Online Algorithms with Advice for Bin Packing and Scheduling Problems

Marc P. Renault\textsuperscript{1,*}, Adi Rosén\textsuperscript{2,*}, and Rob van Stee\textsuperscript{3}

\textsuperscript{1} Université Paris Diderot - Paris 7, France. mrenault@liafa.univ-paris-diderot.fr
\textsuperscript{2} CNRS and Université Paris Diderot - Paris 7, France. adiro@liafa.univ-paris-diderot.fr
\textsuperscript{3} Max Planck Institute for Informatics, Saarbrücken, Germany.

Present address: University of Leicester, Department of Computer Science, UK.
rob.vanstee@leicester.ac.uk

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 \(\ell_p\) 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 showing that for any online algorithm with advice to be optimal, for any of the above scheduling problems, a non-constant number (namely, at least \((1 - \frac{2}{m}) \log m\), where \(n\) is the number of jobs and \(m\) is the number of machines) of bits of advice per request is needed.

1 Introduction

Online algorithms are algorithms that receive their input one piece at a time. The 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 Analysis (cf. \cite{10}). In this framework, the decisions of the online algorithm must be taken with zero 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 zero knowledge of the future and full knowledge of the future.

In many situations, however, an algorithm with zero knowledge of the future is unreasonably restrictive (\cite{10,15}). 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, that attempts to analyze online algorithms with partial information about the future, has been proposed and studied in recent years, e.g. \cite{9,12,14,11}.

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

* Research supported in part by ANR project NeTOC.
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 \textit{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. [16]. Another variant of the setting of online algorithms with advice was proposed by Böckenhauer et al. [8] (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 [16,21], Metrical Task Systems [15], knapsack problem [9], bin packing problem [11], 2 value buffer management [14], reordering buffer management problem [1] 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 \( \ell_p \) 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. [25,18,13,24,26,4,5,17,23,2,22,6]. 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 \((1 + \epsilon)\)-competitive deterministic algorithms with advice that use \( f(1/\epsilon) \) 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 are known for some online problems (e.g. for Metrical Task Systems [16]). 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 [11]).

1.1 Related Work.

The model of online computation with advice that we consider in the present paper was introduced by Emek et al. [16]. In the model of [16], the advice is a fixed amount that is revealed in an online manner with every request. This model is referred to as the \textit{online advice model}. Another variant of the model of online algorithms with advice was proposed by Böckenhauer et al. [8]. 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 \textit{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 \( \left\lceil \frac{\log k}{b-2} \right\rceil \) [21], where \( b \) is the number of bits of advice per request; the Metrical Task System problem has tight bounds on the competitive ratio of \( \Theta(\log N/b) \) [15]; 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 [9], 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) [14], and the reordering buffer problem, for any \( \epsilon > 0 \), has a \((1 + \epsilon)\)-competitive algorithm which uses only a constant (depending on \( \epsilon \)) number of advice bits per input item [1].

To the best of our knowledge, the only scheduling problem studied to date in the framework of online computation with advice is a special case of the job shop scheduling problem [8,20]. Boyar et al. [11] study the bin packing problem with advice, using the semi-online advice model of [8] and present a \( 3/2 \)-competitive algorithm, using \( \log n \) bits of advice in total and a \((4/3 + \epsilon)\)-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. Further, they show that, in total, linear advice is required for a competitive ratio better than 5/4 (in the semi-online advice model; in the online advice model, an algorithm has 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 to be optimal, where $N$ is the optimal number of bins.

For online bin packing without advice, the best known lower bound on the competitive ratio is 1.54037 due to Balogh et al. [6] and the best known deterministic upper bound on the competitive ratio is 1.58889 due to Seiden [23]. Chandra [12] 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 [22] presented the best known deterministic lower bound on the competitive ratio for minimizing the makespan of 1.88. The best known deterministic upper bound on the competitive ratio for minimizing the makespan, due to Fleischer et al. [17], 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 $m \to \infty$ was proved independently by Chen et al. [13] and Sgall [24] and the best known randomized algorithm, due to Albers [2], has a competitive ratio of 1.916.

