. doi:10.1007/s00224-012-9434-z.
URL `http://dx.doi.org/10.1007/s00224-012-9434-z`

[9] R. Dorrigiv, M. He, N. Zeh, On the advice complexity of buffer management, in: ISAAC, 2012, pp. 136–145.

[10] J. Boyar, S. Kamali, K. S. Larsen, A. López-Ortiz, Online bin packing with advice, in: E. W. Mayr, N. Portier (Eds.), 31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014), STACS 2014, March 5-8, 2014, Lyon, France, Vol. 25 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014, pp. 174–186. doi:10.4230/LIPIcs.STACS.2014.174.
URL `http://dx.doi.org/10.4230/LIPIcs.STACS.2014.174`

[11] S. Gupta, S. Kamali, A. López-Ortiz, On advice complexity of the k-server problem under sparse metrics, in: T. Moscibroda, A. A. Rescigno (Eds.), SIROCCO, Vol. 8179 of Lecture Notes in Computer Science, Springer, 2013, pp. 55–67.

[12] A. Adamaszek, M. P. Renault, A. Rosén, R. van Stee, Reordering buffer management with advice, in: C. Kaklamanis, K. Pruhs (Eds.), Approximation and Online Algorithms - 11th International Workshop, WAOA 2013, Sophia Antipolis, France, September 5-6, 2013, Revised Selected Papers, Vol. 8447 of Lecture Notes in Computer Science, Springer, 2013, pp. 132–143. doi:10.1007/978-3-319-08001-7.
URL `http://dx.doi.org/10.1007/978-3-319-08001-7`

[13] W. Fernandez de la Vega, G. S. Lueker, Bin packing can be solved within 1+epsilon in linear time, Combinatorica 1 (4) (1981) 349–355.

[14] D. S. Hochbaum, D. B. Shmoys, Using dual approximation algorithms for scheduling problems theoretical and practical results, J. ACM 34 (1) (1987) 144–162. doi:10.1145/7531.7535.

[15] B. Chen, A. van Vliet, G. J. Woeginger, A lower bound for randomized on-line scheduling algorithms, Information Processing Letters 51 (5) (1994) 219 – 222. doi:10.1016/0020-0190(94)00110-3.

[16] N. Alon, Y. Azar, G. Woeginger, T. Yadid, Approximation schemes for scheduling, in: SODA, 1997, pp. 493–500.

[17] J. Sgall, A lower bound for randomized on-line multiprocessor scheduling, Inf. Process. Lett. 63 (1) (1997) 51 – 55. doi:10.1016/S0020-0190(97)00093-8.

[18] G. J. Woeginger, A polynomial-time approximation scheme for maximizing the minimum machine completion time, Oper. Res. Lett. 20 (4) (1997) 149 – 154. doi:10.1016/S0167-6377(96)00055-7.

[19] A. Avidor, Y. Azar, J. Sgall, Ancient and new algorithms for load balancing in the lp norm, in: SODA, 1998, pp. 426–435.

[20] Y. Azar, L. Epstein, On-line machine covering, J Scheduling 1 (2) (1998) 67–77.

[21] R. Fleischer, M. Wahl, Online scheduling revisited, in: ESA, 2000, pp. 202–210.

[22] S. S. Seiden, On the online bin packing problem, J. ACM 49 (2001) 2002.

[23] S. Albers, On randomized online scheduling, in: STOC, ACM, 2002, pp. 134–143.

[24] J. F. Rudin, III, R. Chandrasekaran, Improved bounds for the on-line scheduling problem, SIAM J. Comput. 32 (3) (2003) 717–735. doi:10.1137/S0097539702403438.

[25] J. Balogh, J. Békési, G. Galambos, New lower bounds for certain classes of bin packing algorithms, TCS 440-441 (0) (2012) 1–13.

[26] J. Dohrau, Online makespan scheduling with sublinear advice, in: G. F. Italiano, T. Margaria-Steffen, J. Pokorný, J. Quisquater, R. Wattenhofer (Eds.), SOFSEM 2015: Theory and Practice of Computer Science - 41st International Conference on Current Trends in Theory and Practice of Computer Science, Pec pod Sněžkou, Czech Republic, January 24-29, 2015. Proceedings, Vol. 8939 of Lecture Notes in Computer Science, Springer, 2015, pp. 177–188. doi:10.1007/978-3-662-46078-8_15.
URL `http://dx.doi.org/10.1007/978-3-662-46078-8`

[27] S. Angelopoulos, C. Dürr, S. Kamali, M. Renault, A. Rosén, Online bin packing with advice of small size, in: Algorithms and Data Structures - 14th International Symposium, WADS 2015, Victoria, BC, Canada, August 5-7, 2015. Proceedings, to appear, pp. 1–12.

[28] B. Chandra, Does randomization help in on-line bin packing?, Inf. Process. Lett. 43 (1) (1992) 15–19. doi:10.1016/0020-0190(92)90023-O.

[29] D. Johnson, Near-optimal bin packing algorithms, Ph.D. thesis, MIT (1973).