Information and Computation 206 (2008)1003–1016

Contents lists available at ScienceDirect

Information and Computation

journal homepage: www.elsevier.com/locate/ic

Bin packing with controllable item sizes

José R. Correa [a,1], Leah Epstein [b,*]

[a] School of Business, Universidad Adolfo Ibáñez, Santiago, Chile
[b] Department of Mathematics, University of Haifa, 31905 Haifa, Israel

ARTICLE INFO

Article history:
Received 22 August 2007
Revised 23 January 2008
Available online 4 June 2008

Keywords:
Bin packing
Online algorithms
Approximation schemes
Discrete time–cost tradeoff

ABSTRACT

We consider a natural resource allocation problem in which we are given a set of items, where each item has a list of pairs associated with it. Each pair is a configuration of an allowed size for this item, together with a nonnegative penalty, and an item can be packed using any configuration in its list. The goal is to select a configuration for each item so that the number of unit bins needed to pack the sizes plus the sum of penalties is minimized. This problem has applications in operating systems, bandwidth allocation, and discrete time–cost tradeoff planning problems.

For the offline version of the problem we design an augmented asymptotic PTAS. That is, an asymptotic approximation scheme that uses bins of size slightly larger than 1. We further consider the online bounded space variant, where only a constant number of bins can be open simultaneously. We design a sequence of algorithms, where the sequence of their competitive ratios tends to the best possible asymptotic competitive ratio. Finally, we study the *bin covering problem*, in which a bin is covered if the sum of sizes allocated to it is at least 1. In this setting, penalties are interpreted as profits and the goal is to maximize the sum of profits plus the number of covered bins. We design an algorithm of best possible competitive ratio, which is 2. We generalize our results for online algorithms and unit sized bins to the case of variable sized bins, where there may be several distinct sizes of bins available for packing or covering, and get that the competitive ratios are again the same as for the more standard online problems.

1. Introduction

The classical bin packing problem was introduced in the early 70s [27,17,18,9,6]. Since then many variants have been studied. In the basic online model, a sequence of items, which are positive numbers no larger than 1, are to be packed into unit sized bins. The sum of items packed into one bin cannot exceed its size and the supply of such bins is unbounded. Each item must be packed into exactly one bin, minimizing the number of bins used. The problem has numerous applications in storage and resource allocation.

Bin packing was investigated in both offline and online scenarios. In the latter scenario, the fact that items arrive *online* means that each item must be assigned in turn, without knowledge of the next items. We analyze the algorithms using the asymptotic approximation ratio (also called competitive ratio for the case of online algorithms). This is a standard measure for bin packing algorithms. For a given input sequence σ, let $\mathcal{A}(\sigma)$ (or $\mathcal{A}$) be the cost of algorithm $\mathcal{A}$ on σ. Let $\mathrm{OPT}(\sigma)$ (or OPT) be the cost of an optimal algorithm, i.e., the minimum possible cost to deal with σ (in the case where online algorithms are studied, this is an offline algorithm which has a complete knowledge of the sequence of items). For minimization problems,

* Corresponding author.
E-mail addresses: correa@uai.cl (J.R. Correa), lea@math.haifa.ac.il (L. Epstein).
[1] Partially supported by FONDECYT 1060035 and Anillo en Redes, ACT08.

0890-5401/$ - see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.ic.2008.04.005

the *approximation ratio* for an algorithm $\mathcal{A}$ is defined to be $\mathcal{R}(\mathcal{A}) = \limsup_{n\to\infty} \sup_\sigma \{\frac{\mathcal{A}(\sigma)}{\mathrm{OPT}(\sigma)} : \mathrm{OPT}(\sigma) = n\}$. For maximization problems, the *approximation ratio* for an algorithm $\mathcal{A}$ is defined to be $\mathcal{R}(\mathcal{A}) = \limsup_{n\to\infty} \sup_\sigma \{\frac{\mathrm{OPT}(\sigma)}{\mathcal{A}(\sigma)} : \mathrm{OPT}(\sigma) = n\}$.

The offline bin packing problem admits an APTAS (Asymptotic Polynomial Time Approximation Scheme), as shown by Fernandez de la Vega and Lueker [11]. Karmarkar and Karp [19] developed an AFPTAS (Asymptotic Fully Polynomial Time Approximation Scheme) for the same problem. Karmarkar and Karp [19] also designed an algorithm which uses at most $\mathrm{OPT}(I) + \log^2[\mathrm{OPT}(I)]$ bins for an input I. The best current results known for online bin packing are an algorithm of competitive ratio slightly less than 1.59 due to Seiden [24], and a lower bound of 1.54 on the competitive ratio of any online algorithm, given by Van Vliet [28].

An interesting class of online algorithms is the class of *bounded space* algorithms, which are defined by the property that they only have a constant number of bins available to accept items at any point during processing. These bins are also called "open bins". The bounded space assumption is quite natural. Essentially the bounded space restriction guarantees that output of packed bins is steady, and that the packer does not accumulate an enormous backlog of bins which are only output at the end of processing. Bounded space algorithms for the standard variant were studied by Lee and Lee [20] and by Woeginger [29]. Lee and Lee defined a sequence of algorithms, where the sequence of the competitive ratios of these algorithms tends to $\Pi_\infty \approx 1.691$ (the series converging to this value is defined later in the paper). They also showed that no bounded space algorithm can have a smaller competitive ratio.

In the *variable-sized* bin packing problem, there is a supply of several distinct bin sizes that can be used to pack the items. The cost of an algorithm is the sum of sizes of used bins. The first to investigate the variable sized bin packing problem were Friesen and Langston [15]. Csirik [7] proposed the Variable Harmonic algorithm and showed that it has performance ratio at most Π_∞. Seiden [23] showed that this algorithm is optimal among bounded space algorithms for any set of bin sizes. Given a finite set of bin sizes, the largest of which is exactly 1, the competitive ratio is a solution of a mathematical program. Unbounded space variable sized bin packing was studied also in [25].

Bin covering is a dual problem to bin packing. Items are to be assigned to bins as in bin packing, however there is no restriction on the sum of items in a bin. A bin is seen as covered if the total size of items assigned to it is at least 1. The goal is to maximize the number of covered bins. The problem was studied in [8,1]. It was shown in [1] that the algorithm Next Fit, which uses a single bin at every time, and moves to the next bin when it is covered, has competitive ratio 2. Csirik and Totik [8] showed that this bound is best possible for any online algorithm. In the case of variable sized bins, a bin is covered if the total size of items assigned to it is at least its size. The goal is to maximize the sum of sizes of covered bins. Woeginger and Zhang [31] showed that the best possible competitive ratio in this case can be computed as follows. Denote the list of bin sizes that are larger than $\frac{1}{2}$ by $1 = b_1 \geq \ldots b_k > \frac{1}{2}$ and let $b_{k+1} = \frac{1}{2}$. Then the best ratio is given by $\max_{i=1,\ldots,k}\{b_i/b_{i+1}\}$. Note that bin sizes which are no larger than $\frac{1}{2}$ have no influence on the competitive ratio.

Another model, which is relevant to our work, is *bin packing with rejection*. In this variant, each item is given a rejection penalty. Each item can be either packed or rejected. The cost of an algorithm is the sum of rejection penalties of rejected items plus the number of bins used to pack items. Bin covering with rejection is defined analogously. These problems were studied in [12,13,3]. In [12] a 2-competitive algorithm was designed for bin covering with rejection. This is best possible due to [8], since the standard online bin covering problem is a special case where all rejection values are zero. In [13] the algorithms of [20], and also the unbounded space algorithm of [22] were adapted to handle rejection keeping the same competitive ratios. Again, the standard problem is a special case where all rejection penalties are infinite. The offline problem admits an APTAS [13,3] and an AFPTAS [14].

In this paper, we study generalizations of bin packing and bin covering. As noted in [12,13], many applications for bin packing have an option of refusing a customer. Sometimes it is easier to pay some penalty for not providing a service, or for buying the service from an external source, rather than directly servicing the customer. Consider the following bandwidth allocation problem. A customer requests a given bandwidth. If it is impossible to provide this amount, it is possible to provide less bandwidth with a small penalty, or much less bandwidth with a greater penalty, or possibly not to provide any bandwidth at all in extreme cases, paying a large penalty. The two extremes (either provide the full amount, or reject) are not enough to model this situation. A similar situation arises when a freight carrier has to ship clients' orders between an origin and a destination. An order consists of a number of items, and there are several orders to be shipped. Some items in an order are critical and must be shipped by the company's own transportation system, while others can be outsourced (by contracting a third party to do some shipments). Therefore, to achieve its task, the company can use their own transport (trucks) or external sources. The freight carrier has to decide which part of each order is going to be externalized and which is going to be shipped by their own means. Once this is decided, they have to pay for the outsourced items and minimize the number of trucks used for the items to be shipped by themselves.

These applications result in the *bin packing problem with controllable sizes* problem, in which each item is associated with a list of possible size/cost configurations. The goal is thus to pack items choosing one configuration for each item, so that the number of bins needed plus the total cost is minimized. The *bin covering problem with controllable sizes* problem is defined similarly, only the second component of a configuration is viewed as a profit rather than a cost, and goal is to maximize the number of covered bins plus the sum of profits. These problems, in which an item has a list of configurations to choose from, generalize both the standard bin packing and bin covering problems and furthermore they generalize bin packing and bin covering with rejection. In the standard problems there is a single possible configuration for every item, whereas the problems with rejection may admit two such configurations for each item.

