Approximation Schemes for Packing with Item Fragmentation

Omer Yehezkely

Approximation Schemes for Packing with Item Fragmentation

Research Thesis

Submitted in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

Omer Yehezkely

Submitted to the Senate of the Technion – Israel Institute of Technology

Heshvan 5767 Haifa November 2006

The research thesis was done under the supervision of Professor Hadas Shachnai in the Department of Computer Science.

The generous financial help of the Technion and the Sam Cohen Windhoek Foundation Trust is gratefully acknowledged.

Contents

Abstract

We consider two 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, and the items may be *fragmented*. This can potentially reduce the total number of bins used 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 that both problems admit *dual* PTAS, *asymptotic PTAS (APTAS)*, and *asymptotic fully polynomial time approximation scheme (AFPTAS)*. In particular, given an instance I of BP-SPF, and $\varepsilon > 0$, we develop

- A *dual* PTAS which packs the items in $OPT(I)$ bins of sizes $1+\varepsilon$, where $OPT(I)$ is the number of bins used by an optimal packing.
- An APTAS which packs the items using at most $OPT(I)(1+\varepsilon)+1$ unit-sized bins.
- A dual AFPTAS, which packs the items into $OPT(I)+k$ unit-sized bins of size $1+\varepsilon$, where $k \leq 4/\varepsilon^2 + 3$.
- An AFPTAS whose running time is linear in n, the number of items in I, which packs the items into $OPT(I)+k$ unit-sized bins, where $k = O(\varepsilon^{-1}\log\varepsilon^{-1})$.

We show that all of these schemes can be applied with slight changes to BP-SIF.

Our AFPTAS yields as a special case AFPTAS with improved running time for the classic *bin packing* problem.

In developing approximation schemes for packing with item fragmentation, we apply several techniques, which are of independent interest. This includes a non-standard transformation of mixed packing and covering linear programs into pure covering programs, and a novel *oblivious* version of the shifting technique, that is used for obtaining a fixed number of distinct fragment sizes in the instance.

List of Symbols and Abbreviations

AFPTAS	Asymptotic Fully Polynomial Time Approximation Scheme
APTAS	Asymptotic Polynomial Time Approximation Scheme
BP	Bin Packing
BP-SIF	Bin Packing with Size Increasing Fragmentation
BP-SPF	Bin Packing with Size Preserving Fragmentation
BPF	Bin Packing with item Fragmentation
CATV	Community Antenna Televesion
DOCSIS	Data-Over-Cable Service Interface Specification
IC	Item Cover
LP	Linear Program
OPT(I)	Optimal solution value for the Instance I
PTAS	Polynomial Time Approximation Scheme
VLSI	Very Large Scale Integration
Δ	Delta, the size of the added header/footer
ε	epsilon, the approximation parameter

Chapter 1

Introduction

In the following sections we give basic definitions of approximation schemes and of the studied problems.

1.1 Approximation Schemes

Definition 1.1.1 *(Approximation Algorithms)*
Let Π *be a minimization problem. Let* $\varepsilon > 0$*, and set* $\rho = 1 + \varepsilon$*.*

- *An algorithm* A *is called a* ρ*-approximation algorithm for problem* Π*, if for all instances* I *of* Π *it delivers a feasible solution with objective value* $A(I)$*, such that* $A(I) - OPT(I) \leq \varepsilon \cdot OPT(I)$*.*
- *An algorithm* A *is called an asymptotic* ρ*-approximation algorithm for problem* Π*, if for all instances* I *of* Π *it delivers a feasible solution with objective value* $A(I)$*, such that* $A(I) - OPT(I) \leq \varepsilon \cdot OPT(I) + k$*, where* k *is a constant.*

Definition 1.1.2 *(Approximation Schemes)*
Let Π *be a minimization problem.*

- *An (Asymptotic) Approximation Scheme for problem* Π *is a family of* $(1+\varepsilon)$ *(asymptotic) approximation algorithms* A_ε *for problem* Π *over all* $0 < \varepsilon < 1$*.*
- *A (Asymptotic) Polynomial Time Approximation Scheme ((A)PTAS) for problem* Π *is an (asymptotic) approximation scheme whose time complexity is polynomial in the input size.*

- *A (Asymptotic) Fully Polynomial Time Approximation Scheme ((A)FPTAS) for problem* Π *is an (asymptotic) approximation scheme whose time complexity is polynomial in the input size and in* $1/\varepsilon$.

1.2 Problem Statement

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 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.

1.2.1 Size Preserving Fragmentation (BP-SPF)

An item a_i can 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* $C \geq 0$. Note that in the special case where $C = 0$ we get an instance of the classic bin packing.

1.2.2 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 a volume of $s(a_{i_j}) + \Delta$.

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 $(0.3 + 0.1, 0.5 + 0.1)$.

1.3 Hardness of BP-SIF and BP-SPF

The hardness of approximation of both BP-SIF and BP-SPF can be shown by a simple reduction from Partition. The Partition problem is defined as follows:

Given a multiset S of positive integers, is there a way to partition S into two subsets S_1 and S_2, such that the sums of the numbers in each subset are equal? The subsets S_1 and S_2 must form a partition in the sense that they are disjoint and they cover S.

The Partition problem is well known to be NP-complete [10].