For machine covering, Woeginger [26] proved tight $O(m)$ bounds on the competitive ratio for deterministic algorithms, and Azar and Epstein [5] 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 [25] presented an asymptotic polynomial time approximation scheme (APTAS) for the bin packing problem. Hochbaum and Shmoys [18] developed a polynomial time approximation scheme (PTAS) for the makespan minimization problem on $m$ identical machines. Subsequently, Woeginger [26] presented a PTAS for the machine covering problem on $m$ identical machine and Alon et al. [3] presented a PTAS for the $\ell_p$ 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 $\ell_p$ 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 $\left( 1 - \frac{2m}{\pi} \right) \log m$ bits of advice per request to be optimal, where $n$ is the number of jobs and $m$ is the number of machines. 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 is the technique of classifying the input items, according to their size, into a constant number (depending on $\varepsilon$) of classes. The sizes of the items in one class differ only by a constant factor (depending on $\varepsilon$). We classify all the items but the smallest ones in this way, where the upper bound on the size of the smallest items again depends on $\varepsilon$ (this is done explicitly in the scheduling algorithms, and implicitly in the bin packing algorithm). We then consider an optimal schedule (or packing) for the input sequence and define machine (or bin, for
the bin packing problem) patterns for the machines. We use the advice bits to indicate with each input item into which machine type (or pattern) it should be scheduled. Since items in the same class are “similar” in size, we are able to circumvent the fact that we do not know exactly which item is scheduled on which machine. The very small items (and the very big items in the case of bin packing) have to be treated separately. We then employ a number of additional ideas and techniques to make this general idea feasible in the online setting.

Some of our techniques are similar to those of [25,18,26,3]. In particular, we use the technique of rounding and grouping the jobs. 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 should not depend on the size of the input. This means for instance that the online algorithm cannot discover the size of the input from the advice, at least not initially.

2 Preliminaries

Throughout the 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 [16]. A deterministic online algorithm with advice is defined by the sequence of pairs \((g_i, u_i), i \geq 1\). The functions \( u_i : R^* \rightarrow U \) 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 \( b \geq 0 \). The advice received with each request \( i \) is the value of the function \( u_i \) on the whole request sequence \( \sigma \in R^* \). The functions \( g_i : R^i \times U^i \rightarrow A_i \) 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, \( a_j \), of the online algorithm with advice is \( g_j(r_1, \ldots, r_j, u_1, \ldots, u_j) \), i.e. a function of the requests and advice received to date.

2.2 Competitive Analysis.

Let \( ALG(\sigma) \) be the cost for an online algorithm \( ALG \) to process \( \sigma \) 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 \( \sigma \) \( ALG(\sigma) \leq c \cdot \text{OPT}(\sigma) + \zeta \), where \( \zeta \) is a constant that does not depend on \( \sigma \). For a maximization problem, an algorithm \( ALG \) is \( c \)-competitive if \( ALG(\sigma) \geq 1/c \cdot \text{OPT}(\sigma) - \zeta \).

2.3 Bin Packing.

The online bin packing problem consists of a request sequence \( \sigma \), 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 \( \sigma \) 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. \( N \) is used to denoted the optimal number of bins \( |B^{OPT}| \). A bin is valid if the total size of the packed items in the bin is at most 1; a packing is valid if all the bins in a packing are valid. For an item \( r_i \in b_j \), where \( b_j \) is a bin in the packing \( B \), we will write \( r_i \in B \). We use in our algorithms a common heuristics for bin packing: next fit (cf. [19]). The algorithm is an online algorithm which considers each item once and packs it irrevocably. An item fits into a bin if its size plus the size of previously packed items in that bin is at most the size of the bin. 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.

