Fast Asymptotic FPTAS for Packing Fragmentable Items with Costs

Hadas Shachnai and Omer Yehezkely

Department of Computer Science, The Technion, Haifa 32000, Israel.
E-mail: {hadas,omery}@cs.technion.ac.il.

Abstract. Motivated from recent applications in community TV networks and VLSI circuit design, we study variants of the classic bin packing problem, in which a set of items needs to be packed in a minimum number of unit-sized bins, allowing items to be *fragmented*. This can potentially reduce the total number of bins used, however, item fragmentation does not come for free. In *bin packing with size preserving fragmentation (BP-SPF)*, there is a bound on the total number of fragmented items. In *bin packing with size increasing fragmentation (BP-SIF)*, fragmenting an item increases the input size (due to a header/footer of fixed size that is added to each fragment). Both BP-SPF and BP-SIF do not belong to the class of problems that admit a *polynomial time approximation scheme (PTAS)*.
In this paper, we develop fast *asymptotic fully polynomial time approximation schemes (AFPTAS)* for both problems. The running times of our schemes are *linear* in the input size. As special cases, our schemes yield AFPTASs for classical bin packing and for variable-sized bin packing, whose running times improve the best known running times for these problems.

1 Introduction

In the classical *bin packing* problem, n items $(a_1, \ldots, a_n)$ of sizes $s(a_1), \ldots, s(a_n) \in (0, 1]$ need to be packed in a minimal number of unit-sized bins. Bin packing is well known to be NP-hard. We consider variants of bin packing in which items may be fragmented (into two or more pieces). This can potentially reduce the number of bins used; however, item fragmentation does not come for free. We study the following two variants.

Size preserving fragmentation (BP-SPF): an item a_i can be split into two fragments: a_{i_1}, a_{i_2}, such that $s(a_i) = s(a_{i_1}) + s(a_{i_2})$. The resulting fragments

can also split in the same way. Each split has a unit cost and the total cost cannot exceed a given *budget* $C \geq 0$. In the special case where $C = 0$ we get an instance of classical bin packing.

Size increasing fragmentation (BP-SIF): a header (or a footer) of fixed size, $\Delta > 0$, is attached to each (whole or fragmented) item; thus, the capacity required for packing an item of size $s(a_i)$ is $s(a_i) + \Delta$. Upon fragmenting an item, each fragment gets a header; that is, if a_i is replaced by two items such that $s(a_i) = s(a_{i_1}) + s(a_{i_2})$, then packing a_{i_j} requires capacity $s(a_{i_j}) + \Delta$.

The above two variants of the bin packing problem capture many practical scenarios, including message transmission in community TV networks, VLSI circuit design and preemptive scheduling on parallel machines with setup times/setup costs. For more details on these applications see [26].

Both BP-SIF and BP-SPF are known to be NP-hard (see in [17] and [26]), therefore, we focus on finding efficient approximate solutions.

1.1 Related Work

It is well known (see, e.g., [20]) that bin packing does not belong to the class of NP-hard problems that admit a PTAS. In fact, bin packing cannot be approximated within factor $\frac{3}{2} - \varepsilon$, for any $\varepsilon > 0$, unless P=NP [9]. However, there exists an *asymptotic* PTAS *(APTAS)* which uses, for any instance I, $(1 + \varepsilon)\, OPT(I) + k$ bins for some fixed k, where $OPT(I)$ is the number of bins used in any optimal solution. Fernandez de la Vega and Lueker [6] presented an APTAS with $k = 1$. Alternatively, a *dual* PTAS, which uses $OPT(I)$ bins of size $(1 + \varepsilon)$ was given by Hochbaum and Shmoys [12]. Such a dual PTAS can also be derived from the work of Epstein and Sgall [5] on multiprocessor scheduling, since bin packing is dual to the *minimum makespan* problem.

Karmarkar and Karp [14] presented an AFPTAS for bin packing which uses $(1+\varepsilon)OPT(I)+O(1/\varepsilon^2)$ bins. The scheme of [14] is based on rounding the (fractional) solution of a *linear programming (LP)* relaxation of bin packing. To solve this linear program in polynomial time, despite the fact that it has a vast number of variables, the authors use a variant of the ellipsoid method due to Grötschel, Lovász and Schrijver (GLS) [11]. The resulting running time is $O(\varepsilon^{-8} n \log n)$. An AFPTAS with substantially improved running time of $O(n \log \varepsilon^{-1} + \varepsilon^{-6} \log^6 \varepsilon^{-1})$ was proposed by Plotkin et al. [24].

In *variable-sized bin packing*, we have a set of items whose sizes are in $(0, 1]$, and a set of bin sizes in $(0, 1]$ (including the size 1) available for packing the items. We need to pack the items in a set of bins of the given sizes, such that the total bin capacity used is minimized. The variable-sized bin packing problem

was first investigated by Friesen and Langston [8], who gave several approximation algorithms, the best of which has asymptotic worst case ratio of 4/3. Murgolo [22] presented an AFPTAS, which solves a covering linear program using the techniques of [14], namely, the dual program is solved approximately using the modified GLS algorithm. Comprehensive surveys on the bin packing problem and its variants appear, e.g., in [2, 27] (see also the recent work of Epstein and Levin [4] on dynamic approximation schemes for the *online* bin packing problem, and the references therein.)