Theorem 1.3.1 *Unless $P = NP$, there is no polynomial time approximation algorithm for BP-SIF or BP-SPF that guarantees an approximation of $(1 + \varepsilon)$, where $\varepsilon < 1/2$.*

Proof: Given an instance I of the Partition problem, denote by $Sum(I)$ the sum of elements in the instance.

1. **BP-SIF:** Generate an instance I' in the following way: for each element e_i in I, create an item a_i in I', such that $s(a_i) = 2e_i/Sum(I)$.

   Denote by $s(I')$ the sum of the item sizes in I'; clearly, $s(I') = 2$. Now, assuming A is a polynomial time algorithm that guarantees an approximation of $(1 + \varepsilon)$, where $\varepsilon < 1/2$, if I can be partitioned, we get a packing of I' in 2 bins, otherwise we get a packing of I' in more than 2 bins.

2. **BP-SPF:** The arguments are the same as for BP-SIF, except that we need to set the fragmentation budget C to zero in I'.

■

In fact, as shown in [27], it is NP-hard to avoid even a *single* split in BP-SPF, and there is no approximation scheme for BP-SIF with a constant additive error, unless $P = NP$.

The above results are a bit discouraging, however we show in the rest of this thesis that both BP-SIF and BP-SPF can be approximated asymptoticly (as is the case for classic bin packing).

1.4 Applications

The following applications motivate our study.

1.4.1 Community Antenna Television 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 [22]. When using Community Antenna Television (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 [20] for more details).

1.4.2 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 internal variables each, 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 two variables, 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 [19].)

1.4.3 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.

1.4.4 Flexible Packaging

In some packaging problems, the cost of the packages is substantive. This includes:

- 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.
- 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.

1.5 Related Work

It is well known (see, e.g., [21]) 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 [10]. However, there exists an APTAS which uses, for any instance I, $(1+\varepsilon)\,OPT(I) + k$ bins for some fixed k.

Fernandez de la Vega and Lueker [8] presented an APTAS with $k = 1$, and Karmarkar and Karp [15] presented an AFPTAS with $k = 1/\varepsilon^2$.The scheme of [15] is based on rounding the (fractional) solution of an 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]. 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. [25].

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 [7] on multiprocessor scheduling, since BP is dual to the Minimum Makespan problem. (Comprehensive surveys on the bin packing problem appear, e.g., in [4, 31].)

Another related problem is the *Variable-sized Bin Packing* problem,in which we have a set of items whose sizes are in $(0, 1]$, and a set of bins whose sizes are in $(0, 1]$. 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 [9], who gave several approximation algorithms, the best of which has asymptotic worst case ratio of $4/3$. Murgolo [23] presented an asymptotic FPTAS, which solves a covering linear program using the techniques of [15]. Thus, the running time of the scheme is essentially the same as the running time of the AFPTAS proposed in [15] for classic bin packing.

Mandal et al. introduced in [19] the BP-SIF problem and showed that it is NP-hard. Menakerman and Rom [20] and Naaman and Rom [24] were the first to develop algorithms for bin packing with item fragmentation, however, the problems studies in [20] and [24] 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 [20] 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. Shachnai, Tamir and Yehezkely [27] showed that it is NP-hard to avoid a *single* split in BP-SPF, for any number of bins, even if the existence of packing that uses no splits is known a-priori. It is also shown in this paper that there is no approximation scheme for BP-SIF with a constant additive error, unless $P = NP$.

There has been some related work in the area of preemptive scheduling on parallel machines. The paper [26] 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 [2, 28]. (Comprehensive surveys of known results on preemptive scheduling are given, e.g., in [3, 18].)

Chapter 2

Preliminaries

In this chapter we give some basic lemmas and properties of packing with item fragmentation. We 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.

In chapter 6, we show that the existence of this structure is a much stronger property of the problems, as it allows us to convert our problems of packing with *fragmentation* into 'simpler' problems.

2.1 Primitive Packings

Definition 2.1.1 *(Bin Packing Graph)*
The bin packing graph (BP graph) *of a given packing is 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.

Definition 2.1.2 *(Primitive Packing)*
A primitive packing *is a feasible packing in which:*

1. *Each bin has at most two fragments of items.*
2. *Each item can be fragmented only once.*

Note that the respective bin packing graph is a collection of paths.

2.2 Primitive Optimal Solutions

Lemma 2.2.1 *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. ■

Lemma 2.2.2 *Any instance of BP-SIF has an optimal packing that is primitive.*

Proof: Similar to the proof of Lemma 2.2.1, 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 note 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. ■

These lemmas allow us to reduce the search for an optimal solution to the set of primitive solutions.

2.3 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 $i \cdot \delta$ for some integer i.

Theorem 2.3.1 *For a fixed positive integer U every discrete instance I can be solved in polynomial time complexity.*

Proof: First we can consider only primitive solutions, hence each item can be fragmented only once and each bin can have up to 2 fragments.

Since the number of items/fragments in a bin is bounded by U, and since the number of different item sizes a fragment can originate from is also bounded by U, the number of distinct bin types is a function of U and hence also a constant. Denote this number by R.

Since the number of bins used is at most n, the number of possible feasible packings to consider is bounded by $P = \begin{pmatrix} n+R \\ R \end{pmatrix}$, which is polynomial in n. Enumerating all the possibilities and picking the best feasible packing will give us the optimal solution. ■

Chapter 3

A Dual PTAS for BP-SPF

Recall that in BP-SPF we are given a list of n items $I = (a_1, a_2, ..., a_n)$, each having a 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 chapter we develop a dual PTAS for BP-SPF.