This problem is related to the discrete time–cost tradeoff planning problems [10,26]. In this setting, jobs, subject to precedence constraints, are to be scheduled in infinitely many parallel processors. As opposed to the continuous version of the problem, in the discrete time–cost tradeoff environment, job lengths can be varied discretely by paying additional penalties. Moreover, there is a deadline T by which all jobs have to be completed and the question is to find the cheapest schedule satisfying the deadline constraint. Interestingly, bin packing with controllable sizes is equivalent to the variation of this problem in which there are no precedences, but, on the other hand we need to minimize the total cost plus the number of machines needed to process all jobs by time T.

1.1. The setting

We denote an instance by I, which consists of a set of items $N = \{i_1, \ldots, i_n\}$, a set of pairs $L = \{(s_j, p_j) : 1 \leq j \leq \ell\}$, together with a list $L(i) \subseteq L$ for each $i \in N$. The values s_j are rational numbers between 0 and 1 while the p_j's are nonnegative rationals. Alternatively, we may view the instance as a bipartite graph $G = (N \cup L, E)$, in which $L(i)$ is the set of neighbors of $i \in N$ in the graph G. The list $L(i)$ thus represents the finite list of pairs, also called configurations to which each item i is associated. In the case of bin packing, the first values in each pair are possible sizes which this item can have when it is packed. The second value in each pair is the rejection penalty to be paid if the item is packed using the first value as its size. Therefore, an algorithm, A, needs to choose a configuration from the list for each item $i \in N$. Denote by $(s_{a(i)}, p_{a(i)})$ the choice of A. Once the choice is made, the algorithm needs to pack the item into a bin, i.e., allocate it a space of size $s_{a(i)}$ in a bin in such a way that the sum of spaces allocated in a bin cannot exceed 1. The cost of the algorithm is the number of bins used for any item plus the sum of penalties $p_{a(i)}$.

In the case of bin covering, the second component of each pair is seen as a value gained by the algorithm. Moreover, there is no restriction on the total size allocated in each bin. The goal is to maximize the sum of gained values and the number of bins that the spaces allocated in them have a total size of at least 1. In variable sized bin packing and bin covering, the chosen sizes may be packed into bins of several given sizes. The goal function changes as follows. In bin packing, the number of used bins is replaced by the total size of used bins. The penalties are paid as before. In bin covering, a bin is covered if the sum of sizes assigned to it is at least its size. The goal function contains the sum of sizes of covered bins instead of the number of such bins.

1.2. Our results

We first consider the offline problem. Given a value $\varepsilon > 0$, we design an algorithm of cost at most $(1+\varepsilon)\mathrm{OPT} + 5$, using bins of size $1+\varepsilon$. Such a scheme is called *An augmented APTAS*. To obtain the result we consider a rounding of the instance to obtain a simple enough structure, using techniques of item classification. Although these techniques are fairly standard, the rounding and classification task is more involved in our situation as we also need to clean up the lists of pairs corresponding to items in a way that an optimal solution to the simplified instance is also near optimal for the original one. Then, the first main idea behind the algorithm is to use a series of maximum flow computations to find a good solution for all items except those having in their list a pair which is very small in both components. To deal with the latter items we use a greedy approach. The justification for this approach comes from a result about the Minkowski sum of two-dimensional convex polytopes, which may be of independent interest. The usage of a Minkowski sum allows us to prove that a greedy algorithm that we apply in one of the cases is a good approximation. The scheme uses enumeration techniques in order to consider a wide enough class of options. Observe that in the discrete time–cost tradeoff setting, this result says that we can find a schedule of cost no more than $(1+\varepsilon)\mathrm{OPT} + 5$ with deadline $(1+\varepsilon)T$, where OPT is the cost of the optimal schedule with deadline T.

In addition, we consider online bounded space bin packing and generalize the results of Lee and Lee [20] to our problem. We design a sequence of algorithms, each of which is an adaptation of the HARMONIC$_k$ algorithms of [20]. Although our algorithms must take all packing options into account, we prove that it has the same competitive ratio. Therefore, we can achieve a competitive ratio which is arbitrarily close to the best possible one, namely $\Pi_\infty \approx 1.691$. We next consider variable sized packing and generalize the results of Seiden [23]. Even though the adaptations are non-trivial, we prove that the competitive ratios are not affected. Finally, we consider online bin covering. We adapt the algorithms of [1,31] and design algorithms of best possible competitive ratio for the problem with unit sized bins and for the variable sized problem. Note that again the competitive ratios of the adapted algorithms remain the same.

2. Offline packing

In this section we consider the offline version of the problem and design an augmented PTAS for it. Before we describe the scheme, we discuss the main steps of the scheme and provide intuition regarding its correctness.

Rounding and discarding. In the scheme we apply classification and rounding techniques. Note that in many problems items have fixed sizes, and thus rounding procedures are applied on the items directly. However, in our case, items do not have a fixed size so the task is more involved. The procedures have to be applied on the configurations, that is, on all the possible sizes and penalties of items.

We would like to partition the configurations (also called *pairs*) into "small" and "large" ones. This is a common approach allowing to round large enough sizes into a constant number of options. Such a rounding is only successful if the sizes are bounded from below, so that the cost of an optimal solution to the resulting instance is within a small factor of the original optimal cost. Since in our case every pair has two numbers associated it, this results in four sets *BE* (big and expensive), *SE* (small and expensive), *BC* (big and cheap), and finally *SC* (small and cheap). However, we need that the gap between the largest small size (price) and the smallest large size (price) is big enough for two main reasons: First, we would like to claim that an item that can be packed using some configuration in *SC*, is always packed using some configuration in *SC* in an optimal solution. This is generally incorrect, but we show that it is possible to adapt an optimal solution into a near-optimal solution that complies with this requirement. This holds as long as the previously mentioned gap is large. Secondly, we want to round the large components of pairs in *BE*,*SE* and *BC* to a constant number of values, and their corresponding small component (if exists) is simply rounded to the maximum possible size of a small component. However, if we just use a threshold t, where sizes no larger than t are defined to be small, and sizes larger than t are defined to be large, we may significantly increase the cost of an optimal solution by rounding all small sizes to t. Again a large gap is required, so that the largest small size is negligible when compared to the smallest large size (and thus the contribution of the small component of a pair to the objective function value will be much smaller than the contribution of its large component).

To create this large gap between the largest small size (price) and the smallest large size (price) we look for a set of "medium" configurations, omit all configurations in it from the list of configurations of every item, and discard all items having their "best" configuration (i.e., the configuration which minimizes the sum of size and penalty) in this set. Such a set must be one that is used relatively rarely in an optimal (or near-optimal) solution. In order to ensure the existence of such a set, the list of possible configurations must be partitioned into a large enough number of sets. Since the assignment of the optimal solution is unknown to us, we use enumeration to check all possible sets of medium size. Once a gap is created between small and large items, it is possible to apply the rounding as above. We prove that for the proper choice of medium configurations, the rounding does not harm the quality of the solution too much. Moreover, we prove that it is possible to overcome the removal of the medium configurations by using different configurations for items packed using the removed configuration in the optimal solution unless this is their "best" configuration.

The search for the proper "medium" configurations set is done in the main cycle of the algorithm. In step (1) we eliminate medium configurations and discard items having their best configuration in this medium set. Then, in step (3) we perform the second clean up step, discarding all items having a configuration in *SC*. Finally, in step (4) we do the required rounding.

Besides proving that an optimal solution to the rounded instance is near optimal, a key observation is that there is a near-optimal solution that packs the rounded instance and the discarded items separately. Thus, our algorithm will first find a solution to the rounded instance and then complete it with the discarded items.

Dealing with the rounded instance. In step (5), the rounded configurations in *BE*, *BC* and *SE* are assigned to items that do not have a configuration in *SC*. We use enumeration to determine the number of items that are packed according to a given configuration. For each such option, we solve a maximum flow problem, in order to check whether it is possible to assign items to configurations that are valid for them, so that the total number of items packed according to each configuration is as expected. The usage of flow is unusual in the design of approximation schemes. This is due to the fact that problems where network flow is helpful are usually just solved completely via flow methods and are thus polynomially solvable. We employ flow to solve an assignment problem due to the unique structure of our problem, since the sizes that we enumerate are just configurations and not associated with particular items.

After we have guessed the number of items to be packed using each possible configuration, we find a packing of large items into a minimum number of bins of size $1+\varepsilon$ (guaranteeing that the number of bins is as small as the optimal number of bins without rounding). This is done by the algorithm in [16], which is sketched below.

Dealing with the discarded items. Once the best packing of items with configurations only in $BE \cup BC \cup SE$ is found, we need to add the items that are going to be packed according to a configuration in *SC* (which we call here the "small items"), which were discarded previously. These items are added to the packing in step (6) in one of two possible ways.

Note that an item may have a number of configurations in *SC*. For every small item, we choose the "best" configuration for it in *SC*. If with this choice the sum of sizes already fills the gaps in the bins left by the packing of large items (and possibly requires additional bins), then it is clearly the best option. The small items are small enough that a Next Fit packing of these items will create bins that are almost full.

Otherwise, for every item, we start with its configuration where the size is smallest (ignoring the penalty). And apply a greedy process of moving one item at a time into a different configuration, until the total size fills the gaps. The item for which the configuration is changed, and the new configuration, are chosen so as to maximize a function which measures the tradeoff between increase in size and reduction in penalty. Surprisingly, this greedy process can be applied to find the best such solution, which we show using an interesting result on Minkowski Sums (that we prove in this paper as well).

The algorithm simply checks the packing resulting from these two greedy approaches and chooses one of the resulting solutions, which has a minimum cost. As the "medium" items discarded in step (1) are negligible, we pack them using new bins according to their best configuration. Finally we choose the packing with minimum cost among all packings considered by our enumeration on the possible sets of medium configurations.

