Approximation Schemes for Packing with Item Fragmentation[*]

Hadas Shachnai [†] Tami Tamir[‡] Omer Yehezkely[§]

Abstract

We consider two variants of the classical bin packing problem in which items may be *fragmented*. This can potentially reduce the total number of bins needed for packing the instance. However, since fragmentation incurs overhead, we attempt to avoid it as much as possible. 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). In *bin packing with size preserving fragmentation (BP-SPF)*, there is a bound on the total number of fragmented items. These two variants of bin packing 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.

While both BP-SPF and BP-SIF do not belong to the class of problems that admit a *polynomial time approximation scheme (PTAS)*, we show in this paper that both problems admit a *dual* PTAS and an *asymptotic* PTAS. We also develop for each of the problems a dual asymptotic *fully polynomial time approximation scheme (AFPTAS)*. Our AFPTASs are based on a non-standard transformation of the mixed packing and covering linear program formulations of our problems into pure covering programs, which enables to efficiently solve these programs.

Keywords: polynomial time approximation schemes, bin packing, item fragmentation, linear programming, algorithms.

1 Introduction

In the classical *bin packing (BP)* 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. This problem is well known to be NP-hard. We consider a variant of BP known as *bin packing with item fragmentation (BPF)*, in which items can be fragmented (into two or more pieces). Therefore, it may be possible to pack

[*]A preliminary version of this paper appears in proceedings of the *3rd Workshop on Approximation and Online Algorithms (WAOA'05)*.

[†]Computer Science Department, The Technion, Haifa 32000, Israel. E-mail: `hadas@cs.technion.ac.il`.

[‡]School of Computer Science, The Interdisciplinary Center, Herzliya, Israel. E-mail: `tami@idc.ac.il`.

[§]Computer Science Department, The Technion, Haifa 32000, Israel. E-mail: `omery@cs.technion.ac.il`.

the items using fewer bins than in classical BP. However, since fragmentation incurs overhead, we attempt to avoid it as much as possible. We study two variants of BPF. In both variants, the goal is to pack all items in a minimum number of bins.

Size increasing fragmentation (BP-SIF): a header (or a footer) of a fixed size, $1 > \Delta > 0$, is attached to each (whole or fragmented) item. That is, the volume 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 volume $s(a_{i_j}) + \Delta$. Assume, for example, that $\Delta = 0.1$, and an instance consists of 3 items of sizes $\{0.4, 0.5, 0.7\}$. Without fragmentation, each item must be packed in a separate bin (occupying the volumes $0.5, 0.6$ and 0.8, respectively), while if, e.g., the item of size 0.4 is fragmented to 0.1 and 0.3, the resulting instance can be packed into two bins, the contents of which are $(0.1+0.1, 0.7+0.1)$, and $(0.3 + 0.1, 0.5 + 0.1)$.

Size preserving fragmentation (BP-SPF): an item a_i can split into two fragments: a_{i_1}, a_{i_2}, such that $s\left(a_i\right) = s\left(a_{i_1}\right) + s\left(a_{i_2}\right)$. 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* $C > 0$. Note that in the special case where $C = 0$ we get an instance of classic bin packing. (Most of our results can be applied to another variant of BP-SPF in which the goal is to minimize the packing cost, and the number of available bins, $b \geq \lceil \sum_i s(a_i) \rceil$ is given as part of the input.)

For any $\Delta > 0$ (in BP-SIF) and $C < \lceil \sum_i s(a_i) \rceil - 1$ (in BP-SPF), the BPF problem is NP-hard (see Sections 1.1 and 2 for hardness results). Therefore, we present approximation algorithms. The following applications motivate our study.

Community Antenna Television (CATV) Networks In many communication protocols, messages of arbitrary sizes are placed in fixed sized frames before they are transmitted. Consider, for example, the Data-Over-Cable Service Interface Specification (DOCSIS), defined by the Multimedia Cable Network System standard committee [17]. When using CATV network for communication, the upstream (data transmission from the subscribers' cable modem to the headend) is divided into numbered mini-slots. The DOCSIS specification allows two types of messages: *fixed location* and *free location*. Fixed location messages are placed in fixed mini-slots, while free location messages can be placed arbitrarily in the remaining mini-slots (each message may need one or more mini-slots). The specification also allows to fragment the free location messages. Since each of the original messages, as well as each of its pieces allotted to a mini-slot, has a header (or footer) attached to it, the problem of scheduling the free location messages yields an instance of the BP-SIF problem (see [15] for more details).

VLSI Circuit Design In high level synthesis of digital systems, when a logic unit is initialized, values of external variables are copied into the unit's internal variables. Each external variable may be copied into multiple internal variables. The logical unit has a fixed number, U, of memory ports that it can access in each work cycle. In order to copy an external variable into n variables, all $n+1$ variables need to be accessed. For example, if $U = 5$ and two external variables x, y need to be copied into 6 and 5 internal variables respectively, a possible initialization process is to copy x in the first cycle into 4 variables, then copy x into the 2 remaining variable and y into a single variable, and in a third cycle, copy y into the 4 remaining variables. Note that all 5 ports are used in each of these cycles. The goal is to complete the initialization process within the fewest possible work cycles. This yields an instance of BP-SIF, where U is the bin size, $\Delta = 1$, and the j-th item size is the number of internal variables that need to be assigned the value of the j-th external variable. (For more details see in [13].)

Preemptive Scheduling with Setup Times/Costs Consider the problem of preemptively scheduling a set of jobs on a minimal number of parallel machines; the i-th job has the length ℓ_i, and all jobs should be completed by time D which is the deadline for all jobs. In the case where starting/resuming a job incurs *setup time*, each preemption (split) causes additional setup time to a new job-segment, therefore the resulting problem is BP-SIF. In the case where preempting/resuming a job incurs a *setup cost*, the resulting problem is BP-SPF, where each preemption (split) causes an additional cost, and the goal is to find a schedule whose total cost (given by the total number of preemptions) does not exceed a given bound C.