3.1 Overview

Given an input I and some $\varepsilon > 0$, the 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, using Next-Fit algorithm.

3.2 Analysis of the Scheme

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

Lemma 3.2.1 *It is possible to pack the rounded large items into $OPT(I)$ bins of size $(1+\varepsilon+\varepsilon^2)$.*

Proof: By Lemma 2.2.1, 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 $\varepsilon^2(1+1/\varepsilon) = (\varepsilon+\varepsilon^2)$. ■

Note that by theorem 2.3.1 such a packing can be found in polynomial time complexity.

Lemma 3.2.2 *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 (1), 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 headers packed in step (4) 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 3.2.3 *BP-SPF admits a dual PTAS.*

Proof: Given an input parameter $\varepsilon' > 0$, then by using Lemmas 3.2.1 and 3.2.2, 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 (1) and (2) are linear and are done once. For Step (3), 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 (4), 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. ■

Chapter 4

An APTAS for BP-SPF

We describe below an asymptotic PTAS for BP-SPF. Given an $\varepsilon > 0$, the scheme packs any instance I into at most $OPT(I)(1+\varepsilon)+1$ bins. The 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.

4.1 Overview

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.

(viii) Pack the remaining large items and small items in an arbitrary order, one at a time, into the bin having maximum free space.

4.2 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 a shifting technique called *linear grouping* (see, e.g., in [31]).

Linear grouping is performed in the following way: 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.

Lemma 4.2.1 *After applying linear grouping, the resulting instance without the largest H items can be packed in $OPT(I)$ bins.*

Proof: Denote the resulting instance after linear grouping by I'. Next, consider a shifting technique which acts exactly like linear grouping but rounds each item down to the size of the smallest item in its subset. Denote the resulting instance I''. Clearly I'' can be packed in $OPT(I)$ bins. However, we can use any packing of I'' to pack all of I' except of the largest H items. (The second subset of I'' will be used for packing the first subset of I', the third subset of I'' will be used for packing the second subset of I', and so on). ■

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:

1. 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.

2. An 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. (in case there are 2 fragments of the same size group, the vector will have a single entry of 2), the rest of the entries are zero.

3. An 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. (in case there are 2 fragments of the same type, the vector will have a single entry of 2), the rest of the entries are zero.

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

4.3 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 $y_1 \leq y_2 \cdot \cdot \leq y_{2c}$, 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.)

Lemma 4.3.1 *The resulting up-shifted fragments, except for the largest Q fragments, can be packed in $OPT(I)$ bins, after packing the non-fragmented and up-shifted large items (except for the largest H items) into these bins.*

Proof: Consider a primitive optimal solution of I. We perform down-shifting on the non-fragmented items. This can only increase the volume available for the fragments in each bin, and, as stated in Lemma 4.2.1, it can be used for packing the large items except for the largest H items.

As for packing the up-shifted fragments, again as in the proof of Lemma 4.2.1, we observe that each packing of down-shifted fragments generates a packing of the up-shifted fragments, except for the largest Q up-shifted fragments. ∎

Next, find the type of fragments generated for each of the c guessed items. Note that the number of pairs of fragment sizes is 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.

4.4 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_{ij}^r 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.

$$
\begin{aligned}
(LP) \qquad \text{maximize} \qquad & \sum_{i=1}^{N} \sum_{j=1}^{m} \sum_{r=1}^{R-1} x_r \ell_{ij}^r \\
\textit{subject to}: \qquad & x_{r_1} + x_{r_2} \geq s_j \quad \forall j,\ (r_1, r_2) \in F_j \\
& \sum_{j=1}^{m} \sum_{r=1}^{R-1} x_r \ell_{ij}^r \leq \Gamma_i \text{ for } i = 1, \ldots, N \qquad (4.1) \\
& x_r \geq 0 \text{ for } r = 1, \ldots, R-1 \\
& x_R = 1
\end{aligned}
$$

Inequality (4.1) 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 $H + Q$ 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 4.4.1 *For any ε, $0 < \varepsilon \leq 1/2$, $H + Q \leq 2\varepsilon OPT(I)$.*

Proof: We first note that $H + Q \leq n\varepsilon^2 + 2c\varepsilon^2$. Since each large item is at least of size ε, we get that $OPT(I) \geq n\varepsilon$. Hence, $H + Q \leq \varepsilon OPT(I) + 2c\varepsilon^2$. In addition, $OPT(I) \geq c$, therefore $H + Q \leq \varepsilon OPT(I) + 2\varepsilon^2 OPT(I)$. Combining this with the fact that $\varepsilon \leq 1/2$ we get the statement of the lemma. ∎

Lemma 4.4.2 *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 4.4.3 *There is an asymptotic PTAS for BP-SPF.*

Chapter 5

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.

5.1 Overview

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 $d = OPT(I)$, the number of bins used by an optimal packing of I.

(iv) Guess $c \leq C$, the number of fragmented items (or simply assume $c = C$).

(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.

5.2 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 $p \leq 1/\varepsilon^4$.