The Hochbaum and Shmoys algorithm. Our algorithm uses the scheduling algorithm of Hochbaum and Shmoys [16] as a procedure. We give a sketch of that algorithm, where we see it as a bin packing algorithm rather than a scheduling algorithm and thus use the terminology of bin packing. The input is a set of $\tilde{n}$ items with given sizes $\{s_1 \geq \ldots \geq s_{\tilde{n}}\}$ that can be packed

into $\tilde{m}$ bins of size 1, and a constant $\varepsilon > 0$. The output of the algorithm is a packing of these items into $\tilde{m}$ bins of size $1+\varepsilon$. The algorithm finds the maximum index ℓ such that $s_\ell > \delta = \frac{\varepsilon}{2}$. If no such index exists, a greedy NEXT FIT packing into bins of size $1+\varepsilon$ already gives the desired result, since all bins (that are used) are occupied with a total size of more than 1 (except for possibly the last such bin), and thus at most $\tilde{m}$ bins are used. Otherwise, the sizes of the first ℓ items are rounded up into sizes s'_i, where $s'_i \geq s_i$ is the minimum number of the form $\delta + \delta^2 j$ for some $j \geq 1$. These ℓ items are called *large* whereas the remaining items are called *small*. This increases the size of items by a factor of at most $1+\delta$. Next, a packing of the rounded items into bins of size $1+\delta$ is found. When rounded items are later replaced with their original sizes, the total size of items in a bin may only decrease. At this time, a constant number of large item sizes are present. Since a bin of size $1+\delta$ may contain at most $\frac{1}{\delta}$ large items, it is possible to enumerate all packing patterns, where the number of configurations is constant as a function of ε. After this step, an assignment problem via integer programming in a fixed dimension (see [21]) is solved. Small items are added greedily, increasing the bin sizes by at most an additional additive factor of δ.

2.1. The algorithm

In the algorithm below we will assume the instance is given as a triple $I = (N,L,G)$ where $N = \{i_1,\ldots,i_n\}$ is the set of items, $L = \{(s_j,p_j) : 1 \leq j \leq \ell\}$ set of pairs, and $G = (N \cup L, E)$ the bipartite graph defining the lists of possible pairs $L(i)$. Furthermore, for each $i \in L$ we let $o(i) \in L(i)$ be the pair selected by the optimal solution, and $m(i) = \arg\min\{s_j + p_j : j \in L(i)\}$. Also, let $\mathrm{OPT}(I)$ denote the value of an optimal solution to instance I.

Input: A instance $I = (N,L,G)$.

For $k = 1,\ldots,4/\varepsilon$, let $M_k = \{j \in L : \varepsilon^{k+1} \leq s_j \leq \varepsilon^k$ or $\varepsilon^{k+1} \leq p_j \leq \varepsilon^k\}$ and **do**

(1) Define a new instance I' as follows: $N' = N \setminus \{i \in N : m(i) \in M_k\}$ and for each $i \in N'$, let $L'(i) = L(i) \setminus \{j : (s_j,p_j) \in M_k\}$. For an item $i \in I'$, we define $m'(i)$ and $o'(i)$ analogously to $m(i)$ and $o(i)$.

(2) Let $\delta = \varepsilon^k$, and partition $L' = \cup_{i \in N'} L'(i)$ into four sets:

- $BE = \{j \in L' : s_j \geq \delta, p_j \geq \delta\}$ (big and expensive).
- $SE = \{j \in L' : s_j \leq \varepsilon\delta, p_j \geq \delta\}$ (small and expensive).
- $BC = \{j \in L' : s_j \geq \delta, p_j \leq \varepsilon\delta\}$ (big and cheap).
- $SC = \{j \in L' : s_j \leq \varepsilon\delta, p_j \leq \varepsilon\delta\}$ (small and cheap).

(3) We now clean up items, such that their corresponding list contains a pair in SC. To this end define an instance I'' as follows: $N'' = N' \setminus \{i \in N' : L'(i) \cap SC \neq \emptyset\}$ and we let $L''(i) = L'(i)$ for all $i \in N''$. We also define BE,SE,BC,SC accordingly (note that we get the same sets as before except that $SC = \emptyset$). *We could have skipped step (2) above, however it will be useful in order to break down the analysis.*

(4) We now proceed to round pairs in BE,SE and BC. Both coordinates of pairs in BE are rounded up to the next number of the form $\delta + r\varepsilon\delta$ (for integer r). The first coordinate (size) of pairs in SE is rounded up to $\varepsilon\delta$ while the second (penalty) is rounded up to the next number of the form $\delta + r\varepsilon\delta$ (for integer r). Finally, the first coordinate (size) of pairs in BC is rounded up to the next number of the form $\delta + r\varepsilon\delta$ (for integer r) while the second (penalty) is rounded up to $\varepsilon\delta$.

(5) With the previous rounding, all sizes and penalties are at least $\varepsilon\delta$, and there are at most $K \leq (1/\varepsilon\delta)^2$ different pairs. Therefore we can guess the optimal assignment of items to pairs for this instance by solving a polynomial number of flow problems as follows:

For each integral vector $(n_1,\ldots,n_K) \geq 0$ such that $\sum_{j=1}^{K} n_j = |N''|$, we will attempt to pack n_j pairs of type (s_j,p_j).

(5.1) Check whether $(n_1,\ldots,n_K)$ is feasible for instance I''. This is done using max-flow in the bipartite graph G'': add a source and a sink, and capacities n_j from vertex (s_j,p_j) to the sink, capacities 1 from the source to every vertex in N'' and capacities 1 to all edges in E (that are directed from the vertices of N'' towards the vertices of L''). Now the maximum flow equals $|N''|$ if and only if there is an assignment of items in N'' to pairs in L'' such that n_j items are matched to a pair of type (s_j,p_j) (i.e., $(n_1,\ldots,n_K)$ is feasible).

(5.2) Pack optimally a bin packing instance with n_j items of size s_j using bins of size $1+\varepsilon$, using the algorithm in [16]. Call $BINS(n_1,\ldots,n_K)$ the number of bins used by the solution. Note that the algorithm of [16] assumes that the number of bins to be used is known. Therefore, in order to apply it for here, we need to find the smallest m such that it is possible to pack the input into m bins of size $1+\varepsilon$. For this, we can run the algorithm of [16] with $\tilde{m} = m$ for all $0 \leq m \leq n$.

(5.3) Choose the feasible vector $(n_1,\ldots,n_K)$ such that $BINS(n_1,\ldots,n_K) + \sum_{j=1}^{K} n_j p_j$ is minimized.

(6) To finish we need to see how to deal with items in $N' \setminus N''$ and with those in $N \setminus N'$.

(6.1) Dealing with $S = N' \setminus N''$. Let C be the total space left in the already opened bins (which are of size $1+\varepsilon$). For each item $i \in S$ we only consider the restriction of its list $L(i)$ to pairs in SC, and denote this new list as $L_{SC}(i)$. We proceed as follows:

(*i*) For all $i \in S$, select from each list $L_{SC}(i)$ the pair with smallest size (first component) and call the index of this pair $a(i)$.
(*ii*) Find $z(i) = \arg\max\{(p_{a(i)} - p_j)/(s_j - s_{a(i)}) : \text{ for } j \in L_{SC}(i)\}$ and let $z(i^*) = \max_{i \in S} z(i)$.
(iii) Update $a(i^*) = z(i^*)$ and go to (ii) until the total size of chosen pairs surpasses $C + \varepsilon$.
(*iv*) Pack items by Next Fit (using the $a(i)$ assignment) in the already opened bins and possibly increasing their size to $1 + 2\varepsilon$.
(*v*) For all $i \in S$, select from each list $L_{SC}(i)$ the pair $m'(i)$ with smallest size plus penalty (i.e., $m'(i) = \arg\min_{j \in L_{SC}(i)}\{s_j + p_j\}$) and pack items by Next Fit (using this assignment) in the already opened bins increasing their size to $1 + 2\varepsilon$, and possibly using new bins.
(*vi*) Choose the solution with smallest total cost from those constructed in (iv) or (v).

(6.2) Dealing with $N \setminus N'$. Each $i \in N \setminus N'$ is assigned to pair $m(i) \in L(i)$. Then they are packed using only new bins by any constant factor approximation for bin packing, e.g. by Next Fit.

end for
Output: Output the lowest cost packing thus constructed.

2.2. Analysis of correctness

The idea of the algorithm is to transform the instance several times so that an near optimal solution to the transformed instance is also near optimal for the original one. Let us start by observing that the main cycle will eventually reach a good value of k.

We define sets I^k as follows. $I^k = \{i : \ m(i) \in M_k \text{ or } o(i) \in M_k\}$.

Lemma 1. *There exists* $1 \le k^* \le 4/\varepsilon$ *such that* $\sum_{i \in I^{k^*}} (s_{m(i)} + p_{m(i)} + s_{o(i)} + p_{o(i)}) \le \varepsilon \mathrm{OPT}(I)$.

Proof. The property $s_{m(i)} + p_{m(i)} \le s_{o(i)} + p_{o(i)}$ holds for every item i. Using this property we clearly have $\sum_{i \in N}(s_{m(i)} + p_{m(i)} + s_{o(i)} + p_{o(i)}) \le 2\mathrm{OPT}(I)$. Furthermore, as each pair appears in at most two sets M_k, we have that:

$$\sum_{k=1}^{4/\varepsilon} \sum_{i \in I^k} (s_{m(i)} + p_{m(i)} + s_{o(i)} + p_{o(i)}) \le 2 \sum_{i \in N} (s_{m(i)} + p_{m(i)} + s_{o(i)} + p_{o(i)}) \le 4\mathrm{OPT}(I).$$

The lemma follows since there has to be a term on the left, such that its value is at most the average. □

Note that, as we do not know $o(i)$, we cannot compute k^* a priori, however, the main cycle of the algorithm will eventually guess such a value. From now on, we assume we have guessed a value of k^* as in Lemma 1.