Flexible Packaging In some packaging problems, the cost of the packages is substantive. This includes (i) storage management, where files need to be stored in a minimal number of disks, and each file or file-segment has a header of fixed size; (ii) transportation problems, where material is to be delivered using minimal number of vehicles. For example, trucks need to ship construction materials, such as sand, and the sand is given in several sizes of bags. The number of trucks used for the shipment may be reduced, by splitting the content of some bags.

1.1 Related Work

It is well known (see, e.g., [16]) that BP does not belong to the class of NP-hard problems that admit a PTAS. In fact, BP cannot be approximated within factor $\frac{3}{2} - \varepsilon$, for any $\varepsilon > 0$, unless P=NP [7]. However, there exists an *asymptotic* PTAS *(APTAS)* which uses, for any instance I, $(1+\varepsilon)\,OPT(I) + k$ bins for some fixed k. Fernandez de la Vega and Lueker [5] presented an APTAS with $k = 1$, and Karmarkar and Karp [10] presented an *asymptotic fully* PTAS *(AFPTAS)* with $k = 1/\varepsilon^2$. Alternatively, a *dual* PTAS, which uses $OPT(I)$ bins of size $(1+\varepsilon)$ was given by Hochbaum and Shmoys [9]. Such a dual PTAS can also be derived from the work of Epstein and Sgall [6] on multiprocessor scheduling, since BP is dual to the *minimum makespan* problem. (Comprehensive surveys on the bin packing problem appear, e.g., in [3, 24].)

Mandal et al. introduced in [13] the BP-SIF problem and showed that it is NP-hard. Menakerman and Rom [15] and Naaman and Rom [18] were the first to develop algorithms for bin packing with item fragmentation, however, the problems studied in [15] and [18] are different from our problems. For a version of BP-SPF in which the number of bins, N, is given, and the objective is to minimize the total cost incurred by fragmentation, the paper [15] 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.

There has been some related work in the area of preemptive scheduling on parallel machines. The paper [20] 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. However, the techniques used in this paper rely strongly on the assumption that the job-wise/total bounds on the number of preemptions are some fixed constants, while in solving our BPF variants the number of splits may depend on the input size. For other related work on preemptive scheduling with preemption costs see, e.g., in [1, 21]. (Comprehensive surveys of known results on preemptive scheduling are given, e.g., in [2, 12].)

1.2 Our Results

In this paper, we develop approximation schemes for the two variants of bin packing with item fragmentation. In Section 2 we give some hardness results and then analyze the performance of a class of natural algorithms for our problems. In Section 3 we develop a dual PTAS and APTAS for BP-SPF. The dual PTAS packs all the items in $OPT(I)$ bins of size $(1+\varepsilon)$, and the APTAS uses at most $(1+\varepsilon)OPT(I)+1$ bins. In Section 5 we show that these schemes can be modified to apply for BP-SIF. Finally, we show that each of the problems admits a dual AFPTAS. Our AFPTASs pack the items into $OPT(I)+k$ bins of size $(1+\varepsilon)$, where $k \leq 1/\varepsilon^2+3$, in time that is polynomial in n and $1/\varepsilon$. For BP-SIF and BP-SPF, the existence of a polynomial time algorithm with constant additive error is open. For the special case of BP-SPF where $C = 0$, i.e., classic bin packing, this is a long standing open problem; see, e.g., in [3].

Technical Contributions The paper contains two technical contributions. (i) Our APTAS for BP-SPF is based on a novel *oblivious* version of the shifting technique (see, e.g., [24]). Given an instance of n items whose sizes are *unknown*, we define a set of items whose (shifted) sizes are given as *variables*; the values of these variables are then revealed by solving a linear programming relaxation of the packing problem. We expect that this non-standard use of the shifting technique will find more applications. (ii) A crucial step in the AFPTAS that we develop for each of our problems is a transformation of the corresponding mixed covering and packing linear program into a pure covering program. This enables to efficiently solve the original program. Our non-standard transformation (see in Section 4) modifies each *packing* constraint to a *covering* constraint, by reversing the inequality. With some simple changes in the objective function, and possibly adding some new covering constraints, we obtain a pure covering program. Such transformation may be useful for a certain class of mixed packing and covering programs, in which the number of packing constraints is small. The advantage over a standard transformation (see, e.g., in [4]) is that we do not add negative entries in the coefficient matrix.

2 Preliminaries

In this section we give some basic lemmas and properties of packing with item fragmentation. We also present a class of natural algorithms and analyze their performance. We first show that for both variants of BPF, there exists an optimal packing of certain structure. This allows us to reduce the search for a good packing to this subset of packings.

Define the *bin packing graph (BP 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 BP graph is a collection of paths. Note that the last condition implies that in any connected component of the BP graph there are two bins including only a single fragment.

Lemma 2.1 *Any instance of BP-SPF has an optimal packing that is primitive.*

Proof: We show that any feasible BP-SPF packing, in particular an optimal one, can be converted into a primitive packing with the same number of splits or fewer. Given a packing with f splits, consider its BP graph. Let $V(C_i), I(C_i)$ denote, respectively, the set of vertices and the set of packed items in a connected component C_i. The connected component C_i has at least $|V(C_i)| - 1$ edges. Note that for $i \neq j$, $I(C_i) \cap I(C_j) = \emptyset$. Thus, for each connected component C_i, the items in $I(C_i)$ can be repacked into the respective bins of $V(C_i)$ by filling the bins one at a time, using an arbitrary order of $I(C_i)$, and splitting (if necessary) the last item packed into the active bin. This results in a primitive packing with at most f splits, since every connected component C_i is replaced by a subgraph with at most $|V(C_i)| - 1$ edges. The resulting component in the BP graph is a path since each of the first and last bins includes a single fragment. ∎

Lemma 2.2 *Any instance of BP-SIF has an optimal packing that is primitive.*

Proof: Similar to the proof of Lemma 2.1, we show that any feasible BP-SIF packing, in particular an optimal one, can be converted into a primitive packing using at most the same number of bins and at most the same number of splits. A given packing is converted into a primitive one separately for each connected component of the corresponding BP graph. Consider a connected component C_i. The total volume allocated to headers in $V(C_i)$ is at least $\Delta \cdot (|I(C_i)| +$