A valid fragmentation vector has the following form: One entry with a value of $+1$ (this is the corresponding entry to the size of the fragmented item), two entries with a value of -1 (these are the entries which correspond to the fragments sizes, in case both fragments are of the same size the vector will have a single entry of value -2) and the rest of the entries are 0.

It should be noted that both matrices A and B, are defined according to ε. The given input of the item sizes has no affect on the matrices.

5.3 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.

$$
\begin{aligned}
(P_1) \qquad \text{minimize} \qquad & \sum_{i=1}^{q} x_i \\
\textit{subject to}: \qquad & \sum_{i=1}^{q} A_{ij} x_i + \sum_{k=1}^{p} B_{kj} z_k \geq n_j \qquad \text{for } j = 1, \ldots, m \qquad (5.1) \\
& \sum_{k=1}^{p} z_k \leq c \qquad (5.2) \\
& x_i \geq 0 \text{ for } i = 1, \ldots, q \\
& z_k \geq 0 \text{ for } k = 1, \ldots, p
\end{aligned}
$$

The constraints (5.1) reflect the coverage requirement for the n_j items of size group j; the constraint (5.2) 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.

$$
\begin{aligned}
(P) \qquad \text{minimize} \qquad & \sum_{i=1}^{q} x_i + \sum_{k=1}^{p} z_k \\
\textit{subject to}: \qquad & \sum_{i=1}^{q} A_{ji} x_i + \sum_{k=1}^{p} B_{jk} z_k \geq n_j \qquad \text{for } j = 1, \ldots, m \\
& \sum_{i=1}^{q} x_i \geq d \qquad (5.3) \\
& \sum_{k=1}^{p} z_k \geq c \qquad (5.4) \\
& x_i \geq 0 \text{ for } i = 1, \ldots, q \\
& z_k \geq 0 \text{ for } k = 1, \ldots, p
\end{aligned}
$$

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.

$$
\begin{aligned}
(D) \qquad \text{maximize} \qquad & \sum_{j=1}^{m} n_j y_j + d y_{m+1} + c y_{m+2} \\
\textit{subject to}: \qquad & \sum_{j=1}^{m} A_{ji} y_j + y_{m+1} \leq 1 \text{ for } i = 1, \ldots, q \qquad (5.5) \\
& \sum_{j=1}^{m} B_{jk} y_j + y_{m+2} \leq 1 \text{ for } k = 1, \ldots, p \qquad (5.6) \\
& y_j \geq 0 \text{ for } j = 1, \ldots, m+2
\end{aligned}
$$

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 [15]; The differences between (D) and the dual of the classical bin packing LP are:

- The addition of y_{m+1} and y_{m+2} to the objective function.
- The addition of y_{m+1} to constraints of type (5.5) and (5.6).
- The addition of type 5.6 constraints (There are less than $m^2 = \varepsilon^{-4}$ constraints of this type).

Recall that for any given point $z \in \Re^n$, the *"separating hyperplane oracle"* provided by [15] either determines that z is a feasible solution, or else produces a vector d such that $d \cdot z > d \cdot y$ for every feasible solution y. This can be achieved by finding the *"most violated"* constraint in (D) for the given z. Finding the *"most violated"* constraint of type (5.5) can be done exactly as proposed by [15] since the coefficient of y_{m+1} in each of these constraints is 1. Since the number of constraints of type (5.6) is no more than ε^{-4} we conclude that a "separating hyperplane oracle" can be implemented in polynomial time.

Applying this technique results in the reduction of the number of constraints in (D) to $O(m^2 log(\varepsilon^{-1} n))$, the number of *"most violated"* constraints. The exact solution of this *reduced* (D) will be within an additive of 1 from the optimal solution of (P). Therefore, by solving the dual program of this *reduced* (D) (which is actually a *reduced* (P)), we get a solution that is within an additive of 1 from the optimal solution for (P).

Next, we transform this 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.

5.4 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.

5.5 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 5.5.1 *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 3.2.1, 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 5.5.2 *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 5.5.3 *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 5.5.1, 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 [15]. 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 [15]. 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., [14, 29], and a comprehensive survey in [30]). Finally, a basic solution for the reduced primal program can be obtained using, e.g., the algorithm of [1],whose running time is polynomial in the reduced size of (P). It follows that the resulting approximation scheme is fully polynomial.

■

Chapter 6

A Linear Time AFPTAS for BP-SPF

In this chapter we describe an AFPTAS for BP-SPF. The main issue in developing such a scheme is how to deal with fragmentation. Indeed, the number of ways to fragment an item is virtually *unbounded*. Moreover, the fragment sizes, as well as the need to fragment items, depend on the selection of *non-fragmented* items. Finally, rounding the fragment sizes may cause bin overflow.

To overcome these difficulties, we turn back to the primitive packing property (as given in Lemmas 2.2.1 and 2.2.2) and rephrase our variants of BPF as problems of packing *connected components* with no fragmentation.

The resulting scheme has a *linear* running time, and it packs the items into bins of sizes $1, 2, ..., c+1$ with no fragmentation. In addition, we improve the best known running time of a scheme for the classical Bin Packing problem (where $C = 0$), as described later on in this chapter.

6.1 Overview

The following is an overview of the scheme:

(i) Partition the instance into *large* and *small items*; any item with size at least ε is large.

(ii) Transform the instance into an instance where the number of distinct item sizes is fixed.