Lemma 2. *All items discarded in step (1) are packed in step (6.2) at a total cost of at most* $3\varepsilon\mathrm{OPT}(I) + 1$.

Proof. By Lemma 1 we have that $\sum_{i : m(i) \in M_{k^*}} (s_{m(i)} + p_{m(i)}) \le \sum_{i \in I^k} (s_{m(i)} + p_{m(i)} + s_{o(i)} + p_{o(i)}) \le \varepsilon\mathrm{OPT}(I)$. Therefore, if we assign every item in $N \setminus N'$ to $m(i)$, Next Fit will need at most $2\varepsilon\mathrm{OPT}(I) + 1$ bins, while the total penalty will be at most $\varepsilon\mathrm{OPT}(I)$. □

Lemma 3. *The instance* I' *defined in step (1) satisfies* $\mathrm{OPT}(I') \le (1 + O(\varepsilon))\mathrm{OPT}(I) + 1$.

Proof. Take an optimal solution (which consists of a choice of a configuration for each item, that implies penalties, together with a packing) to instance I and map it to a feasible solution to instance I' (except for the items in $N \setminus N'$). This mapping is valid all $i \in N'$ except for those such that $(s_{o(i)}, p_{o(i)}) \in M_{k^*}$ (but $(s_{m(i)}, p_{m(i)}) \notin M_{k^*}$). However, by Lemma 1, for these items:

$$\sum_{i : o(i) \in M_{k^*}} (s_{m(i)} + p_{m(i)}) \le \sum_{i \in I^k} (s_{m(i)} + p_{m(i)} + s_{o(i)} + p_{o(i)}) \le \varepsilon\mathrm{OPT}(I).$$

Therefore, they can be packed using only new bins at cost $O(\varepsilon)\mathrm{OPT}(I) + 1$ by simply assigning them to $m(i)$ and using Next Fit. □

The last Lemma implies that it is enough to get an augmented APTAS for instances such as I', in which no pair has a coordinate between ε^{k^*+1} and ε^{k^*}. This motivates the partitioning scheme of step (2). We now turn to see that after the second cleaning step of the algorithm a close to optimal solution is maintained. This justifies the fact that we can deal with the parts of N' separately as we deal with N'' in steps (4) and (5) and with $N' \setminus N''$ in step (6).

Lemma 4. *There exists a solution with a cost of at most* $(1+O(\varepsilon))\mathrm{OPT}(I')+1$, *that for every* $i \in N'$, *such that* $L'(i) \cap SC \neq \emptyset$, *the solution assigns* i *to a pair in* $L'(i) \cap SC$.

Proof. Suppose that in an optimal solution for instance I', each item $i \in F$, where $F \subseteq I'$, is assigned to $o(i) \in L'(i) \setminus SC$ and $L'(i) \cap SC \neq \emptyset$. Consider a new feasible solution that assigns each item $i \in F$ to $m(i)$. Clearly for all $i \in F$, $m(i) \in L'(i) \cap SC$ (since for each pair in SC, its size plus its penalty is at most $2\varepsilon\delta$, while the size plus the penalty of every pair in $L'(i) \setminus SC$ is at least δ).

Observe that for $i \in F$, at least one of $s_{o(i)} \geq \delta$ and $p_{o(i)} \geq \delta$ holds, so we can partition F into $F_s = \{i \in F : s_{o(i)} \geq \delta\}$ and $F_p = \{i \in F : p_{o(i)} \geq \delta, s_{o(i)} < \delta\}$. Thus, an optimal solution to I' uses at least $\delta|F_s|$ bins, and pays a total penalty of at least $\delta|F_p|$. This implies that $|F| \leq \mathrm{OPT}(I')/\delta$. It follows that if we reassign items $i \in F$ to $m(i)$ instead of $o(i)$ (and pack using Next Fit) the cost will increase by at most $2|F_s|\varepsilon\delta + 1 + |F_p|\varepsilon\delta \leq 3|F|\varepsilon\delta + 1 \leq 3\varepsilon\mathrm{OPT}(I') + 1$. □

The previous lemma implies that to get a near optimal packing of I', it is enough to first get a near optimal packing of I'' and then complete it with the rest of the items using pairs which are small and cheap. Indeed, consider an optimal solution to instance I' in which lists $L'(i)$ are changed to $L'(i) \cap SC$, whenever the latter is not empty. Suppose we can also guess the optimal pair $o'(i)$ to be assigned to any item i. We can now remove the items of SC from the optimal packing and repack them using Next Fit in the available space. As we are now using bins of size $1+\varepsilon$ and since all items are of size at most $\delta\varepsilon$, the total size of items that each bin (except for the last one) would contain increases to at least $1+\varepsilon-\varepsilon\delta$. Therefore, the items will fit, and thus the number of bins used will stay the same (or decrease). We now proceed to prove that the rounding of pairs in BE,SE,BC and SC done in step (4), does not affect the optimal value too much.

Lemma 5. *The optimal solution to instance* I'' *with pairs in* BE,SE,BC *rounded as in step* (4), *when using bins of size* $1+\varepsilon$, *has cost at most* $(1+O(\varepsilon))\mathrm{OPT}(I'')+1$.

Proof. Consider an optimal solution for instance I'' before the rounding. Let B, C, D be the items in N'' assigned to pairs in BE,SE and BC in such an optimal solution respectively. Clearly the rounding of BE (i.e., items in B) remains feasible when the size of the bins is augmented to $1+\varepsilon$, since there are at most $1/\delta$ items in any one bin. Also, the total penalty paid by the rounded instance for BE only increases by a factor $1+\varepsilon$. On the other hand, note that for $i \in C$, $p_{o(i)} \geq \delta$, and for $i \in D$, $s_{o(i)} \geq \delta$. This implies that $|C| \leq \mathrm{OPT}(I'')/\delta$ and $|D| \leq \mathrm{OPT}(I'')/\delta$. Therefore, the rounding of BC (i.e., items in D) can only increase the penalty by a factor $1+\varepsilon$. In addition, while the rounding of SE (i.e., items in C) can actually overflow bins by too much, the items overflowing bins can be removed from their current bins and placed in new bins using Next Fit. These new bins will be almost full and therefore there can be at most $\varepsilon\mathrm{OPT}(I'')+1$ new bins. Overall we have increased the cost by at most $2\varepsilon\mathrm{OPT}(I'')+1$. □

With this rounding we have a constant number of elements in $L'' = \cup_{i\in N''} L''(i)$. And using the flow technique described in step (5) we guess the optimal assignment of items $i \in N''$ to pairs in their corresponding $L''(i)$. This is done by solving at most n^K maximum flow problems, where $K \leq (1/\varepsilon\delta)^2$ is the number of possible pairs after the rounding. Once the assignment has been guessed we use [16] to solve the bin packing problem optimally using bins of size $1+\varepsilon$ (we have a constant number of items all of size at least $\varepsilon\delta$). Therefore we obtain an optimal solution of cost smaller than that for instance I'' after the rounding (since we use larger bins).

To finish the proof of correctness we need to see that step (6.1) correctly packs items in $N' \setminus N''$. To this end we first prove a lemma concerning the Mikowski sum of convex two-dimensional polyhedra. For $i = 1,\ldots,n$, let P_i be a convex polyhedron in $\mathbb{R}^2_+$ and let D_i be the dominant of P_i (i.e., $D_i = P_i + \mathbb{R}^2_+$). Denote the set of extreme points of D_i by $L_i = \{x_i^1,\ldots,x_i^{\mu(i)}\} \subset \mathbb{R}^2_+$, where $x_i^j = (s_i^j, p_i^j)$, and assume that x_i's are sorted in increasing order of their first component $s_i^1 < \ldots < s_i^{\mu(i)}$. Let D be the Minkowski sum of sets D_i i.e., $D = \sum_{i=1}^{n} D_i = \{y \in \mathbb{R}^2_+ : y = y_1 + \ldots + y_n, \text{ with } y_i \in D_i\}$ (see Figure 1).

Lemma 6. *The following hold*:

(i) D *is a polyhedral convex set and if* x *is a vertex of* D, x *can be written as sum of vertices in* D_i, *i.e.*, $x = x_1 + \ldots + x_n$, *where each* x_i *is a vertex in* D_i.

(ii) *If* $x = x_1^{l_1} + \ldots + x_n^{l_n}$ *and* $y = x_1^{h_1} + \ldots + x_n^{h_n}$ *are two adjacent vertices of* D *(joined by a one-dimensional face of* D*), then for all* u *such that* $x_u^{l_u} \neq x_u^{h_u}$

$$\frac{p_u^{l_u} - p_u^{h_u}}{s_u^{h_u} - s_u^{l_u}} = \max_{1\leq k\leq n}\left\{\frac{p_k^{l_k} - p_k^{h_k}}{s_k^{h_k} - s_k^{l_k}}\right\},$$

where by convention $0/0 = 0$. *In other words, this ratio is either zero or constant. Furthermore if*

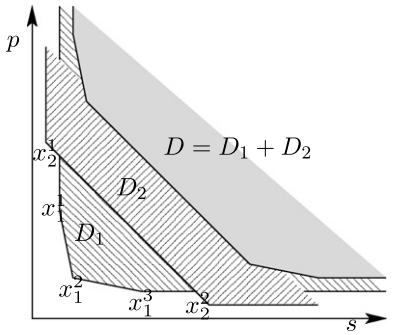

Fig. 1. Example of Minkowski sum.

$$K = \arg\max \left\{ \frac{p_k^{l_k} - p_k^{l_k+1}}{s_k^{l_k+1} - s_k^{l_k}} : k = 1,\ldots,n;\ l_k < \mu(k) \right\} \neq \emptyset,$$

then $x' = x + \sum_{k\in K} (x_k^{l_k+1} - x_k^{l_k})$ *is also a vertex of D which is adjacent to x.*

Proof.