$|V(C_i)| - 1)$. Repack the items in $I(C_i)$ into the respective bins of $V(C_i)$, by filling the bins one after the other (using an arbitrary order of $I(C_i)$) and fragmenting the last item packed into the bin, if necessary, and if possible $-$ it is not possible to fragment the current item if the current bin has less than Δ space left. Clearly, the resulting packing is primitive. We need to show that $|V(C_i)|$ bins are enough. Let b_1 be the number of bins in which the last item splits. These bins are filled to their capacity. By the above transformation, each of the other bins is either the last one or full to capacity at least $1 - \Delta$. The number of splits is therefore b_1, meaning that the total volume required for headers is $\Delta \cdot (|I(C_i)| + b_1)$. We know that the total size of the bins is larger by at least $\Delta(|I(C_i)| + |V(C_i)| - 1)$ than the total item sizes. Therefore, even if $|V(C_i)| - b_1 - 1$ intermediate bins are full only to capacity $1 - \Delta$, there is enough space in the bins. ∎

Hardness of BPF: Clearly, by a simple reduction from Partition, it is NP-hard to decide whether an instance of BPF can be packed in two bins with no splits. This implies that BP-SIF is NP-hard for any $\Delta > 0$. For BP-SPF, by McNaughton's rule [14], if the bound on the number of splits is $C \geq \lceil \sum_i s(a_i) \rceil - 1$, a packing that uses $\lceil \sum_i s(a_i) \rceil$ bins exists and can be found in linear time. We prove that it is NP-hard to avoid using a large number of splits, even if the existence of a packing that uses no splits is known a-priori. Formally,

Theorem 2.3 *For any $b > 1$, if a BP-SPF instance can be packed into b bins with no splits, then it is NP-hard to find a packing into b bins for any budget $C < \lceil b/2 \rceil$.*

Proof: Assume first that b is even. The proof is based on defining an instance for which an optimal packing into b bins requires no splits, while it is NP-hard to fill a bin to full capacity with no fragments. Thus, any efficient algorithm ends up with at least one fragment in any bin, implying at least $b/2$ splits.

We first prove hardness for bins with different sizes and then extend the proof to identical bins. The reduction is from the *Partition* problem. Given $a_1, ..., a_n$, an instance for Partition with total size of items equals to $2S$, construct an instance for BP-SPF with $k = b/2$ bins and k sets of items, $I_0, \ldots, I_{k-1}$. Let $M > (2S + 1)$ be an integer. The set I_0 consists of items of sizes $a_1, a_2, ..., a_n$; I_1 consists of items of sizes $a_1M, a_2M, ..., a_nM$, and in general, I_j consists of items of sizes $a_1M^j, a_2M^j, ..., a_nM^j$. The bin sizes are $S, SM, SM^2, ..., SM^{k-1}$ $-$ two bins in each size. If there exists a partition of the items into two sets of size S, then a packing with no splits exists, by packing I_j into the two bins of size SM^j. Consider a packing of the items into the bins.

Claim 2.4 *Any full bin with no splits induces a partition.*

Proof: Assume that for some z, a bin of size SM^z is full. No item from a set $I_j, j > z$ is packed in the bin, since each item from $I_j, j > z$ is larger than SM^z (because $a_iM^{z+1} > SM^z$). Also, no item from a set $I_j, j < z$ is packed. This is true since the total size of items from earlier sets is $2S(1 + M + M^2 + ...M^{z-1}) = 2S(M^z - 1)/(M - 1)$ which is less than M^z for all $M > 2S + 1$, therefore, no combination of items from the sets $I_j, j < z$, can be used.

It follows that the full SM^z bin contains only items from one set, and by scaling by M^z, its content induces a partition of the original instance. ∎

In order to extend the proof to instances with identical bins, we add to the set I_j two *filler items* of size $S(M^k - M^j)$, and the $2k$ bins have size SM^k. Note that the two smallest filler items have total size $2S(M^k - M^{k-1})$, which is larger than SM^k for any $M > 2$. Therefore, there is at most one filler item in any bin. Also, the total size of non-filler items is too small to fill a bin, therefore a full bin must contain exactly one filler item. Assume that some bin is filled with items, and no fragments. Let $S(M^k - M^z)$ be the size of the filler item in the bin, the rest of the bin is filled by items of total size SM^z, and the proof for the different size bins can be applied here. It means that any efficient algorithm ends up splitting one item in each set, resulting in at least one fragment in each bin, and therefore at least $b/2$ splits.

If b is odd, then by adding a single item of size SM^k, and fixing $k = \lfloor b/2 \rfloor$, we get an instance that can be packed into b bins with no splits, and the rest of the reduction is identical to the case where b is even. Finally, by scaling all sizes by a factor of $1/SM^k$ we get an equivalent problem for unit-size bins. ∎

Discrete Instances: An instance of BP-SPF is *discrete* if for some fixed positive integer U all item sizes are taken from the sequence $\{\delta, 2\delta, ..., U\delta\}$, where $\delta = 1/U$. Note that since U is integral, there exists an optimal (primitive) solution in which any fragmented item splits into two fragments having sizes in $\{\delta, 2\delta, ..., (U-1)\delta\}$, thus, no new sizes are introduced in the fragmentation process. For an instance of BP-SIF to be discrete, it is also required that Δ is of the form $i\delta$ for some integer i.

Given a discrete instance I, for BP-SIF or BP-SPF, define a *bin configuration* to be a vector of length U, in which the j-th entry is the number of items of size $j\delta$ packed in the bin; the configuration is valid if the total size of items (together with their headers - in BP-SIF) is at most 1. The *configuration matrix*, A_I, is the matrix which gives all the possible bin configurations. Each column in A_I gives a bin configuration. The *fragmentation matrix*, B_I, is the matrix which gives all possible fragmentations of items in I. Each column in B_I corresponds to a single possible split, and represents the change in the total number of items in the instance if this split is performed. Each column is a *fragmentation vector* in which all entries are 0 except for a single $(+1)$ entry and a single (-2), or two (-1) entries, such that the total size of the negative entries is equal to the size of the positive entry. For example, if $U = 6$ then the column $[0, -1, -1, 0, 1, 0]$ corresponds to a single split of an item of size $5/6$ into two items of sizes $2/6$ and $3/6$, and the column $[0, -2, 0, 1, 0, 0]$ corresponds to a single split of an item of size $4/6$ into two items of size $2/6$.