(iii) Define the configuration matrices A and B for packing large items in unit-sized bins and in larger bins respectively.

(iv) Guess $d = OPT(I)$.

(v) Efficiently obtain a basic solution for a covering LP.

(vi) Pack (and if needed fragment) the large items according to the basic solution of the LP.

(vii) Pack the small items using the Next-Fit algorithm.

6.2 Reformulation of the Problem

In the following we reformulate BP-SPF as a problem of packing without fragmentation:

Input: A set of items $a_1, a_2, ..., a_n$ with sizes $s(a_1), s(a_2), ..., s(a_n) \in (0, 1]$, an integer budget $C \geq 0$.

Output: Using bins of sizes $1, 2, ..., C + 1$ with respective costs $0, 1, ..., C$ (there is no restriction on the number of bins of each size), pack all items in a minimum volume of bins, such that the total cost of bins used does not exceed C.

We call this problem below *item cover (IC).*

Lemma 6.2.1 *The IC problem is equivalent to BP-SPF.*

Proof: Each connected component C_i in Lemma 2.2.1 can be viewed as a bin with capacity $|C_i|$. Since the cost of such bin is exactly the number of fragments induced by C_i, we conclude that any solution for BP-SPF can be transformed into a solution of IC, and vice versa.

Since both problems attempt to minimize the total volume of used bins, it follows that the two problems are equivalent. ∎

Lemma 6.2.2 *Given a feasible solution of an IC instance in bins of sizes* $1, \ldots, C$*, using a total bin volume of* N*, there exists a solution that uses bins of sizes* $1, \ldots, \lceil 1/\varepsilon \rceil$*, such that the total bin volume used is at most* $N(1 + \varepsilon)$*.*

Proof: By replacing each bin of size $T \geq \lceil 1/\varepsilon \rceil$ with a set of at most $\lfloor \varepsilon T \rfloor$ bins of size $\lceil 1/\varepsilon \rceil$, one bin (if necessary) of size $T \ mod \ \lceil 1/\varepsilon \rceil$, and at most $\lfloor \varepsilon T \rfloor$ unit-sized bins, we can repack the contents of the bin while adding at most volume of εT. Thus, we conclude that the total volume of bins that we use is at most $N(1 + \varepsilon)$. ∎

6.3 Preprocessing the Input

Karmarkar & Karp [15] presented an algorithm for grouping item sizes in a clever way which they call "geometric grouping". This algorithm is based on partitioning the range of item sizes in a geometric way, i.e., to the intervals $\{[0.5, 1), [0.25, 0.5), [0.125, 0.25), ..., [2^{-\ell} \leq \varepsilon, 2^{1-\ell})\}$, and performing linear grouping (Section 4.2) in each interval, while doubling the group sizes as we move between the intervals. Using a group size of $k = s(I)\varepsilon/\log(2\varepsilon^{-1})$ for the interval $[0.5, 1)$, $2k$ for $[0.25, 0.5)$ and so on... Karmarkar & Karp showed in [15] that the resulting number of distinct sizes is $m = O(\varepsilon^{-1}\log\varepsilon^{-1})$, while the optimal solution of the instance requires at most $(1+\varepsilon)OPT(I)$ bins. This holds true also in our problems, as shown in Lemma 4.2.1.

6.4 Defining the Configuration Matrices

We proceed to define for the large items the *configuration matrix*, A, for unit-sized bins in which there are no fragmented items. In particular, for the shifted large items, a *bin configuration* is a vector of size m, 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 $q \leq (1/\varepsilon)^m$. Next, we define a matrix B including as columns the configurations of *oversized* bins. Each of these bins has a size in the range $[2, min(c, \lceil 1/\varepsilon \rceil)]$. 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 $s = O((1/\varepsilon^2)^m)$.

6.5 Solving the Linear Program

6.5.1 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 capacity of the k-th oversized bin 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.

6.5.2 Finding Approximate Solution

A technique developed by Young [32], 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]).

Young's technique is based on repeated calls to an oracle. In our case, the oracle needs to solve $1/\varepsilon$ instances of the Knapsack problem. As stated in [32] the oracle can produce approximate solutions ($(1-\varepsilon)$ approximation), and the technique will still produce a $(1+\varepsilon)$ approximate solution. Using Kellerer & Pferscy's approximation scheme (in [16]) for the Knapsack problem, an oracle call can be implemented in $O(\varepsilon^{-4} \log^2 \varepsilon^{-1})$ steps.

Alternatively, we can solve a Knapsack 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 [6], such IP in fixed dimension can be optimally solved in $O(mM)$ steps, where M is the longest binary representation of any input element.[1]

Since we have to solve this problem $1/\varepsilon$ times, we get that an oracle that solves the Knapsack problems exactly will have a running time of $O(\varepsilon^{-1}mM)$. Thus, we get that the running time of each oracle call is $O(min(\varepsilon^{-4}log^2\varepsilon^{-1}, \varepsilon^{-1}mM))$.

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 systems of linear equations to a *basic* solution.

6.6 Packing the Items

Given the (fractional) solution for the 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.[2]

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

Proof: 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.

Now, some of the large items may need to be packed (since we rounded down the x_i and y_k values). We generate oversized bins from the fractions of the y_k values. This is done in

[1] 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.

[2] Detailed descriptions of First Fit and Next Fit can be found, e.g., in [4].