Mandal et al. introduced in [17] the BP-SIF problem and showed that it is NP-hard. Menakerman and Rom [19] and Naaman and Rom [23] were the first to develop algorithms for bin packing with item fragmentation, however, the problems studied in [19] and [23] are different from our problems. For a version of BP-SPF in which the number of bins is given, and the objective is to minimize the total cost incurred by fragmentation, the paper [19] studies the performance of simple algorithms such as First-Fit, Next-Fit and First-Fit-Decreasing, and shows that for any instance which can be packed in N bins using f^* splits of items, each of these algorithms might end up using $f^* + N - 1$ splits.

The paper [26] presents dual PTASs and APTASs for BP-SPF and BP-SIF. The dual PTASs pack all the items in $OPT(I)$ bins of size $(1+\varepsilon)$, and the APTASs use at most $(1+\varepsilon)OPT(I)+1$ bins. All of these schemes have running times that are polynomial in n and exponential in $1/\varepsilon$. The paper also shows that each of the problems admits a *dual* AFPTAS. The proposed schemes pack the items in $OPT(I) + O(1/\varepsilon^2)$ bins of size $(1+\varepsilon)$. The schemes are based on solving mixed packing and covering LP with negative entries, using the ellipsoid method; this results in the running time of $O(\varepsilon^{-12} n \log n)$, which renders the schemes highly impractical. The question whether the two variants of bin packing admitted *(non-dual)* AFPTASs remained open. In this paper, we resolve this question by presenting for the two problems approximation schemes which pack the items in $(1+\varepsilon)OPT(I) + O(\varepsilon^{-1}\log\varepsilon^{-1})$ unit-sized bins. The proposed schemes improve significantly the running times of the schemes in [26], to times that are *linear* in the input size. Since these schemes are *combinatorial*, they are also easier to implement.

There has been some related work in the area of preemptive scheduling on parallel machines. The paper [25] presents a tight bound on the number of preemptions required for a schedule of minimum makespan, and a PTAS for minimizing the makespan of a schedule with job-wise or total bound on the number of preemptions. For the special case of preemptive scheduling of jobs on identical parallel machines, so as to minimize the *makespan*, the paper uses the property of *primitive* optimal schedules for developing an approximation scheme; however,

the scheme is based on dynamic programming applied to a *discretized* instance. Such discretization cannot be applied to our problems, since the resulting packing may exceed the bin capacities.

1.2 Our Results

In this paper we develop *fast* AFPTASs for BP-SPF and BP-SIF. Our schemes pack the items in $(1+\varepsilon)OPT(I) + O(\varepsilon^{-1}\log\varepsilon^{-1})$ bins, in time that is *linear* in n, the number of items, and polynomial in $1/\varepsilon$ (see Theorems 1 and 4). As special cases, our scheme for BP-SPF yields AFPTASs for classical bin packing and for variable-sized bin packing, whose running times improve the best known running times for these problems (see Theorems 2 and 3). We note that the improvement for variable-sized bin packing is substantial, since the best known running time for this problem is dominated by the running time of the modified GLS algorithm, which is $O(\varepsilon^{-8}n\log n)$ [22].

Techniques: A major difficulty in packing with item fragmentation is in defining the *fragment sizes*. One possible approach is to discretize these sizes (as in [26]); however, this leads to a *dual* AFPTAS. Another approach is to *guess* the bin configurations and to solve a linear program to find the fragment sizes, but then the scheme cannot be *fully* polynomial, as the number of configurations that need to be considered is exponential in $1/\varepsilon$. To get around this difficulty, we initially transform the input I to one that can be packed with *no fragmentation*; later, we transform the resulting packing to a valid packing of I. We use an interesting structural property of the two variants of bin packing, namely, the existence of optimal *primitive* packings (see Section 2.1), to establish a relation between packing with item fragmentation and variable-sized bin packing. In particular, we model any feasible packing satisfying this property as an undirected graph; each connected component is represented by an *oversized* bin, whose size is equal to the number of bins in this component. The items are then packed in bins of variable sizes, with no fragmentation. The scheme completes by generating from each oversized bin a set of ordinary (i.e., unit-sized) bins, allowing some of the items to fragment, while maintaining the constraints on the number of bins/fragmented items. We expect that our non-standard approach will find more uses for other problems that involve splitting of input elements, such as preemptive scheduling of jobs on parallel machines. Recently, Fishkin et al. [7] studied this problem where preemptions incur delays, due to *migrations* of jobs from one machine to another, and showed the existence of optimal *primitive* schedules for certain subclasses of instances. More generally, our schemes may be useful for other scheduling problems in which there exist optimal schedules satisfying McNaughton's rule [18].

Due to space constraints, some of the proofs are omitted. Other proofs are relegated to the Appendix.

2 An AFPTAS for BP-SPF

2.1 Preliminaries

At the heart of our schemes lies an interesting structural property of optimal solutions for our problems. Define the *bin packing graph* of a given packing as an undirected graph, where each bin i is represented by a vertex v_i; there is an edge (v_i, v_j) if bin i and bin j share fragments of the same item. Note that a fragment-free packing induces a graph with no edges. A *primitive packing* is a feasible packing in which (i) each bin has at most two fragments of items, (ii) each item can be fragmented only once, and (iii) the respective bin packing graph is a collection of paths. Note that the last condition implies that in any connected component of the bin packing graph there are two bins including only a single fragment. The following was shown in [26].