2.1 Bounds for Simple Offline and Online Algorithms

Consider the following class of algorithms (defined in [15]). An Algorithm is said to *avoid unnecessary fragmentation* if it follows the two rules:

1. No unnecessary fragmentation: An item is fragmented only if packed in a bin that does not have enough space for it. Upon fragmentation of an item, the first fragment must fill one of the bins. The second fragment is packed by the packing rule of the algorithm.

2. No unnecessary bins: A new bin is opened only if even a fragment of the currently packed item cannot fit into any of the open bins.

In particular, an algorithm that fills the bins to full capacity one after the other (in BP-SIF, the algorithm moves to the next bin when the currently open bin is filled to capacity at least $(1-\Delta)$) avoids unnecessary fragmentation. Note that this greedy algorithm does not assume any order on the items and is therefore an online algorithm.

The following theorems give upper bounds on the performance of any algorithm that avoids unnecessary fragmentation.

Denote by N_{opt}, N_{alg} the number of bins used by an optimal solution and a given algorithm respectively.

Theorem 2.5 *Any algorithm for BP-SIF that avoids unnecessary fragmentation uses at most* $N_{opt}(1+\frac{\Delta}{1-\Delta})+1$ *bins.*

Proof: In any packing achieved by an algorithm that avoids fragmentation, each bin, except maybe for the last one, is either filled to full capacity - with the last item fragmented, or filled up to capacity at least $1-\Delta$. Denote by S^+ the total size of all items and their headers if no fragmentation is used. That is $S^+ = \sum_i (s(a_i)+\Delta)$. Let N_1, N_2 denote the number of full bins and bins filled to capacity $1-\Delta$ in the solution of the algorithm. Note that the number of headers added due to fragments is at most N_1, that is, $S^+ \geq N_1 - N_1\Delta + N_2(1-\Delta) = (N_1+N_2)(1-\Delta)$. On the other hand, $\lceil S^+ \rceil$ is a lower bound on the optimal number of bins required to pack the whole instance. Therefore, $N_{alg} \leq N_1 + N_2 + 1 \leq N_{opt}/(1-\Delta)+1 = N_{opt}(1+\frac{\Delta}{1-\Delta})+1$. ■

Theorem 2.6 *Any algorithm for BP-SPF that avoids unnecessary fragmentation uses* N_{opt} *bins for any budget* $C \geq \lceil \sum_i s(a_i) \rceil - 1$, *and at most* $N_{opt}+Z$ *bins for budget* $C = \lceil \sum_i s(a_i) \rceil - Z$, *where* $Z > 1$.

Proof: In any packing output by an algorithm that avoids unnecessary fragmentation, the first C bins are full, and all other bins (except maybe for one), in which the total packed item size is $\lceil \sum_i s(a_i) \rceil - C = Z$, are at least half full. Therefore the maximal number of bins used is at most $C + 2Z = \lceil \sum_i s(a_i) \rceil + Z \leq N_{opt} + Z$. If, after filling to full capacity the first C bins, the remaining volume of unpacked items is less than 1, that is, $C \geq \lceil \sum_i s(a_i) \rceil - 1$, then the $(C+1)^{th}$ bin can accommodate the remaining items with no additional splits. ■

3 Bin Packing with Size-Preserving Fragmentation (BP-SPF)

Recall that in BP-SPF we are given a list of n items $I = (a_1, a_2, ..., a_n)$, each with the size $s(a_i) \in (0, 1]$. The number of splits is bounded by C. The goal is to pack all items using minimal number of bins and at most C splits. In this section we develop a dual PTAS and an APTAS for BP-SPF.

3.1 A Dual PTAS

Given an input I and some $\varepsilon > 0$, our dual PTAS for BP-SPF packs the items in an optimal number of bins of size at most $(1 + \varepsilon)$. The scheme proceeds in the following steps. (i) Partition the items into two groups by their sizes: the *large* items have size at least ε; all other items are *small.* (ii) Round up the size of each large item to the nearest integral multiple of ε^2. (iii) Guess $OPT(I)$, the number of bins used by an optimal packing of I. (iv) Pack optimally the large items with fragmentation, using at most C splits, into $OPT(I)$ bins of size $(1 + \varepsilon + \varepsilon^2)$. (v) Pack the small items in an arbitrary order, one at a time. Each item is packed into the bin having maximum free space.

For analyzing the scheme, we need the next two lemmas.

Lemma 3.1 *It is possible to pack the rounded large items into $OPT(I)$ bins of size $(1 + \varepsilon + \varepsilon^2)$.*

Proof: By Lemma 2.1, there is an optimal primitive packing of I. In such a packing, there are at most $1/\varepsilon$ large items in each bin (since the size of a large item is at least ε). If a bin contains less than $1/\varepsilon$ non-fragmented large items, it may also contain at most two fragments of large items. Thus, each bin may contain at most $1/\varepsilon - 1 + 2$ distinct large items/fragments. Each item or fragment may be rounded up, and thus the total extension incurred by packing large items into each bin is at most $\varepsilon^2(1 + 1/\varepsilon) = (\varepsilon + \varepsilon^2)$. ∎

Lemma 3.2 *The small items can be added to the $OPT(I)$ bins of size $(1+\varepsilon+\varepsilon^2)$ without causing additional overflow.*

Proof: The small items are added greedily with no fragmentation one at a time, into the bin having maximum free space. The size of any small item is at most ε. As guessed in step (i), all the items – small and large – can be optimally packed in $OPT(I)$ bins of size 1. The optimal packing of the rounded large items uses a minimal number of fragments, thus the total size of items and packed in step (iv) is at most their size in an optimal packing. This implies that when a small item is packed, there exists at least one non-full bin (otherwise, the total capacity of the bins is smaller than the total actual sizes of items in an optimal packing). When a small item is added with no fragmentation into a non-full bin, the bin capacity is never extended beyond $1 + \varepsilon$. ∎