(i) The sum of convex sets is convex. It is also easy to see that it is polyhedral. Let x be a vertex of D, and suppose that $x = x_1 + \ldots + x_n$, with $x_i \in D_i$ for all $i = 1,\ldots,n$, but x_j is not a vertex of D_j for some $1 \le j \le n$. Thus, $x_j = (x'_j + x''_j)/2$ where $x'_j, x''_j \in D_j$. It follows immediately that $x = (x' + x'')/2$, with $x' = x - x_j + x'_j$ and $x'' = x - x_j + x''_j$, where clearly $x', x'' \in D$. A contradiction is obtained.

(ii) Suppose x and y are as in (ii) and let k be an integer so that $x_1^{l_1} \neq x_1^{h_1}, \ldots, x_k^{l_k} \neq x_k^{h_k}, x_{k+1}^{l_{k+1}} = x_{k+1}^{h_{k+1}}, \ldots, x_n^{l_n} = x_n^{h_n}$, that is, x and y differ in the first k terms and they are equal in the last $n-k$ (this is done without loss of generality maybe by reordering the terms in the sum). Let $z_1,\ldots,z_k \in D$ be defined as $z_j = x - x_j^{l_j} + x_j^{h_j}$. Clearly $z := \frac{1}{k}\sum_{j=1}^{k} z_j = \frac{1}{k}y + \left(1 - \frac{1}{k}\right)x$. Therefore z is a convex combination of x and y, thus it lies in the segment $[x,y]$. However, z is also a convex combination of the z_j's, which in turn implies that all z_j's have to lie in the segment $[x,y]$, since all z_j's belong to D. Then, for all $j = 1,\ldots,k$ the slope $(p_j^{l_j} - p_j^{h_j})/(s_j^{h_j} - s_j^{l_j})$ is constant. The first part of assertion is proved.

To see the second, let y be the vertex in D adjacent to x with larger first component. Keeping the notation, let us say $y = x_1^{h_1} + \ldots + x_n^{h_n}$. Thus z_1, defined as above, lies in the segment $[x,y]$ (here we are again assuming $x_1^{h_1} \neq x_1^{l_1}$, and note that $h_1 > l_1$). Thus the slope of segment $[x,y]$ equals, $(p_1^{h_1} - p_1^{l_1})/(s_1^{h_1} - s_1^{l_1})$. Now we have that

$$\frac{p_1^{l_1} - p_1^{l_1+1}}{s_1^{l_1+1} - s_1^{l_1}} \ge \frac{p_1^{l_1} - p_1^{h_1}}{s_1^{h_1} - s_1^{l_1}} \ge \max \left\{ \frac{p_k^{l_k} - p_k^{l_k+1}}{s_k^{l_k+1} - s_k^{l_k}} : k = 1,\ldots,n;\ l_k < \mu(k) \right\}.$$

The first inequality follows from the convexity of D_1 and the choice of the ordering for the vertices of D_1, the second inequality follows form the fact that segment $[x,y]$ is the one having the most negative slope out of all segments $[x,u]$ with u having larger first component than x (in particular this holds for $u = x'$). The latter implies that segment $[x,x']$ is contained in segment $[x,y]$, and from the maximality of K, $y = x'$. □

Lemma 6 essentially says that we can enumerate all extreme points of D efficiently. To do this part (ii) of the lemma suggests that one should start by the vertex of D with highest penalty $x = x_1^1 + \ldots + x_n^1$. Then, if at some point we have vertex $x = x_1^{l_1} + \ldots + x_n^{l_n}$, find the index i such that the slope between $x_i^{l_i}$ and $x_i^{l_i+1}$ is most negative, and move to $x' = x - x_i^{l_i} + x_i^{l_i+1}$. This is repeated until one reaches the last point $x_1^{\mu(1)} + \ldots + x_n^{\mu(n)}$. In particular, the latter implies that the number of extreme point in D is at most $\sum_{i=1}^{n} \mu(i)$.

We now go back to prove that step (6.1) correctly packs items in $N' \setminus N''$. Recall that for all such items it is enough to consider pairs which are both small and cheap (as long as we only would like to get an augmented APTAS).

Lemma 7. *The solution for instance I' obtained at the end of step (6.1) has cost at most* $(1+\varepsilon)\mathrm{OPT}(I') + 3$.

Proof. Consider a near optimal packing of I'' as that constructed in step (5). Given this packing, let us assume that the optimal solution that uses bins of size $1+\varepsilon$, assigns to items $i \in S$ a pair $b(i) \in L_{SC}(i)$. For simplicity of notation assume also that for

$i \in N''$, $b(i)$ is the pair assigned to i in step (5). Therefore $b(i)$ is defined for every $i \in N'$. Clearly, Lemmas 4 and 5 imply that this latter solution has cost, *COST*, at most $(1 + O(\varepsilon))\mathrm{OPT}(I') + 2$. We shall show that the solution constructed in step (6.1) is close to that just described. We consider three cases.

(a) In step (6.1.v) $\sum_{i\in S} s_{m'(i)} > C$. In this situation we are done since the total cost of the packing constructed in step (6.1.v) is at most

$$A(I'') + \sum_{i\in S} s_{m'(i)} - C + 1 + \sum_{i\in S} p_{m'(i)} \le A(I'') - C + 1 + \sum_{i\in S}(s_{b(i)} + p_{b(i)}) \le COST + 1,$$

where $A(I'')$ is the total cost of the solution for instance I'' constructed by the algorithm in step (5). The cost of the sizes and penalties used by all items of N'' in the considered solution is $A(I'') - C$. Since we increase each bin before packing, each bin except for the last one will contain a total size of $1 + 2\varepsilon$ and a total of at least C can be packed into existing bins. Therefore, every bin that receives items of S except for the last one contains a total of at least $1 + 2\varepsilon - \varepsilon\delta > 1 + \varepsilon$. This means that the already existing bins receive a total of at least C, and the number of additional bins needed is at most $\lceil \sum_{i\in S} s_{m'(i)} - C \rceil \le \sum_{i\in S} s_{m'(i)} - C + 1$.

(b) In step (6.1.v) $\sum_{i\in S} s_{m'(i)} \le C$. If this is the case, suppose that $\sum_{i\in S} s_{b(i)} = C + U$, for some positive U, and that $\sum_{i\in S} p_{b(i)} = V$. Therefore,

$$COST \ge BINS + \frac{U}{1+\varepsilon} + \sum_{i\in N''} p_{b(i)} + V \ge \frac{\sum_{i\in N'} s_{b(i)}}{1+\varepsilon} - 1 + \sum_{i\in N'} p_{b(i)},$$

where *BINS* is the number of bins used in step (5) for instance I''. Note that the last inequality holds since in the $b(i)$ assignment all bins (except the las one) are almost full. Consider $T \subset S$ satisfying $U \le \sum_{i\in T}(s_{b(i)} - s_{m'(i)}) \le U + \varepsilon$. Such a T necessarily exists and it is easy to find since for $T = S$ the latter difference is at least U, for $T = \emptyset$ the difference is 0, and moreover, $s_i \le \varepsilon$ for $i \in SC$. With this, and from the choice of $m'(i)$, we also have that $\sum_{i\in T}(p_{b(i)} - p_{m'(i)}) \ge -U - \varepsilon$. Therefore, if we change the assignment for items in $i \in T$ from $b(i)$ to $m'(i)$, keeping the packing of I'' and repacking items in S by Next Fit, the new cost will be at most

$$\sum_{i\in N'\setminus T} s_{b(i)} + \sum_{i\in T} s_{m'(i)} + \sum_{i\in N''} p_{b(i)} + (V + U + \varepsilon) \le \sum_{i\in N'} s_{b(i)} - U + \sum_{i\in N''} p_{b(i)} + (V + U + \varepsilon),$$

$$\le \sum_{i\in N'} s_{b(i)} + \sum_{i\in N'} p_{b(i)} + \varepsilon \le (1+\varepsilon)COST + 1.$$

The latter means that there is an assignment of items in $i \in S$ to a pair $b'(i) \in L_{SC}(i)$ resulting in a solution with cost at most $(1 + \varepsilon)COST + 1$, satisfying $\sum_{i\in S} s_{b'(i)} \le C$. Thus, we reduce to case (c).

(c) The assignment $b(i)$ satisfies $\sum_{i\in S} s_{b(i)} \le C$. In this case, we show that the solution produced at step (6.1.iv) is near optimal. Consider Lemma 6 with P_i being the convex hull of $L_{SC}(i)$, for $i \in S$. The assertion of the lemma guarantees that our algorithm will end up with an assignment $a(i)$ for items $i \in S$, such that the frontier of D contains $\left(\sum_{i\in S} s_{a(i)}, \sum_{i\in S} p_{a(i)}\right)$. Since $s_{a(i)} \le \delta\varepsilon$, from (6.1.iii) it follows that $C + \varepsilon \le \sum_{i\in S} s_{a(i)} \le C + \delta\varepsilon + \varepsilon$. This in turn implies that $\sum_{i\in S} p_{a(i)} \le \sum_{i\in S} p_{b(i)}$, as otherwise $\left(\sum_{i\in S} s_{b(i)}, \sum_{i\in S} p_{b(i)}\right)$ would be a point in D, such that its components are strictly smaller that those of $\left(\sum_{i\in S} s_{a(i)}, \sum_{i\in S} p_{a(i)}\right)$. Overall the cost of the solution returned by the algorithm is $A(I'') + \sum_{i\in S} p_{a(i)} \le COST$.