Lemma 1. *Any instance of BP-SPF has an optimal primitive packing.*

Consider a primitive solution for a BP-SPF instance. Suppose that some connected component consists of $c \geq 1/\varepsilon$ bins, then we can partition this component to εc components, each containing $1/\varepsilon$ bins. To avoid violating the budget of C on the number of fragmented items, whenever we need to fragment an item, we can add a new bin in which the item is packed with no fragmentation. Overall, we may add at most εc new bins. Thus, we have shown

Lemma 2. *Any feasible solution for BP-SPF which uses N bins can be replaced by a solution in which each connected component is of size at most $\lceil 1/\varepsilon \rceil$.*

Recall that in BP-SPF we are given a set of n items $I = (a_1, a_2, ..., a_n)$, where a_i has the size $s(a_i) \in (0, 1]$. The number of fragmented items is bounded by C. The goal is to pack all items using minimal number of bins and at most C splits. The following is an outline of our scheme for BP-SPF. (i) Preprocess the input to obtain a *fixed* number of item sizes. (ii) Guess $OPT(I) = d$, the number of bins used by an optimal packing of I. (iii) Solve a linear programming relaxation of the resulting instance of packing the items (with no fragmentations) into *ordinary* and *oversized* bins. (iv) Round the (fractional) solution for the LP and pack the large items according to the integral solution. (v) Pack the remaining large items and the small items, one at a time (see below).

2.2 Preprocessing the Input

Initially, we partition the items into two groups by their sizes: the *large* items have size at least ε; all other items are *small*. We then transform the instance I to an instance I' in which the number of distinct items sizes is fixed. This can be done by using the shifting technique (see, e.g., in [27]). Generally, the items are sorted in non-decreasing order by sizes, then, the ordered list is partitioned into at most $1/\varepsilon^2$ subsets, each containing $H = \lceil n\varepsilon^2 \rceil$ items. The size of each item is rounded up to the size of the largest item in its subset. This yields an instance in which the number of distinct item sizes is $m = n/H \leq 1/\varepsilon^2$. Denote by n_j the number of items in size group j, $1 \leq j \leq m$.

We proceed to define for the large items the *configuration matrix*, A, for bins in which there are no fragmented items. In particular, for the shifted large items, a *bin configuration* is a vector of size $m \leq 1/\varepsilon^2$, in which the j-th entry gives h_j, the number of items of size group j packed in the bin. The configuration matrix A consists of the set of all possible bin configurations, where each configuration is a column in A; therefore, the number of columns in A is $q \leq (1/\varepsilon)^{1/\varepsilon^2}$. Next, we define a matrix B including as columns the configurations of *oversized* bins. Each of these bins has a size in the range $[2, C+1]$. An oversized bin represents a connected component in some optimal primitive packing of the input. Each configuration of an oversized bin gives the number of items of each size group in this bin; thus, the number of columns in B is $s = O((C/\varepsilon)^{1/\varepsilon^2})$.

2.3 Solving the Linear Program

In the following we describe the linear program that we formulate for finding a (fractional) packing of the items with no fragmentation. We number the oversized bin configurations by $1, \ldots, s$. Note that the capacities of the oversized bins used will determine the number of fragmented items in the solution output by the scheme. Let f_k be the minimum capacity of an oversized bin having the k-th configuration, then the number of fragmented items in the connected component corresponding to this bin is $f_k - 1$. Denote by $\mathbf{x}, \mathbf{y}$ the variable vectors giving the number of ordinary and oversized bins having certain configuration, respectively. We want to minimize the total bin capacity used for packing the input, such that the number of fragmented items does not exceed the budget C.

Having guessed correctly the value of d, we need to find a feasible solution for the following program.

$$
\begin{aligned}
(LP) \qquad & \sum_{i=1}^{q} A_{ji} x_i + \sum_{k=1}^{s} B_{jk} y_k \geq n_j \qquad \text{for } j = 1, \ldots, m \\
& \sum_{i=1}^{q} x_i + \sum_{k=1}^{s} f_k y_k \leq d \\
& \sum_{k=1}^{s} (f_k - 1) y_k \leq C \\
& x_i \geq 0 \qquad \text{for } i = 1, \ldots, q \\
& y_k \geq 0 \qquad \text{for } k = 1, \ldots, s
\end{aligned}
$$

The first set of constraints guarantees that we pack all the items in size group j, for all $1 \leq j \leq m$; the two last constraints guarantee that the total number of bins used is at most d, and that the number of fragmented items does not exceed the budget C.

A technique developed by Young [28], for obtaining fast approximately feasible solutions for mixed linear programs, yields a solution which may violate the packing constraints in the above program at most by factor of ε, namely, $\sum_i x_i + \sum_k f_k y_k \leq d(1+\varepsilon)$, and $\sum_k (f_k - 1) y_k \leq C(1+\varepsilon)$ (see also in [13]).[1] Generally, the technique is based on repeated calls to an oracle, which solves in each iteration (one or more) instances of the knapsack problem. Thus, the heart of the scheme is in efficient implementation of the oracle. In solving the LP for our problem, the oracle needs to solve a set of $C+1$ instances of the *multiple choice knapsack (MCK)* problem. In the c-th instance, $1 \leq c \leq C+1$, we are given a bin of capacity c and m items; the i-th item, a_i, has the size $\varepsilon \leq s(a_i) \leq 1$ and the profit $0 < p(a_i) \leq 1$. We need to find a feasible packing of the items in the bin, by selecting any number of copies of each item, such that the overall profit is maximized. The oracle should solve this problem within $1+\varepsilon$ from the optimal, for each bin size $1 \leq c \leq C+1$. We call this set of instances of the MCK problem *all_MCK*.