Theorem 3.3 *BP-SPF admits a dual PTAS.*

Proof: Given an input parameter $\varepsilon' > 0$, then by using Lemmas 3.1 and 3.2, and taking $\varepsilon = \varepsilon'/2$, we get that the scheme packs all the items in at most $OPT(I)$ bins, each of size at most $1+\varepsilon'$. We turn to analyze the time complexity of the scheme. Steps (i) and (ii) are linear and are done once. For Step (iii), note that $\lceil s(I) \rceil \leq OPT(I) \leq n$, therefore $OPT(I)$ can be guessed in $O(\log n)$ iterations; each iteration involves packing the rounded large items and adding the small items. When packing the large items in step (iv), since there are at most $1/\varepsilon$ large items in a bin, and the number of distinct item sizes is $m \leq 1/\varepsilon^2$, we need to consider $O(n^{m^{1/\varepsilon}}) = O(n^{(1/\varepsilon^2)^{1/\varepsilon}})$ packings. Finally, the small items are added in $O(n)$ steps. ■

3.2 An APTAS for BP-SPF

We describe below an asymptotic PTAS for BP-SPF. Given an $\varepsilon > 0$, our scheme packs any instance I into at most $OPT(I)(1+\varepsilon)+1$ bins. Our scheme applies shifting to the item sizes, and *oblivious* shifting to the (unknown) fragment sizes in some optimal solution. The latter is crucial for efficiently finding a primitive packing that is close to the optimal.

The following is an overview of the scheme. (i) Guess $OPT(I)$ and $c \leq C$, the number of fragmented items. (ii) Partition the instance into *large* and *small items*; any item with size at least ε is large. (iii) Transform the instance to an instance where the number of distinct item sizes is fixed. (iv) Guess the configuration of the i-th bin in some optimal packing (defined below). (v) Let $R \geq 1$ be the number of distinct fragment sizes in a shifted optimal solution, where $R \leq 1/\varepsilon^2$ is a constant. Guess the number of fragmented items of the j-th group, having fragments of types $1 \leq r_1, r_2 \leq R$. (vi) Solve a linear program which gives the fragment sizes for each fragmented item. (vii) Pack the non-fragmented items and the fragments output by the LP, using the bin configurations. (iix) Pack the small items in an arbitrary order, one at a time, into the bin having maximum free space.

Transformation of the Input and Guessing Bin Configurations: First, guess the values $OPT(I)$ and c, the number of fragmented items. Since there exists an optimal primitive packing, $1 \leq c \leq \min(C, OPT(I)-1)$. Next, transform the instance I to an instance I' in which there are at most $1/\varepsilon^2$ item sizes. This can be done by using the shifting technique (see, e.g., in [24]). 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 including $H = \lceil n\varepsilon^2 \rceil$ 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 $m = n/H \leq 1/\varepsilon^2$. We renumber the resulting sets of items in non-decreasing order by the (shifted) sizes.

Define an *extended bin configuration* to be a vector which gives the number of items of each size-set packed in the bins, as well as at most two fragments which may be added (since we find a primitive packing). Each extended bin configuration consists of three parts: (i) a vector $(h_1, \ldots, h_m)$ where h_j, $1 \leq j \leq m$, is the number of non-fragmented items of the j-th size-set packed in the bin, (ii) a binary indicator vector of length m, with at most two '1' entries – in the indices of at most two size groups $1 \leq j_1, j_2 \leq m$ that contribute a fragment to the bin, and (iii) a binary indicator vector of length R (to be defined), with at most two '1' entries – in the indices corresponding to at most two types of fragments $1 \leq r_1, r_2 \leq R$ which are packed in the bin.

Now, since both the number of size groups and the number of fragment types are fixed constants, the number of bins of each configuration can be guessed in polynomial time .

Guessing the Fragment Types: The heart of the scheme is in finding how each of the c guessed items is fragmented. This is done by using the following oblivious shifting procedure. Suppose that in some optimal packing the (unknown) fragment sizes are $y_1 \leq y_2 \cdots \leq y_{2c}$, then apply shifting to this sorted list, by partitioning the entries into subsets of sizes at most $Q = \lceil 2c\varepsilon^2 \rceil$, and round up the size of each fragment to the largest entry in its subset. Now, there are $R \leq 1/\varepsilon^2$ distinct fragment sizes. These sizes are determined later, by solving a linear program for packing the instance with fragmentation.

Next, find the type of fragments generated for each of the c guessed items. Note that the number of pairs of fragment sizes are R^2. Thus, guess for each size group j the number of items in this group having a certain pair of fragment types. This can be done in polynomial time.

Solving the LP and Packing the Items: Having guessed the fragment types for each fragmented item, use a linear program to obtain the fragment sizes that yield a feasible packing. Let x_r denote the size of the r-th fragment in a shifted optimal sorted list. For any item in group j, that is fragmented to the pair of type (r, s), we verify that the sum of fragment sizes is at least s_j, the size of an item in group j. Denote by ℓ^r_{ij} the indicator for packing a fragment of type r contributed by size group j in bin i, $1 \leq r \leq R$, $1 \leq j \leq m$, $1 \leq i \leq OPT(I)$. The goal is to find R fragment sizes, $x_1 \leq \cdots \leq x_R$, which enable to pack all the items. That is, once a correct guess of the fragment types is made, the total volume packed by the LP is the volume of the c fragmented items. Let $N = OPT(I)$. Denote the set of fragment pairs assigned to the items of size group j F_j, $1 \leq j \leq m$. Finally, Γ_i is the space left in bin i after packing the non-fragmented items. We solve the following linear program.

$$
\begin{array}{lll}
(LP) & \text{maximize} & \displaystyle\sum_{i=1}^{N}\sum_{j=1}^{m}\sum_{r=1}^{R} x_r \ell^r_{ij} \\
 & \textit{subject to}: & x_{r_1} + x_{r_2} \geq s_j \quad \forall j,\ (r_1, r_2) \in F_j \\
 & & \displaystyle\sum_{j=1}^{m}\sum_{r=1}^{R-1} x_r \ell^r_{ij} \leq \Gamma_i \text{ for } i = 1, \ldots, N \\
 & & x_r \geq 0 \text{ for } r = 1, \ldots, R \\
 & & x_R = 1
\end{array}
\tag{1}
$$

Inequality (1) ensures that the fragment types $1, \ldots, R-1$ can be packed in the $OPT(I)$ bins, given the correct guess.