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 Next Fit, in linear time, adding at most $4m+8$ unit-sized bins. Finally, we pack the small items. Again, this can be done in linear time, using Next Fit. ■

6.7 Analysis of the Scheme

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

Lemma 6.7.1 *The total number of fragmented items is at most C.*

Proof: 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 $1 \leq k \leq s$. ■

Next, we bound the total number of bins used by the scheme.

Lemma 6.7.2 *The input is packed in at most $(1+\varepsilon)^2(1+2\varepsilon)OPT(I) + 4(m+2)$ bins.*

Proof: Using geometric grouping (Section 6.3) and bounding the number of distict bin sizes (Lemma 6.2.2), we get an approximation of $(1+\varepsilon)^2 OPT(I)$

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 $(1+\varepsilon)^2(1+2\varepsilon)OPT(I)$-approximation for the optimal number of bins.

Since we use a rounded down basic solution (with at most $m+2$ variables), after packing according to the rounded solution and also packing in the extra oversized bins, the total volume of the remaining 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 the size of the extra bins is rounded down).

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 a $(1+\varepsilon)OPT(I)+O(m)$ approximation scheme.

■

We now analyze the running time of the scheme. We need the following technical lemma:

Lemma 6.7.3 *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 $O(n\log\varepsilon^{-1} + \varepsilon^{-y}\log^{x+z}\varepsilon^{-1})$.*

Proof: If $n \geq \log^x n \cdot \varepsilon^{-y}\log^z\varepsilon^{-1}$, then the running time can be bounded by $O(n\log\varepsilon^{-1})$; else, we have that $O(\log n) = O(\log(\log^x n \cdot \varepsilon^{-y}\log^z\varepsilon^{-1})) = O(\log\varepsilon^{-1} + \log\log n)$. Hence, $O(\log n) = O(\log\varepsilon^{-1})$.

■

Lemma 6.7.4 *The above scheme can be implemented in linear time.*

Proof: The running time of the scheme is determined by the following steps:

(1) Preprocess the input: we first partition the items to 'large' and 'small'; then, we apply shifting to the large items to obtain $m = O(\varepsilon^{-1}\log\varepsilon - 1)$ distinct items sizes. This can be implemented in $O(n\log m)$ steps (see, e.g., in [25]).

(2) Applying the technique of Young [32] for solving the LP, we can use a result of [13] to bound the number of calls to the oracle by $O(m(\log m + \varepsilon^{-2}))$. By Section 6.5.2, overall, this requires $O(m(\log m + \varepsilon^{-2})min(\varepsilon^{-4}log^2\varepsilon^{-1}, \varepsilon^{-1}mM))$ steps for solving the LP. Since we need to 'guess' d, the optimal number of bins, we solve the LP $O(\log n)$ times.

(3) Applying a technique of Beling and Megiddo [1], which transforms the solution for LP into a basic one, requires $O(m^{1.62}m(\log m + \varepsilon^{-2}))$ steps.

(4) Finally, we pack the items and divide each oversized bin into a set of ordinary bins. By Lemma 6.6.1, this can be done in $O(n+m)$ steps.

Hence, we get that the total running time of the scheme is $O(n\log m + \log n \cdot m(\log m + \varepsilon^{-2})\cdot min(\varepsilon^{-4}\log^2\varepsilon^{-1}, \varepsilon^{-1}mM)) = O(n\log\varepsilon^{-1} + \log n\cdot\varepsilon^{-3}\log\varepsilon^{-1}\cdot min(\varepsilon^{-4}\log^2\varepsilon^{-1}, \varepsilon^{-2}\log\varepsilon^{-1}M))$.

From Lemma 6.7.3 we get that the running time can be bounded by $O(n\log\varepsilon^{-1} + \varepsilon^{-3}\log^2\varepsilon^{-1}\cdot min(\varepsilon^{-4}\log^2\varepsilon^{-1}, \varepsilon^{-2}\log\varepsilon^{-1}M))$.

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

Summarizing the above discussion, we have

Theorem 6.7.5 *There is a linear time AFPTAS for BP-SPF.*

6.8 Application for the Bin Packing Problem

Recall that the bin packing problem is a special case of BP-SPF in which $C = 0$. Thus, when applying the above scheme for BP, we do not need to consider oversized bins. This means that the oracle needs to solve only 'simple' instances of the Knapsack problem (which consist of unit-sized bins).

We turn to calculate the resulting running time of the scheme.

- **Oracle calls:** Since in each configuration there are at most $1/\varepsilon$ items, and there are $m = \varepsilon^{-1}\log\varepsilon^{-1}$ distinct item types, solving the Knapsack problem using the scheme of [16], we get the running time $O(\varepsilon^{-3}\log\varepsilon^{-1})$. Alternatively we can solve an *integer program* with a running time of $O(mM)$. Hence, the oracle running time is $O(min(\varepsilon^{-3}\log\varepsilon^{-1}, mM))$

- **Solving the LP:** For solving the LP we need $O(m(\log m + \varepsilon^{-2})) = O(\varepsilon^{-3} \log \varepsilon^{-1})$ oracle calls; we need to solve the LP $O(\log n)$ times. Therefore, we get a total running time of $O(\log n \cdot \varepsilon^{-3} \log \varepsilon^{-1} \cdot min(\varepsilon^{-3} \log \varepsilon^{-1}, mM))$.
- **Preprocessing:** Preprocessing the input requires $O(n \log \varepsilon^{-1})$ steps.
- **Packing:** The items are packed in $O(n + m)$ steps.