By Lemma 2, we may assume that the maximum bin size in our all_MCK problem is $\lceil 1/\varepsilon \rceil + 1$. Since each item a_i, has a size $s(a_i) \geq \varepsilon$, we can pack at most $(1/\varepsilon)^2$ copies of each item. Therefore, for each bin size $1 \leq c \leq \lceil 1/\varepsilon \rceil$, we can solve the knapsack problem with $n = m/\varepsilon^2$ items. Using a fast FPTAS of Kellerer and Pferscy [15], this can be done in $O(\varepsilon^{-3} m \log \varepsilon^{-1})$ steps.

[1] We show below that such a solution can be fixed to maintain the budget constraint in our BP-SPF instance.

Alternatively, we can solve each MCK instance exactly, by formulating the problem, with a bin of size c, as the following *integer program (IP).*

$$\{maximize \sum_{i=1}^{m} p(a_i)z_i \;\; subject\ to: \sum_{i=1}^{m} s(a_i)z_i \leq c,\ z_i \geq 0\ 1 \leq i \leq m\}$$

where z_i is the number of copies selected from the i-th item. As shown in [3], such IP in fixed dimension can be optimally solved in $O(mM)$ steps, where M is the longest binary representation of any input element.[2] We repeat this for $c = 1, 2, \ldots, C+1$. Thus, we have shown

Lemma 3. *For any $\varepsilon > 0$, the above all_MCK problem can be solved within factor of $(1+\varepsilon)$ from the optimal in $O(\frac{m}{\varepsilon} \cdot \min\{\varepsilon^{-2}\log\varepsilon^{-1}, M\})$ steps, where M is the longest binary representation of any input element.*

Finally, given a feasible $(1+\varepsilon)$-approximate solution for LP, we apply a technique of Beling and Megiddo [1] for transforming a given solution for a system of linear equations to a *basic* solution.

2.4 Packing the Items

Given the (fractional) solution for LP, we obtain integral vectors $\mathbf{x}'$ and $\mathbf{y}'$, by rounding down the entries in the vectors $\mathbf{x}$ and $\mathbf{y}$, respectively; that is, $\mathbf{x}' = \lfloor \mathbf{x} \rfloor$, and $\mathbf{y}' = \lfloor \mathbf{y} \rfloor$. Note that since we have a basic solution, at most $m+2$ variables can be assigned non-zero values. We start by packing the items according to the rounded solution (defined by $\mathbf{x}'$ and $\mathbf{y}'$), using the respective bin configurations. Next, for each non-integral variable y_k, we add a bin of size $\lfloor (y_k - y'_k) \cdot f_k \rfloor$. Denote this set of new bins by R. We proceed by packing the remaining large items into R, using the First Fit algorithm. Note that the number of (unit-sized) bins used so far does not exceed the number of bins in the solution defined by $\mathbf{x}$ and $\mathbf{y}$. If any large item remains unpacked, we add unit-sized bins in which these items are packed using First Fit. We then add the small items in arbitrary order, using Next Fit.[3]

Our scheme completes by dividing each of the oversized bins into a set of ordinary bins. In the following we show that we can pack all items efficiently.

Lemma 4. *Using the rounded solution for the LP, we can pack all items in the instance in $O(n + m\log m)$ steps.*

[2] An instance solved by the oracle consists of a set of rationals that give $s(a_i), p(a_i)$ for $1 \leq i \leq m$. The size of the binary representation of each rational is the sum of the sizes of its numerator and denominator.

[3] Detailed descriptions of First Fit and Next Fit can be found, e.g., in [2].

3 Analysis of the Scheme

We first show that, although (i) we obtain only *approximately* feasible solution for LP, and (ii) rounding the solution for LP may result in adding oversized bins, the packing output by our scheme maintains the budget constraint.

Lemma 5. *The total number of fragmented items is at most C.*

Proof. Recall that the technique in [28] yields an *approximately feasible* solution, in which the packing constraints may be violated by factor of ε. Thus, given an approximately feasible solution for LP, either the number of bins or the number of fragmented items may be increased by factor of ε. We note that any extra fragmentation can be replaced by an extra bin, namely, we can pack the fragmented item in a new bin with no fragmentation. This will result in at most $\varepsilon \cdot OPT(I)$ new bins. Therefore we may assume that in our solution for the LP at most C items are fragmented.

Next, we show that while packing the items using the (rounded) integral solution, we do not exceed the bound of C. In particular, we need to show that the number of items fragmented due to the usage of extra bins while packing the large items, is at most C. Consider the variable y_k, and let $0 < g_k = y_k - \lfloor y_k \rfloor < 1$. Recall that f_k is the capacity of the oversized bin having the k-th configuration, where $1 \leq f_k \leq C + 1$. Then, we define a new oversized bin whose capacity is $\lfloor f_k g_k \rfloor$. The number of fragmented items due to this new bin is $\lfloor f_k g_k \rfloor - 1 \leq (f_k - 1) \cdot g_k$. Since the right hand side in the last inequality is the number of items fragmented due to the fractional part of y_k (in the solution for LP), we have not increased the total number of fragmented items. This holds for any $1 \leq k \leq s$. ∎

The proof of the next lemma is given in the Appendix.