Given the solution for the LP, the fragment sizes for each of the c fragmented items are known. Pack the non-fragmented items as given in the bin configuration and add the fragments of sizes $1, \ldots, R-1$ in these $OPT(I)$ bins. Next, add $H+Q \leq 2\varepsilon OPT(I)$ new bins. In each of the H new bins pack separately a non-fragmented item in the largest size group generated during shifting. In the Q new bins, pack the fragments of type R (i.e., the largest fragments) greedily. Finally, add greedily the small items.

Lemma 3.4 *For any ε, $0 < \varepsilon \leq 1/2$, adding the small items may add at most one extra bin.*

Proof: Denote by N' the total number of bins used, after packing the small items. Since we add extra bins only if necessary, if we added bins then at least $N'-1$ bins are at least $1-\varepsilon$ full. Therefore, $(N'-1)(1-\varepsilon) \leq OPT(I)$. Thus, $N' \leq OPT(I)/(1-\varepsilon)+1$, and since $\varepsilon \leq 1/2$, $N' \leq OPT(I)/(1-\varepsilon)+1 \leq (1+2\varepsilon)OPT(I)+1$. ∎

Given an input parameter $\varepsilon > 0$, by taking in the scheme as an input parameter $\varepsilon' = \varepsilon/2$, we will use at most $OPT(I)(1+\varepsilon)+1$ bins.

Theorem 3.5 *There is an asymptotic PTAS for BP-SPF.*

4 A Dual AFPTAS for BP-SPF

We now describe an asymptotic dual FPTAS for BP-SPF, which packs the items of a given BP-SPF instance into $OPT(I)+k$ bins of size $(1+\varepsilon)$, where $k \leq 4/\varepsilon^2+3$ is some constant. Our scheme applies some of the steps used in the dual PTAS given in Section 3.1; however, since *guessing* the fragmented items results in number of iterations that is exponential in $1/\varepsilon$, we use instead a linear programming formulation of the packing problem, whose solution yields a feasible packing of the instance.

The scheme proceeds in the following steps. Steps (*i*)-(*iii*) are the same as in the dual PTAS described in Section 3.1. (*iv*) Guess $c \leq C$, the number of fragmented items, and $d = OPT(I)$, the number of bins used in an optimal packing. (*v*) Define for the large items the *configuration matrix*, A, and the *fragmentation matrix*, B, each having $1/\varepsilon^2$ rows. (*vi*) Solve within an additive factor of 1 a linear programming formulation of the problem for packing the large items. (*vii*) Round the solution of the linear program and pack accordingly the large items in at most $OPT(I)+k$ bins of size $1+\varepsilon+\varepsilon^2$, where $k \leq 1/\varepsilon^2+3$. (*viii*) Add the small items in arbitrary order to the bins (without overpacking).

We describe below how our scheme finds a good packing of the large items.

Constructing the Configuration and Fragmentation Matrices: Recall that, for the rounded 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 $q \leq (1/\varepsilon)^{1/\varepsilon^2}$. The fragmentation matrix, B, consists of all possible fragmentation vectors for the given set of large items; the k-th vector is the k-th column in B, and the number of columns is $p \leq 1/\varepsilon^4$.

Solving the Linear Program: We now formulate the problem of packing the rounded large items in a minimum number of bins as a linear program. Let n_j denote the number of items in the j-th size group. Denote by x_i the number of bins having the i-th configuration, $1 \leq i \leq q$. Let z_k, $1 \leq k \leq p$ denote the number of items that are split according to the k-th fragmentation vector. A natural linear programming relaxation of our problem is the following.

$$\begin{array}{lll}
(P_1) & \text{minimize} & \sum_{i=1}^{q} x_i \\
& \textit{subject to}: & \sum_{i=1}^{q} A_{ij} x_i + \sum_{k=1}^{p} z_k B_{kj} \geq n_j \quad \text{for } j = 1, \ldots, m \qquad (2) \\
& & \sum_{k=1}^{p} z_k \leq c \qquad (3) \\
& & x_i \geq 0 \text{ for } i = 1, \ldots, q \\
& & z_k \geq 0 \text{ for } k = 1, \ldots, p
\end{array}$$

The constraints (2) reflect the coverage requirement for the n_j items of size group j; the constraint (3) guarantees that at most c items split.

Note that the above is a mixed covering and packing program, in which some of the coefficients may be negative. We now show that by modifying the objective function and by adding a constraint on the total number of bins used, (P_1) can be transformed to a pure covering program, for which a basic feasible solution can be obtained in time that is polynomial in n and $1/\varepsilon$. Consider the following program.

$$\begin{array}{lll}
(P) & \text{minimize} & \sum_{i=1}^{q} x_i + \sum_{k=1}^{p} z_k \\
& \textit{subject to}: & \sum_{i=1}^{q} A_{ji} x_i + \sum_{k=1}^{p} B_{jk} z_k \geq n_j \quad \text{for } j = 1, \ldots, m \\
& & \sum_{i=1}^{q} x_i \geq d \qquad (4)
\end{array}$$

$$\sum_{k=1}^{p} z_k \geq c \tag{5}$$

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

$$z_k \geq 0 \text{ for } k = 1, \ldots, p$$

In the program (P), we minimize the total number of bins plus the number of splits, and require that the solution uses at least d bins and c splits. Indeed, having guessed correctly d and c, the solution will use exactly d bins and c splits.