The online scheduling problem on m identical machines consists of m identical machines and a request sequence $\sigma$. 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 $S$, $L_i(S) = \sum_{r_j \in M_i} v(r_j)$ will denote 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;
- $\ell_p$ norm: minimizing the $\ell_p$ norm, $1 < p \leq \infty$, of the load of all the machines. For a schedule $S$, the $\ell_p$ norm is defined to be $\|L(S)\|_p = (\sum_{i=1}^{m}(L_i(S))^p)^{1/p}$. Note that minimizing the $\ell_\infty$ norm is equivalent to minimizing the makespan.

3 Online Algorithms with Advice for Bin Packing

In this section, we present an online bin packing algorithm with advice called BPA. The main result of this section is the following.

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

Initially, we will present an algorithm, ABPA that uses less than $\frac{1}{\varepsilon} \log \left(\frac{1}{\varepsilon}\right) + \log \left(\frac{1}{\varepsilon}\right) + 3$ 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.

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

We begin by creating a rounded input $\sigma'$ based on $\sigma$ using the scheme of Fernandez de la Vega and Lueker [25]. 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.

An item is called large if it has size more than $\varepsilon$. Items with size at most $\varepsilon$ are called small items. Let the number of large items in $\sigma$ be $L$.

Sort the large items of $\sigma$ in order of nonincreasing size. Let $h = \varepsilon^2 L$. For $i = 0, \ldots, 1/\varepsilon^2 - 1$, assign the items $ih + 1, \ldots, ih + [\varepsilon^2 L]$ 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 $[\varepsilon^2 L]$ 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 $\sigma'$ be the subsequence of $\sigma$ restricted to the large items with their sizes rounded up in the manner previously described.

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 $\varepsilon$, $N \geq \varepsilon L$. This implies the following fact.

**Fact 1.** $|B_1| \leq \varepsilon N$

For the remaining large items, i.e. types 2 to $1/\varepsilon^2$, in $\sigma'$, a packing, $B'_2$ that uses at most $N$ bins can be found efficiently [25]. The packing of each bin $b_i \in B'_2$ can be described by a vector of length at most $1/\varepsilon$, denoted $p_i$, where each value in the vector ranges from 1 to $1/\varepsilon^2$ representing the type of each of the at most $1/\varepsilon^2$ 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 $\sigma'$ 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 describe in $p_j$. If no such bin
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 \( \sigma' \) is processed sequentially. That is, for \( i < j \) and every \( b_i, b_j \in S' \), there exists an \( r_j \in b_j \) such that, for all \( r_q \in b_j, p < q \). From Fact 1 and that \( |B_2| \leq N \), we have the following fact.

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

We now extend \( S' \) to include the small items and define \( S \). Sequentially, by the order 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 possible the last one, are filled to at least \((1 - \varepsilon)\). Since the total size of the items is at most \( N \), we have \(|S| \leq (1 - \varepsilon)N \) and, therefore, \( |S| \leq \frac{N}{1 - \varepsilon} + 1 \leq (1 + 2\varepsilon)N + 1 \).

We now define abpa. It is defined given a \( \sigma \), and an \( \varepsilon, 0 < \varepsilon \leq 1/2 \). 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. BPA 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 BPA, 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 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 for a formal definition) is defined so that \( \text{BPA} \) will schedule the same number of small items on each bin as \( S \). Further, \( \text{BPA} \) schedules both the small and large jobs in the order the arrive on the least recently opened bin just as \( S \) which implies the following fact.

**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 [I]

**Formal Advice Definition.**

*Item 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. For the enumeration, item pattern vectors with length less than \( 1/\varepsilon \) will be padded to a length \( 1/\varepsilon \) with addition entries with a new value of \( \perp \).

The algorithm requires less than \( \left\lceil \frac{1}{2} \log \left( \frac{\varepsilon}{2} + 1 \right) \right\rceil < \frac{1}{2} \log \left( \frac{\varepsilon}{2^3} \right) + 1 \) bits of advice per request to encode the index of the item pattern from an enumeration of all possible item patterns in binary.

**Advice per Request** In order to define the advice, for each bin \( b_i \in S \), we define a value \( \kappa_i \) 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 \( 1/\varepsilon \log \left( \frac{\varepsilon}{2^3} \right) + 1 \) 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 and 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.
- **y:** \( r_i \) 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 \( z \) is 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.