In all three cases we obtain a feasible solution, together with a packing in bins of size $(1 + 2\varepsilon)$, of total cost of at most $(1 + \varepsilon)COST + 1 \le (1 + O(\varepsilon))\mathrm{OPT}(I') + 3$. □

Theorem 8. *The solution returned by the algorithm has cost at most* $(1 + O(\varepsilon))\mathrm{OPT}(I) + 5$

Proof. Last lemma together with Lemmas 2 and 3 imply that the cost of the solution returned by the algorithm is at most $(1 + O(\varepsilon))\mathrm{OPT}(I') + 3 + 3\varepsilon\mathrm{OPT}(I') + 1 \le (1 + O(\varepsilon))\mathrm{OPT}(I') + 4 \le (1 + O(\varepsilon))\mathrm{OPT}(I) + 5$. □

3. Online bounded space packing

We first study the problem where the items are allocated spaces in unit sized bins. In Section 4, we generalize it to the case of variable sized bins. Also, in Section 5 we study online bin covering, and its generalization to variable sized bins.

3.1. Algorithm Picky Harmonic

We define our algorithm which uses as a procedure the Harmonic$_k$ algorithm of Lee and Lee [20]. Our algorithm is called Picky Harmonic$_k$ (PH$_k$). The main idea of "harmonic-based" algorithms is to first classify items by size, and then pack an item according to its class (as opposed to letting the exact size influence packing decisions, as in fit based algorithms such as First Fit and Next Fit). Since our items do not have a fixed size, given an item, we first decide which configuration to use and then apply a "harmonic-based" classification. Our algorithm uses an integer parameter $k \geq 2$.

We first show how an item is packed after one of its possible configurations is chosen. We call the size of the item in this configuration "its chosen size". First define two auxiliary functions: For $s \in [0,1]$, let $K(s) = k$ if $s \leq \frac{1}{k}$ and $K(s) = \lfloor \frac{1}{s} \rfloor$ otherwise; also let $w(s) = \frac{1}{K(s)}$ for $s \in \left(\frac{1}{k},1\right]$ and $w(s) = \frac{ks}{k-1}$ for $s \in \left[0,\frac{1}{k}\right]$.

For the classification of chosen sizes, we partition the interval $(0,1]$ into sub-intervals. We use $k-1$ sub-intervals of the form $(\frac{1}{i+1},\frac{1}{i}]$ for $i = 1,\ldots,k-1$ (intervals $1,\ldots,k-1$) and one final sub-interval $[0,\frac{1}{k}]$ (interval k). Each packed bin will contain only items of chosen sizes associated with a single sub-interval. Items of chosen size which belong to sub-interval i, are packed i to a bin for $i = 1,\ldots,k-1$ (except for possibly the very last bin dedicated to this interval). Items of chosen size in interval k are packed using the greedy algorithm Next Fit. This algorithm keeps a single open bin and packs chosen sizes which belong to interval k to this bin until some chosen size does not fit. Then a new bin is opened for interval k, and the previous bin is never used again. For $1 \leq i \leq k-1$, a bin which received the full number of items (according to its type, that is, received i items) is closed, so at most $k-1$ bins are open or active simultaneously (one per interval, except for $(\frac{1}{2},1]$ which does not need an active bin).

The packing of items after configurations are chosen is similar to the algorithm of [20]. The only difference is that we may need to pack items of size zero. They are packed into bins of small items.

We define a function $f(a,b)$, where $a \in [0,1]$ and $b \geq 0$, as follows: $f(a,b) = w(a) + b$. For an item i, let (s_i,p_i) be a configuration which minimizes the function f. Then Picky Harmonic$_k$ chooses this configuration and packs item i with chosen size s_i and penalty p_i. Ties are broken arbitrarily.

As a first step to analyze the algorithm we assign costs to items (similar to weighting [27,24]). The cost of an item is a function of its allowed configurations. We say that the "chosen weight" of item i is $w(i)$ and the "chosen cost" of item i is $f(s_i,p_i)$, which is denoted by $c(i)$. We obtain the result using a sequence of lemmas.

Lemma 9. *The cost of* PH$_k$ *is at most* $\sum_{i=1}^{n} c(i) + k - 1$.

Proof. Let u_j be the number of bins used for items with chosen size in the interval $\left(\frac{1}{j+1},\frac{1}{j}\right]$ for $1 \leq j \leq k-1$ and of chosen size in $\left[0,\frac{1}{k}\right]$ for $j = k$. The cost of the algorithm is therefore $Alg \leq \sum_{i=1}^{n} p_i + \sum_{j=1}^{k} u_j \leq \sum_{i=1}^{n} (c(i) - w(i)) + \sum_{j=1}^{k} u_j$. Consider items of chosen size in $\left(\frac{1}{j+1},\frac{1}{j}\right]$ for some $1 \leq j \leq k-1$. Let n_j be the number of such items, then the sum of their chosen weights is $\frac{n_j}{j}$. We also have $u_j = \lceil \frac{n_j}{j} \rceil \leq \frac{n_j}{j} + 1$ for $2 \leq j \leq k-1$ and $u_1 = n_1$. Consider next items of chosen size in $\left[0,\frac{1}{k}\right]$. Let S be the total size of such items, then the sum of their chosen weights is $\frac{Sk}{k-1}$. Since a bin is closed only after it is filled by at least a total of $1 - \frac{1}{k}$, we also have $u_k \leq \lceil \frac{Sk}{k-1} \rceil \leq \frac{Sk}{k-1} + 1$. One special case is when $S = 0$ and yet some items of chosen size zero are packed. In this case $u_k = 1$, which means that the bound $u_k \leq \frac{Sk}{k-1} + 1$ still holds. We get $Alg \leq \sum_{i=1}^{n} (c(i) - w(i)) + \sum_{i=1}^{n} w(i) + k - 1 = \sum_{i=1}^{n} c(i) + k - 1$. □

For the analysis, we use the following well known sequence π_i, $i \geq 1$, which occurs quite often in bin packing. Let $\pi_1 = 2, \pi_{i+1} = \pi_i(\pi_i - 1) + 1$ and let $\Pi_\infty = \sum_{i=1}^{\infty} \frac{1}{\pi_i - 1} \approx 1.69103$. This sequence is presented in [20]. It is not difficult to show that $1 - \sum_{i=1}^{t} \frac{1}{\pi_i} = \frac{1}{\pi_{i+1}-1}$. It is shown in [20] that the sequence of asymptotic competitive ratios of the algorithms Harmonic$_k$ tends to Π_∞ as k grows, and that no bounded space algorithm can have an asymptotic competitive ratio smaller than Π_∞. We show that the generalized algorithm PH$_k$ has the same properties. Clearly, the lower bound for the problem with controllable sizes follows from the lower bound for classical bin packing, which is the special case where each item has a single configuration. In [20], the competitive ratio is analyzed using a weighting function. The weight of item i of size s_i is defined to be $w(s_i)$ (according to our definition of the function w as above). It is shown in [20] that the total weight of items that can fit into one

bin is at most $\Pi_\infty + \delta(k)$, where $\delta(k)$ is a non-negative function of k which tends to zero as k tends to infinity. We extend this property for our problem as follows.

Lemma 10. *Let T denote a set of items all packed in one bin by some algorithm, using a configuration (x_i,y_i) for $i \in T$. Then the sum of chosen weights of items in this bin is at most $\Pi_\infty + \delta(k)$, where $\delta(k)$ is a non-negative function of k which tends to zero as k tends to infinity.*

Proof. Since the configuration according to which each item is packed is fixed in advance, we can see the chosen sizes of items simply as items for a classical bin packing problem, and their chosen weights as their weights. Therefore, the claim is implied by the result of [20]. □

Lemma 11. *Let $\mathcal{R} \geq 1$ be a constant. Assume that we are given a set of items packed in one bin by some algorithm, using specific configurations, and for every bin packed by the algorithm, the sum of chosen weights of items assigned to this bin is at most $\mathcal{R}$. Then the competitive ratio of PH_k is at most $\mathcal{R}$.*

Proof. By Lemma 9 we have that $Alg \leq \sum_{i=1}^{n} c(i) + k - 1$. Consider an optimal packing OPTand let (x_i,y_i) be the configuration according to which OPTpacks item i. We have $c(i) \leq w(x_i) + y_i$. Let z be the number of bins used by OPTand let T_j be the subset of items packed into bin j (for an item i which is packed by OPTso that $y_i = 0$, we still assume that it is assigned a bin). Thus $Alg \leq \sum_{i=1}^{n}(w(x_i) + y_i) + k - 1 = \sum_{i=1}^{n} y_i + \sum_{j=1}^{z}\sum_{i\in T_j} w(x_i) + k - 1 \leq \sum_{i=1}^{n} y_i + z \cdot \mathcal{R} + k - 1$, while $\mathrm{OPT} = \sum_{i=1}^{n} y_i + z$. The claim follows. □

Theorem 12. *The asymptotic competitive ratio of PH_k tends to Π_∞ as k grows. No algorithm can have a smaller asymptotic competitive ratio.*

Proof. The lower bound follows from that in [20], since the standard online bounded space bin packing problem is a special case of our problem. The upper bound is a consequence of Lemmas 9,10,11. □

4. Algorithm Variable Picky Harmonic

In this section, we define an algorithm which uses as a procedure the variable Harmonic$_\varepsilon$ algorithm of Csirik [7], later analyzed by Seiden [23]. This algorithm is a generalization of the Harmonic$_k$ (with $\varepsilon = \frac{1}{k}$) algorithm of Lee and Lee [20], and takes into account the different bin sizes.