In the dual of the program (P) there is a variable y_j for each constraint.

$$(D) \qquad \text{maximize} \qquad \sum_{j=1}^{m} n_j y_j + d y_{m+1} + c y_{m+2}$$

$$\textit{subject to}: \qquad \sum_{j=1}^{m} A_{ji} y_j + y_{m+1} \leq 1 \text{ for } i = 1, \ldots, q \tag{6}$$

$$\sum_{j=1}^{m} B_{jk} y_j + y_{m+2} \leq 1 \text{ for } k = 1, \ldots, p \tag{7}$$

$$y_j \geq 0 \text{ for } j = 1, \ldots, m+2$$

The dual program (D) is a fractional packing linear program, in which some coefficients may be negative. Note that the number of constraints in (D) is exponential in $1/\varepsilon$; however, it is possible to solve (D) using the modified ellipsoid method, as described in [10]; The differences between (D) and the dual of the classical bin packing LP are: (i) The addition of y_{m+1} and y_{m+2} to the target function. (ii) The addition of y_{m+1} to constraints of type 6 (see in [10]). (iii) The addition of type 7 constraints (There are less than $m^2 = \varepsilon^{-4}$ restrictions of this type). All These additions can be handled with ease by simple modifications to the "separating hyperplane oracle" provided by [10]. This will result in a reduction of the number of constraints in (D) to $O(m^2 log(\varepsilon^{-1} n))$. Then, solving the dual of this reduced LP yields a solution that is within an additive of 1 from the optimal for (P). Next, transform the solution of (P) into a basic solution, in which at most $m + 2$ variables have positive values. The scheme uses this basic solution for packing the *large* items.

Packing the Items: For packing the large items, round down the x_i and the z_k values in the (fractional) basic solution for (P). Consequently, some of the items cannot be packed/fragmented. We add new bins, in which these remaining items are packed according to the configurations corresponding to the rounded x_i values; for rounded z_k values, pack the item in the corresponding fragmentation vector in a separate bin. Overall, at most $m + 2$ additional bins may be used.

Finally, the small items are added in an arbitrary order with no fragmentation; each of the small items can be added to any bin with remaining capacity larger than ε.

4.1 Analysis of the Scheme

In the following, we show that our scheme packs all the items in at most $OPT(I) + k$ bins of size $(1+\varepsilon+\varepsilon^2)$, where $k \leq 1/\varepsilon^2 + 3$ is some constant. We distinguish between the large and the small items.

Lemma 4.1 *For some $k \leq 1/\varepsilon^2 + 3$, the scheme packs the large items in at most $OPT(I) + k$ bins of size $1 + \varepsilon + \varepsilon^2$.*

Proof: Given a (fractional) solution for (P) that is within an additive factor of 1 to the optimal, by Lemma 3.1, after rounding down the x_i values, it is possible to pack the large items in at most $OPT(I) + 1$ bins of size $1 + \varepsilon + \varepsilon^2$. Also, in the *basic* solution for (P), at most $m + 2 \leq 1/\varepsilon^2 + 2$ variables get strictly positive values; therefore, while packing the remaining items, at most this number of new bins may be added. ■

Lemma 4.2 *Adding the small items requires no additional bins.*

Proof: Assuming that $OPT(I)$ is correctly guessed, the total size of the items is at most $OPT(I)$. In addition, since each small item has size smaller than ε, if the scheme needs to add bins, it already packed items of total size at least $OPT(I) + k > OPT(I)$. A contradiction. ■

Theorem 4.3 *For any $\varepsilon \in (0, 1)$, there is a dual AFPTAS for BP-SPF which packs the items in at most $OPT(I) + 4/\varepsilon^2 + 3$ bins of sizes $1 + \varepsilon$.*

Proof: The bound of $1 + \varepsilon$ on the bin sizes follows from Lemma 4.1, by taking in the scheme as an input parameter $\varepsilon' = \varepsilon/2$.

For the running time of the scheme, we note that the program (D) has the same structure as the dual program of the classic bin packing problem given in [10]. It is easy to verify that the constraints added to the BP program, and the changes applied to the original constraints, can be handled by simple modifications to the "separating hyperplane oracle" proposed in [10]. In fact, since the number of constraints added to the program is at most ε^{-4}, it is possible to verify that none of them is violated by checking each of the constraints separately. Once a solution for (D) is obtained, the dual of the resulting reduced program, which has polynomial number of variables, can be solved using an algorithm for fractional covering (see, e.g., [11, 22], and a comprehensive survey in [23]). Finally, a basic solution for the reduced primal program can be obtained using, e.g., the algorithm of [19],whose running time is polynomial in the reduced size of (P). It follows that the resulting approximation scheme is fully polynomial.

■

5 Bin Packing with Size-Increasing Fragmentation (BP-SIF)

Recall that, in BP-SIF, the input is a list of n items, $I = (a_1, a_2, ..., a_n)$, each has the size $s(a_i) \in (0, 1]$. The number of splits is unbounded, but since there is a header of size Δ attached to each item or fragment, each fragmentation increases the input size by Δ, the size of an extra header. The goal is to pack all items using minimal number of bins. In this section we show how the approximation schemes developed for BP-SPF can be slightly modified to yield approximation schemes for BP-SIF. Note that *bin configuration*, the configuration matrix A, and the fragmentation matrix B are all well-defined for BP-SIF.

By Theorem 2.5, any algorithm that avoids unnecessary fragmentation uses at most $N_{opt}/(1-\Delta) + 1$ bins. Let $\varepsilon > 0$ be the parameter of the scheme. For any $\Delta \leq \varepsilon/(1+\varepsilon)$ it holds that $1/(1-\Delta) \leq (1+\varepsilon)$. Therefore,

Corollary 5.1 *If $\Delta \leq \varepsilon/(1+\varepsilon)$ then there is a linear time AFPTAS for BP-SIF.*

We note that when $\Delta > \varepsilon/(1+\varepsilon)$ the number of items or fragments packed in each bin does not exceed $1/\Delta < (1+\varepsilon)/\varepsilon$, which is a constant. This fact seems to simplify the problem; however, since small items are treated easily anyway, we are left with the challenge of packing the large items. The schemes for BP-SIF can be slightly simplified by taking $\varepsilon' = \varepsilon/(1+\varepsilon)$, which implies that there are no small items, and therefore the steps involving the small items can be skipped.

For an item a_i, the *actual size* of a_i, denoted by $s^+(a_i)$, is the volume required for packing a_i with no fragmentation; that is, $s^+(a_i) = s(a_i) + \Delta$.