Lemma 6. *The input is packed in at most $(1+\varepsilon)^2(1+2\varepsilon)OPT(I) + 4(m+2)$ bins.*

We now analyze the running time of the scheme.

Lemma 7. *The above scheme can be implemented in linear time.*

Proof. The running time of the scheme is determined by the following steps. (i) Preprocess the input: we first partition the items to 'large' and 'small'; then, we apply shifting to the large items to obtain $m = O(1/\varepsilon^2)$ distinct items sizes.

This can be implemented in $O(n \log m)$ steps (see, e.g., in [24]). (*ii*) Applying the technique of Young [28] for solving the LP, we get that the number of calls to the oracle is $O(m \log m \varepsilon^{-2})$. By Lemma 3, overall, this requires $O(m \log m \varepsilon^{-2} \cdot \frac{m}{\varepsilon} \cdot \min\{\varepsilon^{-2} \log \varepsilon^{-1}, M\})$ steps for solving the LP. Applying a technique of Beling and Megiddo [1], which transforms the solution for LP into a basic one, requires $O(m^{2.62} \log m \cdot \varepsilon^{-2})$ steps. Since we need to 'guess' d, the optimal number of bins, we solve the LP $O(\log n)$ times. (*iii*) Finally, we pack the items and divide each oversized bin into a set of ordinary bins. By Lemma 4, this can be done in $O(n + m \log m)$ steps. Hence, we get that the total running time of the scheme is

$$O(n \log m + \log n \cdot m^2 \log m \cdot \varepsilon^{-2} \cdot (m^{0.62} + \varepsilon^{-1} \cdot \min\{\varepsilon^{-2} \log \varepsilon^{-1}, M\})), \quad (1)$$

and since m, the number of distinct item sizes, is fixed, we get the statement of the lemma. ∎

3.1 Combining Shifting with Geometric Grouping

In the following we show that the linear grouping technique used for shifting the item sizes can be replaced by *geometric grouping* [14], while maintaining the bound on the total number of fragmented items. Recall that, in geometric grouping, the interval $(0, 1]$ is partitioned to sub-intervals of geometrically decreasing sizes, such that the smallest interval is $(0, \delta)$, for some $0 < \delta \leq \varepsilon$. This results in a set of sub-intervals $(0, \delta], (\delta, 2\delta), \ldots, (1/4, 1/2], (1/2, 1]$. Let $k = size(I) \cdot \varepsilon / \log_2 \varepsilon^{-1}$, where $size(I)$ is the total size of the items in the instance I. We partition the set of items whose sizes are in $(1/2, 1]$ to groups of k items. We then proceed to the sub-interval $(1/4, 1/2]$ and use groups of $2k$ items, and continue sequentially to the other sub-intervals; in each sub-interval, the number of items per group increases by factor of 2. Now, we apply shifting to the items in each sub-interval as follows. The sizes of the items in each group are rounded up to the smallest size of an item in the next size group in this sub-interval. This results in $H = O(\varepsilon^{-1} \log \varepsilon^{-1})$ size groups.

Lemma 8. *Applying shifting combined with geometric grouping, the items can be packed in the bins using at most C fragmentations.*

To obtain the overall running time of the scheme, we take in (1) $m = O(\varepsilon^{-1} \log \varepsilon^{-1})$, and use the next technical lemma (proof omitted).

Lemma 9. *For any $x, y, z \geq 1$ $O(n \log \varepsilon^{-1} + \log^x n \cdot \varepsilon^{-y} \log^z \varepsilon^{-1})$ can be bounded by $O(n \log \varepsilon^{-1} + \varepsilon^{-y} \log^{x+z} \varepsilon^{-1})$.*

Theorem 1. *There is an AFPTAS for BP-SPF which packs the items in* $(1+\varepsilon)OPT(I)+O(\varepsilon^{-1}\log\varepsilon^{-1})$ *bins, and whose running time is* $O(n\log\varepsilon^{-1}+\varepsilon^{-5}\log^4\varepsilon^{-1}\cdot\min\{\varepsilon^{-2}\log\varepsilon^{-1},M\})$*, where* M *is the longest binary representation of any input element.*

In the Appendix, we show that our scheme for BP-SPF can be modified to apply for BP-SIF.

4 Application to Bin Packing and Variable-Sized Bin Packing

As special cases, our scheme for BP-SPF can be applied to classical bin packing and variable-sized bin packing, improving the best known running times for these problems.