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

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

**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 \( \left\lceil \log(1/\varepsilon) \right\rceil \) bits of advice per request define the bin number in which \( r_i \) is packed in an optimal packing. \( \text{BPA} \) will pack \( r_i \) into the bin as specified by the advice. This case requires less than \( \log(1/\varepsilon) + 2 < 1/\varepsilon \log \left( \frac{\varepsilon}{2^3} \right) + \log \left( \frac{\varepsilon}{2^3} \right) + 3 \) (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 [I] and Corollary [I] prove Theorem [I]
4 Online Algorithms with Advice for Scheduling

In this section, we present a general online framework for approximating an optimal schedule, using advice. We apply the general framework to the problems of minimizing the makespan, maximizing machine cover and minimizing the \( \ell_p \) norm. For the minimization problems, we get algorithms with a competitive ratio of \( 1 + \varepsilon \) and, for the maximization problems, a competitive ratio of \( 1/(1-\varepsilon) \).

The amount of advice per request is \( O(1/\varepsilon \log 1/\varepsilon) \) for \( 0 < \varepsilon < 1/2 \). We note that with \( \log m \) bits of advice per request, the trivial algorithm, that indicates the machine in which to schedule the item, would be optimal.

4.1 General Framework

The machines are numbered from 1 to \( m \). Given an \( \varepsilon \), \( 0 < \varepsilon < 1/2 \), 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 \( (\varepsilon(1+\varepsilon)^i U, (1+\varepsilon)^{i+1} U) \) for \( i \in [0, \lceil \log_{1+\varepsilon} \frac{1}{\varepsilon} \rceil] \). These jobs will be called large jobs. Jobs with processing times at most \( \varepsilon U \) 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, \( U \), or the value of \( \varepsilon \).

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 all the objective functions that we consider, we will show that there exist optimal schedules for which this assumption holds.

**Assumption 1.** \( S^* \) is an optimal schedule such that each huge job is scheduled on a machine without any other jobs.

The general framework is defined given a \( \sigma \), an \( \varepsilon \), 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 serial number 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 type 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.

**Lemma 2.** The small jobs of \( S^* \) 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 size of the \( j \)th small job in this order by \( x_j \), \( j = 1, \ldots \). For \( i = 1, \ldots, m \), let \( y_i \) be the total size of small jobs assigned to machine \( i \) in \( S^* \). Let \( i(0) = 0 \), and for \( k = 1, \ldots, m \), let \( i(k) \) be the minimum index such that \( \sum_{j=1}^{i(k)} x_j \geq \sum_{i=1}^{m} y_i \). Finally, for \( k = 1, \ldots, m \), assign the small jobs \( i(k-1) + 1, \ldots, i(k) \) to machine \( k \). (If \( i(k) = i(k-1) \), machine \( k \) receives no small jobs.)

By definition of \( i(k) \) and the fact that all small jobs have size at most \( \varepsilon U \), the total size of small jobs assigned to machines \( 1, \ldots, k \) is in \([\sum_{i=1}^{k} y_i, \sum_{i=1}^{k} y_i + \varepsilon U]\) for \( k = 1, \ldots, m \). By taking
the difference between the total assigned size for the first \( k - 1 \) and for the first \( k \) machines, it immediately follows that the total size 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 item \( r_i \in \sigma \), in the order they appear in \( \sigma \), assign \( r_i \) with type \( t_i \) to the first machine in \( S \) such that the number of items with of type \( t_i \) currently scheduled is less than the number of items 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 \( 1 \leq i \leq m \), \( L_i(S) \in \left[(1 - \varepsilon)L_i(S^*) - \varepsilon U, (1 + \varepsilon)L_i(S^*) + \varepsilon U\right] \)

**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. 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 \( \sigma \), the type of \( r_i \), denoted \( t_i \), is received as advice. The framework schedules \( r_i \), according to \( t_i \), as follows:

**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** (\( 0 \leq t_i \leq \lceil \log_{1 + \varepsilon} \frac{1}{\varepsilon} \rceil \)). The algorithm schedules \( r_i \) on a machine where the number of items of type \( t_i \) is less than the number indicated by its pattern.