5.1 A Dual PTAS

The scheme described in Section 3.1 can be applied for BP-SIF with the following changes. When partitioning the items by sizes, the *large* items have size at least $\varepsilon - \Delta$; all other items are *small*. (Note that if $\Delta \geq \varepsilon$ then all items are large.) Also, we round up the actual size, $s^+(a_i)$, of each large item to the nearest integral multiple of ε^2.

Lemmas 3.1 and 3.2 hold, and the scheme has the same running time. Hence, we have

Theorem 5.2 *BP-SIF admits a dual PTAS.*

5.2 An APTAS for BP-SIF

We distinguish between two cases:

1. $\Delta \leq \varepsilon/(1+\varepsilon)$: In this case by simply filling each bin with items in arbitrary order and splitting the last item if needed (and possible) we will always get a $(1+\varepsilon)N_{opt} + 1$ approximation. This is due to the fact that we will use at most $\lceil (1+\varepsilon)\sum_{i=1}^{n}(a_i + \Delta) \rceil$ bins while

$\sum_{i=1}^{n}(a_i + \Delta)$ is a lower bound on the number of bins needed. In this case the running time of the scheme is $O(n)$.

2. $\Delta > \varepsilon/(1+\varepsilon)$: In this case all of the items can be considered as *large*, since for each item, a_i, its actual size is $s(a_i) + \Delta > \varepsilon/(1+\varepsilon)$. This implies that each bin can only hold *less* than $(1+\varepsilon)/\varepsilon = 1/\varepsilon + 1$ distinct items/fragments (or in other words, each bin can hold *at most* $1/\varepsilon$ distinct items/fragments). We can now implement the scheme as described in Section 3.2, without the need to partition the input into large and small items - as all of our items are large. When using the scheme for BP-SIF, the only change is in the LP: the constraints (8), which guarantee that the fragments can be packed in the remaining capacity, in each bin, consider now the actual sizes of the fragments (see the details in Section 3.2).

$$\begin{array}{lll}
(LP-SIF) & \text{maximize} & \displaystyle\sum_{i=1}^{N}\sum_{j=1}^{m}\sum_{r=1}^{R}(x_r+\Delta)\ell_{ij}^{r} \\
& \textit{subject to}: & x_{r_1}+x_{r_2}\ge s_j \quad \forall j,\ (r_1,r_2)\in F_j \\
& & \displaystyle\sum_{j=1}^{m}\sum_{r=1}^{R-1}(x_r+\Delta)\ell_{ij}^{r}\le \Gamma_i \text{ for } i=1,\ldots,N \qquad (8) \\
& & x_r \ge 0 \text{ for } r=1,\ldots,R
\end{array}$$

Theorem 5.3 *BP-SIF admits an APTAS.*

5.3 An AFPTAS for BP-SIF

The scheme described in Section 4 can be applied for BP-SIF with the following modifications. (i) We start by adding Δ to the size of each item. (ii) In the fragmentation matrix B, each column now reflects a feasible fragmentation under BP-SIF; that is, if item a_i split into two fragments j and k, then $s(j) + s(k)$ equals to the sum $s(a_i) + \Delta$ rounded up to the nearest multiple of ε^2. In addition, $s(j) > \Delta$ and $s(k) > \Delta$. The analysis of the scheme is similar to analysis given in Section 4.

Theorem 5.4 *BP-SIF admits a dual AFPTAS.*

Acknowledgments We would like to thank Yuval Rabani for helpful comments and suggestions. We also thank two anonymous referees who read the paper carefully and contributed significantly to the accuracy and the clarity of the presentation.

References

[1] O. Braun and G. Schmidt, Parallel Processor Scheduling with Limited Number of Preemptions. *SIAM J. Comput.*, 32:3, 671–680, 2003.

[2] P. Brucker, *Scheduling algorithms*, fourth edition, Springer-Verlag, Berlin, Germany, 2004.

[3] 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.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms*, 2nd Edition, MIT Press and McGraw-Hill, 2002.

[5] 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.

[6] 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,* volume 1643 of *Lecture Notes in Computer Science,* 151-162. Springer-Verlag, 1999.

[7] 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.

[8] M. Grötschel, L. Lovász and A. Schrijver, The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, I, 169–197, 1981.

[9] 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.

[10] 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.

[11] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–396, 1984.

[12] J. Y-T. Leung (ed.), *Handbook of scheduling: Algorithms, models and performance analysis*, Computer and Information Science Series, Chapman & Hall / CRC, Boca Raton, Florida, 2004.

[13] 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.

[14] R. McNaughton. Scheduling with deadlines and loss functions. *Manage. Sci.*, 6:1–12, 1959.

[15] N. Menakerman and R. Rom. Bin Packing Problems with Item Fragmentations. *Proc. of WADS*, 2001.

[16] R. Motwani. Lecture notes on approximation algorithms. Technical report, Dept. of Computer Science, Stanford Univ., CA, 1992.

[17] Multimedia Cable Network System Ltd., Data-Over-Cable Service Interface Specification, *http://www.cablelabs.com*, 2000.

[18] N. Naaman and R. Rom. Packet Scheduling with Fragmentation. *Proc. of INFOCOM'02*, 824-831, 2002.

[19] P. Beling and N. Megiddo, Using fast matrix multiplication to find basic solutions. *Theoretical Computer Science* 205 (1998) 307-316.

[20] H. Shachnai, T. Tamir and G.J Woeginger. Minimizing Makespan and Preemption Costs on a System of Uniform Machines, *Algorithmica* 42, 309–334, 2005.

[21] F. Sourd, Preemptive scheduling with position costs. *Algorithmic Operations Research* AOR Vol. 1, Number 2, 2006.

[22] D.A. Spielman and S-H Teng. Smoothed Analysis of Termination of Linear Programming Algorithms. *Math. Program., Ser. B 97*, 375-404, 2003.

[23] M.J. Todd, The Many Facets of Linear Programming. *Math. Program., Ser. B 91*, 417-436, 2002.

[24] V.V. Vazirani. Bin Packing. In *Approximation Algorithms*, 74-78, Springer, 2001.