Theorem 2. *There is an AFPTAS for bin packing which packs the items in at most* $(1+\varepsilon)OPT(I)+O(\varepsilon^{-1}\log\varepsilon^{-1})$ *bins in time* $O(n\log\varepsilon^{-1}+\varepsilon^{-4}\log^3\varepsilon^{-1}\cdot\min\{\varepsilon^{-2},m^{0.62}M\})$*, where* M *is the longest binary representation of any input element.*

Proof. We can solve bin packing as a special case of BP-SPF in which $C=0$. We use the scheme as given in Section 2. Initially, we omit from the input items of size smaller than ε, and apply the scheme for the remaining items. The running time of the scheme is determined by the following steps. (i) Preprocessing the input. We apply as before geometric grouping, which can be done in $O(n\log m)$ steps. (ii) Solving approximately the linear program for bin packing. In applying the technique of Young [28], we can model the LP as a fractional covering problem with a simplex as a block $\{(x_C)|\sum x_C=r\}$ with guessed objective value r. Thus, using a result of [13] (see also [10]) the number of iterations, and the number of non-zero variables in the solution can be bounded by $O(m(\log m+\varepsilon^{-2}))$. In each call, the oracle needs to find a $(1+\varepsilon)$-approximate solution for the following instance of MCK. Given is a set of m items, such that each item a_i has a size $\varepsilon\le s(a_i)\le 1$ and profit $0<p(a_i)\le 1$. We may select from each item unbounded number of copies. We want to pack a set of item copies in a unit-sized bin such that the total profit is maximized. Since the total number of items in any feasible packing is at most $1/\varepsilon$, we can solve the problem as an instance of the knapsack problem, with $n=m/\varepsilon$ items. Using the scheme of [15], this can be done in $O((\frac{m}{\varepsilon}+\varepsilon^{-3})\log\varepsilon^{-1})$ steps. Alternatively, we can solve the IP for MCK in $O(mM)$ steps, where M is the longest binary representation of any input element. Thus, we solve the LP in $O(m(\log m+\varepsilon^{-2})\cdot\min\{(\frac{m}{\varepsilon}+\varepsilon^{-3})\log\varepsilon^{-1},mM\}))$

steps. Now, we use the technique of [1] to convert the solution for the LP to a basic solution. This is done in $O(m^{2.62}(\log m+\varepsilon^{-2}))$ steps. Since we need to 'guess' d, the optimal number of bins, we solve the LP $O(\log n)$ times. (iii) The items are packed in $O(n+m\log m)$ steps. Summarizing, we get that the running time of the scheme is $O(n\log m+\log n\cdot m(\log m+\varepsilon^{-2})\cdot\min\{\varepsilon^{-3}\log\varepsilon^{-1},m^{1.62}M\}))$. Since $m=O(\varepsilon^{-1}\log\varepsilon^{-1})$, we get the running time $O(n\log\varepsilon^{-1}+\log n\cdot\varepsilon^{-4}\log^2\varepsilon^{-1}\cdot\min\{\varepsilon^{-2},m^{0.62}M\})$. Applying Lemma 9, we get the statement of the theorem. Finally, the small items are added in linear time, using Next-Fit. ∎

We now show how our scheme for BP can be modified to solve *variable-sized* bin packing. Given an instance of variable-sized bin packing, with the set of bin sizes $B_1,\ldots,B_N$, such that $0<B_j\leq 1$, and there exists j such that $B_j=1$, we omit bins of sizes smaller than ε. also, we partition the items by their sizes: the *large* items have size at least ε^2; all other items are *small*. Initially, we find a packing of the large items. In the preprocessing step of the scheme, we use geometric grouping to reduce the number of distinct item sizes to $O(\varepsilon^{-2}\log\varepsilon^{-1})$. Let $\mathbf{x}$ denote the number of bins of each possible configuration (see in Section 2.2), and $\mathbf{f}$ is the vector of bin sizes. Then, having guessed correctly d, the total capacity of an optimal solution, we need to solve the linear program $\{LP':B\mathbf{x}\geq\mathbf{n},\ \mathbf{f}\cdot\mathbf{x}\leq d\}$, where $\mathbf{n}$ is the vector giving the number of items of each size. We can apply our scheme for BP with LP' replacing LP. The oracle needs to solve in each iteration an all_MCK instance, for all possible bin sizes in $[\varepsilon,1]$. To implement the oracle efficiently, we use the next lemma (proof omitted).

Lemma 10. *Given a feasible packing of a variable-sized bin packing instance in bins of possible sizes $B_j\in[\varepsilon,1]$, $1\leq j\leq N$, using the total bin capacity D, there exists a packing that uses $O(\log\varepsilon^{-1}/\varepsilon)$ distinct bin sizes, such that the total bin capacity used is at most $D(1+\varepsilon)$.*

Since each large item has the size at least ε^2, in solving MCK for each bin size, we can use the fast scheme of [15] for the knapsack problem with $n=m/\varepsilon^2$ items. As before, we can solve the problem exactly in $O(mM)$ steps. The oracle needs to solve this problem repeatedly, for $O(\varepsilon^{-1}\log\varepsilon^{-1})$ bin sizes. Then the solution for LP' can be converted to a basic one in $O(m^{2.62}(\log m+\varepsilon^{-2}))$ steps [1]. Since we need to guess the value of d, the total capacity used by an optimal solution, we solve LP' $O(\log n)$ times. Finally, we note that preprocessing the input, i.e., reducing the number of distinct item sizes and the number of distinct bin sizes, requires $O((N+n)\log m)$ steps. Also, the items are packed in $O(n+m)$ steps. To

get the total running time of the scheme, we summarize the above steps, taking $m = O(\varepsilon^{-2} \log \varepsilon^{-1})$, and apply Lemma 9.[4]