**Formal advice definition.**

**Machine Patterns** For the first \( m \) request, 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 item 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 $\beta(v) \leq \left\lceil \log(2 + \left\lceil \log_{1+\varepsilon} \frac{1}{\varepsilon} \right\rceil) \right\rceil + 1$ bits are required to encode the index of a machine pattern in an enumeration of all possible machine patterns in binary. For the cases of makespan, machine cover and $\ell_p$ norm, $v \leq \frac{1}{\varepsilon} + 1$ and $\beta(v) < \frac{1}{1+\varepsilon} \log \left( \frac{4\log(1/\varepsilon)}{\log(1+\varepsilon)} \right) + 1$ bits are required to encode the index of a machine pattern in an enumeration of all possible machine patterns in binary.

Advice per Request In order to define the advice, for each machine $m_i \in S$, we define a value $\kappa_i$ that is the number of small items scheduled on $m_i$. Per request, the advice string will be $wxyz$, where $w$ has a length of $\left\lceil \log(2 + \left\lceil \log_{1+\varepsilon} \frac{1}{\varepsilon} \right\rceil) \right\rceil + 1$ 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 $\log \left( \frac{4\log(1/\varepsilon)}{\log(1+\varepsilon)} \right) + \beta(v) + 4$ bits of advice per request.

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

Theorem 2. For any $\sigma$, an $\varepsilon$, $0 < \varepsilon < 1/2$, and a $U > 0$ such that there exists an $S^*$ under Assumption 1, the general framework schedules $\sigma$ 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 $U = \text{OPT}$, where OPT is the minimum makespan for $\sigma$.

Fact 5. If $U = \text{OPT}$, 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 a machine in $S^*$ with more than $\frac{1}{\varepsilon}$ jobs with processing times greater than $\varepsilon \text{OPT}$ is more than the maximum makespan, a contradiction.

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
\( \frac{2}{\varepsilon} \left( \log \left( \frac{4 \log(1/\varepsilon)}{\log(1+\varepsilon)} \right) \right) + 5 \) bits of advice per request.

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

Proof. By Fact 5 Assumption 4 holds and Theorem 2 applies.

Lemma 5. There exists an optimal schedule \( S^* \) such that any job with processing time at least 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 and 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 nonhuge 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 \( S \) 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 \( \text{OPT} \) as it contains a huge job and the load on machine \( i \) in \( S^* \) is greater than \( \text{OPT} \) given that it was \( \text{OPT} \) 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 items 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 items. 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 \( \frac{1}{\varepsilon} \) jobs scheduled and, since its load is \( U \), it cannot contain a huge job which 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 \( \text{OPT} \) as it has at least \( 1 + \frac{1}{\varepsilon} \) large items scheduled on it and the load on machine \( i \) in \( S^* \) is greater than \( \text{OPT} \) given that it was \( \text{OPT} \) 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 items 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
\( \frac{2}{\varepsilon} \left( \log \left( \frac{4 \log(1/\varepsilon)}{\log(1+\varepsilon)} \right) \right) + 5 \) bits of advice per request.
Lemma 10. \( \text{norm} \), contradicting that we started with an optimal schedule.

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

Let \( f \) 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 \( \text{opt} = L_i(S^*) \leq L_i(S^*) \) for all \( 1 \leq i \leq m \).

4.4 The \( \ell_p \) Norm

For minimizing the \( \ell_p \) 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 \( f(x) \) 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 \( f(x) \) is convex (so \( f''(x) \geq 0 \)) and \( \theta_1 \leq y_0 < \theta_2 \), 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 \( \ell_p \) 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 \( \ell_p \) 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 be 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 \( \ell_p \) norm, contradicting that we started with an optimal schedule.