Our algorithm is called Variable Picky Harmonic$_\varepsilon$ (VPH$_\varepsilon$). We again classify by size, but the classes depend on the bin sizes. Our algorithm uses a real parameter $\varepsilon > 0$. Denote the bin set by B, and let $1 = b_1 > b_2 > \ldots b_s > 0$ be the list of bins sizes. We define the set of breakpoints for the bin size b_i as follows, $Br_i = \{\frac{b_i}{j} > \varepsilon | j \text{ is integer}\}$. We define the set of all breakpoints as $Br = \bigcup_{i=1}^{s} Br_i$. Note that Br is a set and not a multiset, that is, some breakpoints may come from more than one set Br_i. We sort Br and enumerate it as $1 = t_1 > t_2 > \ldots > t_p$. We let $t_{p+1} = \varepsilon$ and $t_{p+2} = 0$. This results in $p + 1$ intervals $I_i = (t_{i+1},t_i]$ for $1 \leq i \leq p$ and $I_{p+1} = [0 = t_{p+2},t_{p+1}]$. The algorithm keeps at most one open bin for each such interval.

We define the function w' which is based on the partition into intervals as above. For $i \leq p$, let ℓ_i be an integer such that $t_i \in Br_{\ell_i}$ and j_i be such that $j_i \cdot t_i = b_{\ell_i}$. The function w' is piecewise linear and is equal to $\frac{b_{\ell_i}}{j_i}$ in the interval I_i for $1 \leq i \leq p$, and in I_{p+1}, $w'(x) = \frac{x}{1-\varepsilon}$.

Given an item, the algorithm decides on a configuration according to which the item will be packed. We explain later how this configuration is chosen. Let such a configuration for an item i be (c_i,d_i). The item is then classified into an interval I_z according to the value c_i. If $z = p + 1$, a space of size c_i is allocated in an active bin of this interval. If a new bin needs to be opened, i.e., the size c_i is larger than the remaining space in the active bin for I_{p+1}, a new bin of size 1 is opened for this interval, and a space of size c_i is allocated there. If $z \leq p$ we act as follows. If there is no active bin for this interval, a bin of size b_{ℓ_z} is opened. Then a space of size c_i is allocated in the active bin for this interval. If as a result the bin contains j_z items, it is closed. Since $t_z = \frac{b_{\ell_z}}{j_z}$, and all spaces allocated in this bin are of size at most t_z, this process does not fail.

We define a function $f(a,b) = w'(a) + b$, similarly to the previous section, (but w' is different from w and is a function of B). For an item i, let $(a,b) = (s_i,p_i)$ be a configuration which minimizes the function f. Then Variable Picky Harmonic$_\varepsilon$ chooses this configuration and packs item i with chosen size s_i and penalty p_i. Ties are broken arbitrarily. We assign costs to items. The cost of an item is a function of its allowed configurations. We say that the "chosen weight" of item i is $w'(i) = w'(s_i)$ and the "chosen cost" of item i is $f(s_i,p_i)$, which is denoted by $c(i)$. The result follows from the following lemmas.

Lemma 13. *The cost of* VPH_ε *is at most* $\sum_{i=1}^{n} c(i) + \frac{s}{\varepsilon}$.

Proof. We start with bounding the value p. We have $p = |Br| \leq \sum_{i=1}^{s} |Br_i| \leq \sum_{i=1}^{s} \lfloor \frac{1}{\varepsilon} \rfloor \leq \frac{s}{\varepsilon}$.

Let u_z be the number of bins used for items with chosen size in the interval I_z for $1 \leq z \leq p+1$. The cost of the algorithm is therefore $Alg \leq \sum_{i=1}^{n} p_i + \sum_{z=1}^{p+1} u_z \cdot b_{\ell_i} \leq \sum_{i=1}^{n} (c(i) - w'(i)) + \sum_{z=1}^{p+1} u_z \cdot b_{\ell_i}$. Consider items of chosen size in I_z for some $1 \leq z \leq p$. Let n_z be the number of such items, then the sum of their chosen weights is $\frac{b_{\ell_z} \cdot n_z}{j_z}$. We also have $u_z = \lceil \frac{n_z}{j_z} \rceil \leq \frac{n_z}{j_z} + 1$ and $u_z \cdot b_z \leq \frac{b_z \cdot n_z}{j_z} + b_z \leq \frac{b_z \cdot n_z}{j_z} + 1$. Note that for $z = 1$, we have $j_1 = 1$ and $u_1 = n_1$. Consider next items of chosen size in I_{p+1}. Let S be the total size of such items, then the sum of their chosen weights is $\frac{S}{1-\varepsilon}$. Since a bin is closed only after some chosen size (in I_{p+1}) does not fit, it is filled by at least a total of $1 - \varepsilon$, we also have $u_{p+1} \leq \lceil \frac{S}{1-\varepsilon} \rceil \leq \frac{S}{1-\varepsilon} + 1$. One special case is when $S = 0$ and yet some items of chosen size zero are packed. In this case $u_{p+1} = 1$, which means that the bound $u_{p+1} \leq \frac{S}{1-\varepsilon} + 1$ still holds. We get $Alg \leq \sum_{i=1}^{n} (c(i) - w(i)) + \sum_{i=1}^{n} w(i) + p \leq \sum_{i=1}^{n} c(i) + \frac{s}{\varepsilon}$. □

Lemma 14. *Consider the algorithm for a parameter* $\varepsilon > 0$. *For a bin set B, let* $\mathcal{W}$ *be an upper bound on the maximum ratio for any bin size, between the total weight of items that can be assigned to this bin and the size of the bin (which can be computed by the mathematical program in [23]). Let T denote a set of items all packed in one bin of size* b_j *by some algorithm, using a configuration* (x_i, y_i) *for* $i \in T$. *Then the sum of chosen weights of items in this bin is at most* $b_j \mathcal{W}$.

Proof. Since the configuration according to which each item is packed is fixed in advance, we can see the chosen sizes of items as items for a classic bin packing problem, and their chosen weights as their weights. □

Lemma 15. *Let* $\mathcal{R} \geq 1$ *be a constant. Assume that we are given a set of items packed in one bin of size* b_j *by some algorithm, using specific configurations, and for every bin packed by the algorithm, the sum of chosen weights of items assigned to this bin is at most* $b_j \mathcal{R}$. *Then the competitive ratio of* VPH_ε *is at most* $\mathcal{R}$.

Proof. Using Lemma 13 we have $Alg \leq \sum_{i=1}^{n} c(i) + \frac{s}{\varepsilon}$. Consider an optimal packing OPTand let (x_i, y_i) be the configuration according to which OPTpacks item i. We have $c(i) \leq w(x_i) + y_i$. Let z_m be the number of bins of size b_m used by OPT, and let $T_{j,m}$ be the subset of items packed into the j-th bin of size b_m, for $1 \leq m \leq s$. (For an item i which is packed by OPTso that $y_i = 0$, we assume that it is assigned a bin.) Thus $Alg \leq \sum_{i=1}^{n} w(x_i) + y_i + \frac{s}{\varepsilon} = \sum_{i=1}^{n} y_i + \sum_{m=1}^{s} \sum_{j=1}^{z_m} \sum_{i \in T_{j,m}} w(x_i) + \frac{s}{\varepsilon} \leq \sum_{i=1}^{n} y_i + \sum_{m=1}^{s} \sum_{j=1}^{z_m} b_m \cdot \mathcal{R} + \frac{s}{\varepsilon} \leq \sum_{i=1}^{n} y_i + \sum_{m=1}^{s} z_m \cdot b_m \cdot \mathcal{R} + \frac{s}{\varepsilon}$. However, we have $\mathrm{OPT} = \sum_{i=1}^{n} y_i + \sum_{m=1}^{s} z_m \cdot b_m$, the claim follows. □

Theorem 16. *The asymptotic competitive ratio of* VPH_ε *is the same as the competitive ratio of the algorithm Variable Harmonic with the same value of* ε. *No algorithm can have a smaller asymptotic competitive ratio.*

Proof. The lower bound follows from the lower bound proved in [23], since the standard variable sized online bounded space bin packing problem is a special case of our problem. The upper bound is a consequence of Lemmas 13,14,15. □

5. Online bin covering

In this section we study the dual problem, which is a generalized bin covering problem. We start by a simple adaptation of the algorithm of [1] for the case of unit bins.

5.1. Algorithm Picky Cover

The algorithm Picky Cover (PC) acts as follows. We define a function $h(a,b)$, where $a \in [0,1]$ and $b \geq 0$, as follows. $h(a,b) = a + b$. For an item i, Let (s_i, p_i) be a configuration which maximizes the function h. Then Picky Cover chooses this configuration and packs item i with chosen size s_i and value p_i. The algorithm keeps a single active bin, and packs the sizes s_i into this bin. Once the bin is covered, that is, contains a total of at least 1, this bin is closed and a new one is opened.

We prove the following theorem.

Theorem 17. *The competitive ratio of* PC *is* 2 *and this is the best possible ratio*.

Proof. The lower bound follows from the result of [8], since standard online bin covering is a special case where each item has a single configuration with its original size, and value zero. To prove the upper bound, consider an optimal solution OPT. Denote the configuration used by OPT for item i by (x_i,y_i). We have $\mathrm{OPT} \leq \sum_{i=1}^{n}(x_i+y_i) \leq \sum_{i=1}^{n} h(s_i,p_i) = \sum_{i=1}^{n}(s_i+p_i)$. This is true by the choice of configurations (s_i,p_i).

Since $s_i \leq 1$ for all i, no bins that the algorithm covers is covered by a total of more than 2. Thus, the number of bins that are successfully covered is at least $\frac{\sum_{i=1}^{n} s_i - 1}{2}$. The total value of the algorithm is therefore at least $Alg \geq \sum_{i=1}^{n} p_i + \frac{\sum_{i=1}^{n} s_i - 1}{2} \geq \frac{\sum_{i=1}^{n}(s_i+p_i)-1}{2} \geq \frac{\mathrm{OPT}}{2} - \frac{1}{2}$ □

5.2. *Algorithm* Variable Picky Cover

As in the previous cases, our algorithm first applies a decision rule, and then uses a known algorithm. In our case this is the algorithm of [31] (which is the same as the greedy algorithm of [1], in the special case of unit bins).