Theorem 3. *There is an AFPTAS for variable-sized bin packing which packs the items using a total bin capacity of at most* $(1+\varepsilon)OPT(I) + O(\varepsilon^{-2} \log \varepsilon^{-1})$, *where* $OPT(I)$ *is the optimal capacity, and whose running time is* $O((N+n) \log \varepsilon^{-1} + \varepsilon^{-8} \log^3 \varepsilon^{-1} \cdot \min\{\varepsilon^{-2} \log \varepsilon^{-1}, \varepsilon^{-0.24} M\})$, *where* M *is the longest binary representation of any input element.*

References

1. P. Beling and N. Megiddo, Using fast matrix multiplication to find basic solutions. *Theoretical Computer Science* 205 (1993) 307-316.
2. E.G. Coffman Jr., M.R. Garey, and D.S. Johnson. Approximation algorithms for bin packing: a survey. In D.S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems,* 46-93. PWS Publishing, Boston, MA, 1997.
3. F. Eisenbrand, Fast integer programming in fixed dimension. In *Proc. of ESA*, 2003.
4. L. Epstein and A. Levin. A Robust APTAS for the Classical Bin Packing Problem. In *Proc. of ICALP*, 2006.
5. L. Epstein and J. Sgall. Approximation schemes for scheduling on uniformly related and identical parallel machines. In *Proc. of the 7th European Symposium on Algorithms,* Vol. 1643 of *Lecture Notes in Computer Science,* 151-162. Springer-Verlag, 1999.
6. W. Fernandez de la Vega and G.S. Lueker. Bin packing can be solved within $1+\varepsilon$ in linear time. *Combinatorica,* 1:349-355, 1981.
7. A.V. Fishkin, K. Jansen, S. Sevastianov and R.Sitters, Preemptive Scheduling of Independent Jobs on Identical Parallel Machines Subject to Migration Delay. In *Proc. of ESA*, 2005.
8. D. K. Friesen and M. A. Langston. Variable sized bin packing. *SIAM J. on Computing*, 15:222–230, 1986.
9. M.R. Garey and D.S. Johnson, *Computers and Intractability: A guide to the theory of NP-completeness*, W. H. Freeman and Company, San Francisco, 1979.
10. M.D. Grigoriadis, L.G. Khachiyan, L. Porkolab and J. Villavicencio. Approximate Max-Min Resource Sharing for Structured Concave Optimization, *SIAM J. on Optimization*, 11(4):1081–1091, 2000.
11. M. Grötschel, L. Lovász and A. Schrijver, The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, I, 169–197, 1981.
12. D.S. Hochbaum and D.B. Shmoys. Using dual approximation algorithms for scheduling problems: Practical and theoretical results. *Journal of the ACM,* 34(1):144-162, 1987.

[4] We give the detailed scheme in the full version of the paper.

13. K. Jansen and L. Porkolab. On Preemptive Resource Constrained Scheduling: Polynomial-time Approximation Schemes. In proc. of IPCO, 2002, pp. 329–349.
14. N. Karmarkar and R.M. Karp. An efficient approximation scheme for the one dimensional bin packing problem. *Proc. 23rd IEEE Annual Symposium on Foundations of Computer Science*, 312-320, 1982.
15. H. Kellerer and U. Pferschy. A New Fully Polynomial Approximation Scheme for the Knapsack Problem. *J. of Combinatorial Optimization*, 3:59–71, 1999.
16. H. Kellerer, U. Pferschy, D, Pisinger, *Knapsack Problems*. Springer, 2004.
17. C.A. Mandal, P.P Chakrabarti, and S. Ghose. Complexity of fragmentable object bin packing and an application. *Computers and Mathematics with Applications*, vol.35, no.11, 91–97, 1998.
18. R. McNaughton. Scheduling with deadlines and loss functions. *Manage. Sci.*, 6:1–12, 1959.
19. N. Menakerman and R. Rom. Bin Packing Problems with Item Fragmentations. *Proc. of WADS*, 2001.
20. R. Motwani. Lecture notes on approximation algorithms. Technical report, Dept. of Computer Science, Stanford Univ., CA, 1992.
21. Multimedia Cable Network System Ltd., Data-Over-Cable Service Interface Specification, *http://www.cablelabs.com*, 2000.
22. F.D. Murgolo. An Efficient Approximation Scheme for Variable-Sized Bin Packing. *SIAM J. Comput.* 16(1), 149–161, 1987.
23. N. Naaman and R. Rom. Packet Scheduling with Fragmentation. *Proc. of INFOCOM'02*, 824-831, 2002.
24. S.A. Plotkin, D.B. Shmoys, Eva Tardos. Fast Approximation Algorithms for Fractional Packing and Covering Problems. In *Proc. of FOCS*, 1995.
25. H. Shachnai, T. Tamir and G.J. Woeginger. Minimizing Makespan and Preemption Costs on a System of Uniform Machines, *Algorithmica* 42, 309–334, 2005.
26. H. Shachnai, T. Tamir and O. Yehezkely, "Approximation Schemes for Packing with Item Fragmentation", to appear in *Theory of Computing Systems*.
27. V.V. Vazirani. Bin Packing. In *Approximation Algorithms*, 74-78, Springer, 2001.
28. N. E. Young. Sequential and Parallel Algorithms for Mixed Packing and Covering. In I*Proc. of FOCS, 538–546, 2001.*

A An AFPTAS for BP-SIF

In solving BP-SIF, we may assume that $\Delta \geq \varepsilon$. Indeed, when $\Delta < \varepsilon$, any packing which fragments the last item in each bin yields a $(1+\varepsilon)$-approximation to the optimal. We use the next result [26].