Lemma 10. In any optimal schedule \( S \), there are at most \( \frac{1}{p} \) non-small items scheduled on each machine.

12
Proof. By Lemma 9, in an optimal schedule, any machine with a huge job will have only one item.

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

From Lemma 10, $v = \frac{1}{\epsilon}$. 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 $2 \left( \log \left( \frac{3 \log(1/\epsilon)}{\log(3 + \frac{1}{\epsilon})} \right) \right) + 5$ bits of advice per request.

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

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

The algorithm schedules the jobs such that

$$
\|L(S)\|_p = \left( \sum_{i=1}^{m} (L_i(S))^p \right)^{1/p} \\
\leq \left( \sum_{i=1}^{m} \left( (1 + \epsilon)L_i(S^*) + \epsilon \frac{W_m}{m} \right)^p \right)^{1/p} \quad \text{by Theorem 2} \\
\leq \left( \sum_{i=1}^{m} (1 + \epsilon)L_i(S^*)^p \right)^{1/p} + \left( \sum_{i=1}^{m} \left( \epsilon \frac{W_m}{m} \right)^p \right)^{1/p} \quad \text{by Corollary 2} \\
= (1 + \epsilon) \text{OPT} + \epsilon \text{OPT} + \epsilon \text{OPT} + \epsilon \text{OPT} \\
= (1 + 2\epsilon) \text{OPT},
$$

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

5 Lower Bound for Scheduling

Boyar et al. [11] show that at least $(n - 2N) \log N$ bits of advice in total (i.e., at least $(1 - \frac{2N}{n}) \log N$ 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 $(1 - \frac{2m}{n}) \log m$ bits of advice per request) is required for any online scheduling algorithm with advice on $m$ identical machines to be optimal for makespan, machine cover or the $\ell_p$ norm.

Let

$$
k = n - 2m, \\
\sigma_1 = \left\{ \frac{1}{2k+2}, \frac{1}{2k+3}, \ldots, \frac{1}{2k+m+1}, \frac{1}{2l}, \ldots, \frac{1}{2l+1} \right\} \quad \text{and} \\
\sigma_2 = \langle x_1, x_2, \ldots, x_m \rangle,
$$

where $x_i$ will be defined later in an adversarial manner. The entire adversarial request sequence will be $\sigma = \langle \sigma_1, \sigma_2 \rangle$. The adversarial 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 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 $\ell_p$ norm.

**Fact 9.** Every subset of the requests of $\sigma_1$ has a unique sum that is less than $1/2$. 

13
Let $T$ be the set of all possible schedules on $m$ identical machines for the requests of $\sigma_1$. The adversary will schedule each of the first $m$ requests of $\sigma_1$ 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 $\sigma_1$ onto the $m$ distinct machines. Note that $V \subset T$ and that $|V| = m^k$. Let $S_{\sigma_1}^{adv} \in V$ be the adversarial schedule of the items of $\sigma_1$. $S_{\sigma_1}^{adv}$ will be determined later. Let $x_i = 1 - L_i(S_{\sigma_1}^{adv})$. Note that using Fact 2.10 we have that the $m$ values $x_i$, $1 \leq i \leq m$, are distinct. Further, note that $\sigma$ allows for a balanced schedule, where all machines have load 1.

**Observation 1.** For every $S_{\sigma_1} \in V \setminus S_{\sigma_1}^{adv}$, every possible scheduling of the jobs of $\sigma_2$ into $S_{\sigma_1}$ results in a schedule $S_{\sigma}$ 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 time of all jobs of $\sigma_1$ 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 $\sigma_2$ will have a load greater than 1. It follows that such a schedule also has a machine that does not have any job from $\sigma_2$, and, hence, has load less than 1. We, therefore, consider a schedule $S_{\sigma}$ that schedules a single job from $\sigma_2$ on each machine.