The algorithm Variable Picky Cover (PC) acts as follows (again using $h(a,b)$). For an item i, let (s_i,p_i) be a configuration which maximizes the function h. Then Picky Cover chooses this configuration and packs item i with chosen size s_i and value p_i. The packing is performed as in [31]. We next describe the packing method.

Recall that the bin sizes which are of size at least $1/2$ are denoted $b_1 > b_2 > \ldots > b_k > 1/2$, and $b_{k+1} = 1/2$. We define $q = \max_{i=1,\ldots,k}\{b_i/b_{i+1}\}$ and $m = -\log_2(q-1)$. Note that due to the definition of q, m is polynomial in the length of the input. Thus we have $b_i \leq qb_{i+1}$ for $i < k$ and $b_k \leq qb_{k+1} = qb_1/2$. Note that this definition is valid since $1 < q \leq 2$.

We use $mk+1$ intervals, where the items of each interval are packed independently of other items. Interval $I(i,j)$, for $1 \leq i \leq k$, $0 \leq j \leq m-1$ is defined as $(\frac{b_{i+1}}{2^j}, \frac{b_i}{2^j}]$ and interval $I_{last} = [0, \frac{1}{2^m}]$. The algorithm keeps an active bin for each interval, and packs all items of size in this interval into the bin. For interval $I(i,j)$, the bin is of size b_{i+1} if $i < k$ and of size $b_1 = 1$ otherwise. For interval I_{last} this is a bin of size $b_1 = 1$.

Once a bin of size b_i is covered, that is, contains a total of at least b_i, this bin is closed and a new one of the same size is opened for the respective interval. Note that a bin for interval $I(i,j)$ is covered if exactly 2^j items were assigned to it, if $i < k$ and otherwise if 2^{j+1} items were packed into it. Therefore, each bin, which belongs to interval $I(i,j)$ for some $1 \leq i < k$ (and has size b_{i+1}) is covered by at least a total of b_{i+1} and at most a total of $b_i \leq qb_{i+1}$, and each bin which belongs to interval $I(k,j)$ (and has size $b_1 = 1$) is covered by at least a total of 1 and at most $2b_k \leq qb_1$.

This is true for all bins except one last bin from each interval. We prove the following theorem.

Theorem 18. *The competitive ratio of* VPC *is* q *and this is the best possible ratio*.

Proof. The lower bound follows from the result of [31], since it is proved for a special case of our problem. To prove the upper bound, consider an optimal solution OPT. Denote the configuration used by OPTfor item i by (x_i,y_i). Let V_{Alg} and V_{OPT} be the total size of bins covered by the algorithm we defined and an optimal offline algorithm respectively.

We have $\mathrm{OPT} = \sum_{i=1}^{n} x_i + V_{\mathrm{OPT}} \leq \sum_{i=1}^{n}(x_i+y_i) \leq \sum_{i=1}^{n} h(s_i,p_i) = \sum_{i=1}^{n}(s_i+p_i)$. This is true by the choice of configurations (s_i,p_i). Since the contents of no covered bin exceed its size by a factor of more than q, and no more than $km+1$ additional bins were used to pack items, but not covered, we also have $\sum_{i=1}^{n} s_i \leq qV_{Alg} + mk + 1$. Since $Alg = V_{Alg} + \sum_{i=1}^{n} p_i$, we have, $\mathrm{OPT} \leq qV_{Alg} + mk + 1 + \sum_{i=1}^{n} p_i \leq qAlg + mk + 1$.

Since for a given set of bin sizes, m and k are constant, we get that the algorithm has an asymptotic competitive ratio of at most q. □

6. Concluding remarks

We have designed an augmented APTAS for *bin packing with controllable item sizes*. Such an APTAS allows to relax the problem in two ways. The algorithm both uses larger bins (of size $1+\varepsilon$) than the ones used by an optimal solution OPT, and the number of bins used is slightly larger (by a factor of $1+\varepsilon$ plus an additive constant) than the number of bins in OPT.

An open question is whether it possible to use only one such relaxation rather than both of them, that is, whether a dual PTAS exists for the problem, and whether a standard APTAS for the problem can be designed.

Typically, bin packing problems with more than one parameter become hard; this is the case for 2-dimensional vector packing and for 2-dimensional bin packing [30,2,5]. On the other hand, some such problems admit an APTAS and an AFPTAS, such as bin packing with rejection [13,3,14] and bin packing with cardinality constraints [4,14].

7. Acknowledgment

We thank two anonymous referees for carefully reading the manuscript and helping us improving the readability of the paper.

References

[1] S.F. Assmann, D.S. Johnson, D.J. Kleitman, J.Y.-T. Leung, On a dual version of the one-dimensional bin packing problem, Journal of Algorithms, 5 (1984) 502–525.
[2] N. Bansal, J.R. Correa, C. Kenyon, M. Sviridenko, Bin packing in multiple dimensions: inapproximability results and approximation schemes, Mathematics of Operations Research 31 (2006) 31–49.
[3] W. Bein, J.R. Correa, X. Han, A fast asymptotic approximation scheme for bin packing with rejection, Theoretical Computer Science 393 (2008) 14–22.
[4] A. Caprara, H. Kellerer, U. Pferschy, Approximation schemes for ordered vector packing problems, Naval Research Logistics 92 (2003) 58–69.
[5] M. Chlebík, Janka Chlebíková, Inapproximability results for orthogonal rectangle packing problems with rotations, in: Proceedings of the 6th Italian Conference on Algorithms and Complexity (CIAC2006), 2006, pp. 199–210.
[6] E.G. Coffman, M.R. Garey, D.S. Johnson, Approximation algorithms for bin packing: a survey, in: D. Hochbaum (Ed.), Approximation Algorithms, PWS Publishing Company, 1997.
[7] J. Csirik, An online algorithm for variable-sized bin packing, Acta Informaticae 26 (1989) 697–709.
[8] J. Csirik, V. Totik, On-line algorithms for a dual version of bin packing, Discrete Applied Mathematics 21 (1988) 163–167.
[9] J. Csirik, G.J. Woeginger, On-line packing and covering problems, in: A. Fiat, G.J. Woeginger (Eds.), Online Algorithms: The State of the Art, 1998, pp. 147–177.
[10] P. De, E.J. Dunne, J.B. Ghosh, C.E. Wells, Complexity of the discrete time–cost tradeoff problem for project networks, Opererations Research 45 (1997) 302–306.
[11] W. Fernandez de la Vega, G.S. Lueker, Bin packing can be solved within $1+\varepsilon$ in linear time, Combinatorica 1 (1981) 349–355.
[12] G. Dósa, Y. He, Bin packing problems with rejection penalties and their dual problems, Information and Computation 204 (2006) 795–815.
[13] L. Epstein, Bin packing with rejection revisited, Algorithmica, in press.
[14] L. Epstein, A. Levin, AFPTAS results for common variants of bin packing: a new method to handle the small items, Manuscript.
[15] D.K. Friesen, M.A. Langston, Variable sized bin packing, SIAM Journal on Computing 15 (1986) 222–230.
[16] D.S. Hochbaum, D.B. Shmoys, Using dual approximation algorithms for scheduling problems: theoretical and practical results, Journal of the ACM 34 (1987) 144–162.
[17] D.S. Johnson, A. Demers, J.D. Ullman, Michael R. Garey, Ronald L. Graham, Worst-case performance bounds for simple one-dimensional packing algorithms, SIAM Journal on Computing 3 (1974) 256–278.
[18] D.S. Johnson, Fast algorithms for bin packing, Journal of Computer and System Sciences 8 (1974) 272–314.
[19] N. Karmarkar, R.M. Karp, An efficient approximation scheme for the one-dimensional bin-packing problem, in: Proceedings of the 23rd Annual Symposium on Foundations of Computer Science, 1982, pp. 312–320.
[20] C.C. Lee, D.T. Lee, A simple online bin packing algorithm, Journal of the ACM 32 (1985) 562–572.
[21] H.W. Lenstra Jr., Integer programming with a fixed number of variables, Mathematics of Operations Research 8 (1983) 538–548.
[22] P. Ramanan, D.J. Brown, C.C. Lee, D.T. Lee, Online bin packing in linear time, Journal of Algorithms 10 (1989) 305–326.
[23] S.S. Seiden, An optimal online algorithm for bounded space variable-sized bin packing, SIAM Journal on Discrete Mathematics 14 (2001) 458–470.
[24] S.S. Seiden, On the online bin packing problem, Journal of the ACM 49 (2002) 640–671.
[25] S.S. Seiden, R. van Stee, L. Epstein, New bounds for variable-sized online bin packing, SIAM Journal on Computing 32 (2003) 455–469.
[26] M. Skutella, Approximation algorithms for the discrete time–cost tradeoff problem, Mathematics of Operations Research 23 (1998) 909–929.
[27] J.D. Ullman, The performance of a memory allocation algorithm, Tech. Report 100, Princeton University, Princeton, NJ, 1971.
[28] A. van Vliet, An improved lower bound for online bin packing algorithms, Information Processing Letters 43 (1992) 277–284.
[29] G.J. Woeginger, Improved space for bounded-space online bin packing, SIAM Journal on Discrete Mathematics 6 (1993) 575–581.
[30] G.J. Woeginger, There is no asymptotic ptas for two-dimensional vector packing, Information Processing Letters 64 (1997) 293–297.
[31] G.J. Woeginger, G. Zhang, Optimal on-line algorithms for variable-sized bin covering, Operations Research Letters 25 (1999) 47–50.