Lemma 11. *Any instance of BP-SIF has an optimal packing that is primitive.*

Given an instance of BP-SIF, we apply the steps of our scheme for BP-SPF, except that in step (iii) we solve the following linear program.

$$(LP') \qquad \sum_{i=1}^{q} A_{ji}x_i + \sum_{k=1}^{s} B'_{jk}y_k \geq n_j \qquad \text{for } j = 1,\ldots,m$$

$$\sum_{i=1}^{q} x_i + \sum_{k=1}^{s} f_k y_k \leq d$$

$$x_i \geq 0 \qquad \text{for } i = 1,\ldots,q$$

$$y_k \geq 0 \qquad \text{for } k = 1,\ldots,s$$

The matrix B' gives the configurations of oversized bins: we pack in each bin of size $2 \leq c \leq 1/\varepsilon$ items of total size at most $c - \Delta(c-1)$, to guarantee that the fragmented items can be packed along with their headers.[5] As before, we may assume that the capacity of each oversized bin is at most $\lceil 1/\varepsilon \rceil$. Hence, we have

Theorem 4. *There is an AFPTAS for BP-SIF, which packs the items in* $(1+\varepsilon)OPT(I)+O(\varepsilon^{-1}\log\varepsilon^{-1})$ *bins, and whose running time is* $O(n\log\varepsilon^{-1}+\varepsilon^{-5}\log^3\varepsilon^{-1}\cdot\min\{\varepsilon^{-2}\log\varepsilon^{-1}, M\})$, *where* M *is the longest binary representation of any input element.*

B Some Proofs

Proof of Lemma 4: We pack the items in the instance as follows. We first pack large items in linear time, using the configurations in the (rounded) integral solution. We note that some of the large items may remain unpacked, since we rounded *down* the x_i and y_k values. We now generate oversized bins from the fractions of the y_k values. This is done in time that is linear in m, since the basic solution contains at most $m+2$ non-zero variables. Now, we pack the remaining items greedily into each of the new bins. When packing items into a bin, if an item cannot be packed, we close the bin and start packing in a new bin. In the worst case, this may cause a waste of 1 in each of the bins. We complete packing items into these new bins in $O(m)$ steps.

Since we may have wasted some of the capacity of the new bins, we may have a set of large items, of total size at most $2m+4$ that are not packed yet. We can pack these items using First Fit, in $O(m\log m)$ steps. Finally, we pack the small items. This can be done in linear time, using Next Fit. ∎

[5] We describe the scheme in detail in the full version of the paper.

Proof of Lemma 6: The preprocessing step may increase the number of bins used by factor of ε. Since we may need to pack the subset of largest items in a new set of bins. Also, by Lemma 2, we increase the number of bins in the solution by factor of ε, overall using $(1+\varepsilon)^2 OPT(I)$ bins.

Assuming that we have guessed d correctly, the approximately feasible solution for LP may use at most $d(1+\varepsilon)$ bins. Also, as argued above, we may need to *trade* extra fragments for extra bins, thus increasing the total number of bins used at most by factor of ε. Hence, the solution for LP yields a $((1+\varepsilon)^2(1+2\varepsilon))$-approximation for the optimal number of bins.

Since we rounded down the basic solution (which has at most $(m+2)$ non-zero variables), after packing large items by the rounded solution, and adding extra oversized bins, the total volume of the unpacked large items cannot exceed $2m+4$. (We may waste a volume of $m+2$, since we use Next Fit for filling the extra bins, and additional volume of $m+2$, since we round down the capacities of the extra bins.) Finally, Packing a volume of $2m+4$ of items using the Next-Fit algorithm into unit-sized bins may require at most $2(2m+4)$ bins.

Finally, packing the small items, using the Next Fit algorithm, will not affect the quality of our approximation. This is obvious if we do not add any bin in the process. If we do add new bins, this means that each bin (except, maybe, for the last one) is full to at least $1-\varepsilon$ of its capacity. Since $\varepsilon < 1/2$ (see below), the total number of bins used in the solution is at most $(1+2\varepsilon)OPT(I)+1$.

Given an input parameter ε, by taking an approximation parameter $\varepsilon' = \varepsilon/8$ in the scheme (and thus, we may assume that $\varepsilon < 1/2$), we get that the instance is packed in $(1+\varepsilon)OPT(I) + O(m)$ bins. ∎

Proof of Lemma 8: Consider the instance I' resulting from shifting the item sizes with geometric grouping. Let I'' be the instance resulting from I by using geometric grouping and *rounding down* the size of each item to the smallest in its group. Then, omitting the group of largest items in each sub-interval, clearly, we can solve BP-SPF for the remaining items, by fragmenting the same number of items, without increasing the total number of bins used, i.e., $OPT(I'') \leq OPT(I)$.

Now, we pack separately the group of largest items in each sub-interval. For each sub-interval, this may require additional k bins, in which the items are packed with no fragmentation: the largest items in the sub-interval $(1/2, 1]$ are packed one in a bin, the items in the sub-interval $(1/4, 1/2]$ are packed two in a bin, and so on. Overall, we add to the packing $O(k \cdot \log_2 \varepsilon^{-1}) = size(I) \cdot \varepsilon \leq \varepsilon OPT(I)$ bins. ∎