Since the sum of the processing time of all the jobs of $\sigma$ is $m$, note that if we have a machine with load greater than 1 then there must be a machine with load less than 1. We can therefore assume by way of contradiction that in $S_{\sigma}$ all machines have load exactly 1. As each job $x_i$ of $\sigma_2$ is scheduled on a distinct machine, we have that in $S_{\sigma}$ the total processing time of the jobs from $\sigma_1$ on the machine that has job $x_i$ is exactly $1 - x_i$. Fact 2.10 implies that $S_{\sigma}$ equals $S_{\sigma_1}^{adv}$, 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 $\ell_p$ 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 scheduling problem. Let $S_{\sigma_1}$ be the schedule produced by ALG for $\sigma_1$. If $S_{\sigma_1} \in T \setminus V$, i.e. $S_{\sigma_1}$ is such that the first $m$ requests are not scheduled on distinct machines, then, by Observation 1, $S_{\sigma}$ 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 $ALG(s, u)$ 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 $|\{ALG(\sigma_1, u) \mid u \in U\}| < m^k = |V|$. Therefore, given $ALG$, $S_{\sigma_1}^{adv}$ is chosen by the adversary such that $S_{\sigma_1}^{adv} \in T \setminus \{ALG(\sigma_1, u) \mid u \in U\}$. Note that this choice defines $\sigma_2$.

We now have, by Observation 1 that $S_{\sigma}$ 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 $\sigma$, $S_{\sigma}$ is not optimal for makespan due to machine $i$, $S_{\sigma}$ is not optimal for machine cover due to machine $j$, and $S_{\sigma}$ is not optimal for the $\ell_p$ norm by Corollary 2.2.

6 Conclusions

We give online algorithms with advice, for bin packing and scheduling problems, that, with constant number of bits per request, achieve competitive ratio 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 ratio 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. Adamaszek, A., Renault, M.P., Rosén, A., van Stee, R.: Reordering buffer management with advice. In: WAOA (2013), to appear
2. Albers, S.: On randomized online scheduling. In: STOC. pp. 134–143. ACM (2002)
3. Alon, N., Azar, Y., Woeginger, G., Yadid, T.: Approximation schemes for scheduling. In: SODA. pp. 493–500 (1997)
4. Avidor, A., Azar, Y., Sgall, J.: Ancient and new algorithms for load balancing in the lp norm. In: SODA. pp. 426–435 (1998)
5. Balogh, J., Békési, J., Galambos, G.: New lower bounds for certain classes of bin packing algorithms. TCS 440–441(0), 1–13 (2012)
6. Böckenhauer, H.J., Komm, D., Královic, R., Královic, R.: On the advice complexity of the k-server problem. In: ICALP. pp. 207–218 (2011)
7. Böckenhauer, H.J., Komm, D., Královic, R., Královic, R., Mömke, T.: On the advice complexity of online problems. In: ISAAC. pp. 331–340 (2009)
8. Böckenhauer, H.J., Komm, D., Královic, R., Rossmanith, P.: On the advice complexity of the knapsack problem. In: LATIN. pp. 61–72 (2012)
9. Borodin, A., El-Yaniv, R.: Online computation and competitive analysis. Cambridge University Press, New York, NY, USA (1998)
10. Boyar, J., Kamali, S., Larsen, K.S., López-Ortiz, A.: Online bin packing with advice. CoRR abs/1212.4016 (2012)
11. Chandra, B.: Does randomization help in on-line bin packing? Inf. Process. Lett. 43(1), 15–19 (Aug 1992)
12. Chen, B., van Vliet, A., Woeginger, G.J.: A lower bound for randomized on-line scheduling algorithms. Information Processing Letters 51(5), 219 – 222 (1994)
13. Dorrigiv, R., He, M., Zeh, N.: On the advice complexity of buffer management. In: ISAAC. pp. 136–145 (2012)
14. Fernandez de la Vega, W., Lueker, G.S.: Bin packing can be solved within 1+epsilon in linear time. Combinatorica 1(4), 349–355 (1981)
15. Woeginger, G.J.: A polynomial-time approximation scheme for maximizing the minimum machine completion time. Oper. Res. Lett. 20(4), 149 – 154 (1997)