Hence, we get a total running time of: $O(n \log \varepsilon^{-1} + \log n \cdot \varepsilon^{-3} \log \varepsilon^{-1} \cdot min(\varepsilon^{-3} \log \varepsilon^{-1}, mM))$. By Lemma 6.7.3, this can be bounded by $O(n \log \varepsilon^{-1} + \varepsilon^{-3} \log^2 \varepsilon^{-1} \cdot min(\varepsilon^{-3} \log \varepsilon^{-1}, mM))$.

Theorem 6.8.1 *There is an AFPTAS for BP which packs the items in* $(1 + \varepsilon)OPT(I) + O(\varepsilon^{-1} \log \varepsilon^{-1})$ *bins, and whose running time is* $O(n \log \varepsilon^{-1} + \varepsilon^{-3} \log^2 \varepsilon^{-1} \cdot min(\varepsilon^{-3} \log \varepsilon^{-1}, mM))$.

This is the fastest AFPTAS for BP that we are aware of.

Chapter 7

Bin Packing with Size-Increasing Fragmentation

In BP-SIF, the input is a list of n items, $I = (a_1, a_2, ..., a_n)$, each has the size $s(a_i) \in (0, 1 - \Delta]$. The number of possible splits is unbounded, but since a header/footer of size Δ is attached to each item or fragment, each fragmentation increases the volume to be packed by Δ, the size of an extra header. The goal is to pack all items using minimal number of unit-sized bins. In this chapter, 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.

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, $s^+(a_i) = s(a_i) + \Delta$.

7.1 Easy Instances of BP-SIF

If Δ is small enough then there is a simple linear time AFPTAS for BP-SIF. This is formalized in the next theorem.

Theorem 7.1.1 *If $\Delta \leq \varepsilon/(1+\varepsilon)$ then there is a linear time AFPTAS for BP-SIF.*

Proof: Denote by N the number of bins used by greedily packing the items into the bins and fragmenting the last item in each bin (if needed and possible).

Using this algorithm, at most a volume of Δ is wasted in each bin, except maybe the last bin. Therefore we get that $OPT(I) \geq s^+(I) \geq (1-\Delta)(N-1) = \frac{1}{1+\varepsilon}(N-1) = \frac{N}{1+\varepsilon} - \frac{1}{1+\varepsilon}$. Hence, $(1+\varepsilon)OPT(I) + 1 \geq N$. ∎

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.

7.2 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 *actual size* of at least ε; all other items are *small*. (Note that, if $\Delta \geq \varepsilon$, 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 3.2.1 and 3.2.2 hold, and the scheme has the same running time. The resulting scheme proceeds in the following steps.

(i) If $\Delta \leq \varepsilon/(1+\varepsilon)$ pack the items greedily as described in Theorem 7.1.1, else go to (ii).

(ii) Add Δ to the size of each item.

(iii) Partition the items into two groups by their sizes: the *large* items have size at least ε; all other items are *small*.

(iv) Round up the size of each large item to the nearest integral multiple of ε^2.

(v) Guess $OPT(I)$, the number of bins used by an optimal packing of I.

(vi) Pack optimally the large items with fragmentation, using at most C splits, into $OPT(I)$ bins of size $(1+\varepsilon+\varepsilon^2)$.

(vii) Pack the small items in an arbitrary order, using algorithm Next-Fit.

Theorem 7.2.1 *BP-SIF admits a dual PTAS.*

7.3 An APTAS for BP-SIF

Since the actual size of each item, a_i, is $s(a_i) + \Delta > \varepsilon/(1+\varepsilon)$, each bin can only hold *less* than $(1+\varepsilon)/\varepsilon = 1/\varepsilon + 1$ distinct items/fragments. We can now implement the scheme as described in Chapter 4, with no 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 (7.1), which guarantee that the fragments can be packed in the remaining capacity in each bin, refer now to the actual sizes of the fragments (see the details in Chapter 4).

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

The following is an overview of the scheme.

(i) If $\Delta \leq \varepsilon/(1+\varepsilon)$ pack the items greedily as described in theorem 7.1.1, else go to (ii).

(ii) Guess $OPT(I)$ and $c \leq \lceil s(I) \rceil - 1$, the number of fragmented items.

(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.

(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.

(viii) Pack the remaining large items.

Theorem 7.3.1 *BP-SIF admits an APTAS.*

7.4 A Dual AFPTAS for BP-SIF

The scheme described in Chapter 5 can be applied for BP-SIF with the following modifications:

1. We start by adding Δ to the size of each item.
2. In the fragmentation matrix B, each column now reflects a feasible fragmentation under BP-SIF; that is, if item a_i splits 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 $s(k) > \Delta$.
3. Note that the number of bins needed to optimally pack the instance is not necessarily *monotone* in the number of fragmented items. Indeed, due to the headers which are added to the fragments, increasing the number of fragmented items may increase the number of bins in an optimal solution. Consequently, we cannot use binary search for guessing the number of fragmented items, and therefore this step increases the overall running time by factor $O(n)$.

Again, there is no need for partitioning the input into small and large items, as all of the items are large.

The modified scheme proceeds as follows.

(i) If $\Delta \leq \varepsilon/(1+\varepsilon)$ pack the items greedily as described in Theorem 7.1.1, else go to (ii).

(ii) Add Δ to the size of each item.

(iii) Guess c, the number of fragmentations in an optimal solution.

(iv) Guess $d = OPT(I)$, the number of bins used by an optimal packing.

(v) Round up the size of each large item to the nearest integral multiple of ε^2.

(vi) Define the *configuration matrix*, A, and the *(modified) fragmentation matrix*, B, each having $1/\varepsilon^2$ rows.

(vii) Solve within an additive factor of 1 a linear program for packing the large items using at least d bins and at least c fragmentations.

(viii) Round the solution of the linear program and pack accordingly the items in at most $OPT(I) + k$ bins of size $1 + \varepsilon + \varepsilon^2$, where $k \leq 1/\varepsilon^2 + 3$.

The analysis of the scheme is similar to analysis given in Chapter 5.

Theorem 7.4.1 *BP-SIF admits a dual AFPTAS.*

7.5 An AFPTAS for BP-SIF

As before, we may assume that all of the items are large and we refer to the actual sizes as the input. The major change in the scheme is in the formulation of the LP that we solve.

$$
\begin{aligned}
(LP') \qquad \text{minimize} \qquad & \sum_{i=1}^{q} x_i + \sum_{k=1}^{s} f_k y_k \\
\textit{subject to}: \qquad & \sum_{i=1}^{q} A_{ji} x_i + \sum_{k=1}^{s} {B'}_{jk} y_k \geq n_j \qquad \text{for } j = 1, \ldots, m \\
& x_i \geq 0 \qquad \text{for } i = 1, \ldots, q \\
& y_k \geq 0 \qquad \text{for } k = 1, \ldots, s
\end{aligned}
$$

The 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. We may assume, as before, that the capacity of each oversized bin is at most $1/\varepsilon$.

The following is an overview of the modified schem.

(i) If $\Delta \leq \varepsilon/(1+\varepsilon)$ pack the items greedily as described in Theorem 7.1.1, else go to (ii).

(ii) Add Δ to the size of each item.

(iii) Transform the instance to one in which the number of distinct item sizes is fixed (using geometric grouping).

(iv) Define the configuration matrices A and B' for packing items in unit-sized bins and in larger bins, respectively.

(v) Efficiently obtain a basic solution for LP'.

(vi) Pack (and if needed fragment) the items according to the basic solution of LP'.

Hence, we have

Theorem 7.5.1 *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*

$O(n\log\varepsilon^{-1}+\varepsilon^{-3}\log^2\varepsilon^{-1}\cdot min(\varepsilon^{-4}\log^2\varepsilon^{-1},\varepsilon^{-2}\log\varepsilon^{-1}M))$, *Where* M *is the longest binary representation of any input element.*

Chapter 8

Discussion

8.1 Summary

We studied two variants of BPF and developed for each of the problems a dual PTAS, APTAS, a dual AFPTAS and a linear time AFPTAS. All of our schemes relied on the existence of an optimal solution that has a *primitive* structure. In the AFPTAS, we took this property one step further and defined a new problem, that is equivalent to BP-SPF and BP-SIF, and which does not require item fragmentation. This enabled the development of a very fast (linear time) and quite simple scheme for both problems. As a special case, this scheme yields AFPTAS for the classical bin packing, whose running time improves the best known running time for this problem.

Several interesting techniques are developed in the thesis, including *oblivious shifting*, which allows to apply shifting to *unknown* item sizes, and a technique for converting a class of mixed covering/packing linear programs into pure covering LP without introducing negative coefficients in the process.

8.2 Future Work

We list below several interesting avenues for further research.

- Efficient online or semi online algorithms for BPF. This thesis focused on the offline version of the problems. Menakerman and Rom [20] and Naaman and Rom [24] gave analysis of simple online algorithms. It would be interesting to improve the

approximation ratios of these algorithms, by applying more sophisticated packing rules, especially, in deciding how to fragment items.

- Other BPF variants, e.g., variants that set tradeoff between fragmentation and the number of bins, different cost per fragmentation/item, or BPF problems with variable-sized bins.

- Can known techniques for solving multidimensional bin packing be applied to multi-dimensional bin packing with item fragmentation?

- Algorithms that guarantee an approximation of $OPT(I)+k$, where k is some constant. The existence of such algorithms is a long standing open problem already for bin packing.

- Applying the techniques presented in this thesis for other packing problems. In particular, Murgolo [23] developed an AFPTAS for variable-sized bin packing, that is based on the scheme of Karmarkar and Karp [15] for bin packing. We expect that applying the techniques in this thesis, for bounding the distinct number of bin sizes and to obtain a basic solution for the corresponding covering program, will yield a significant improvement in the running time of the scheme of [23].

Bibliography

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

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

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

[4] 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.

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

[6] F. Eisenbrand, Fast integer programming in fixed dimension. In *Proc. of ESA*, 2003.

[7] 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.

[8] 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.

[9] D. K. Friesen and M. A. Langston. Variable sized bin packing. *SIAM J. on Computing*, 15:222–230, 1986.

[10] 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.

[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.

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

[15] 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.

[16] H. Kellerer and U. Pferschy. A New Fully Polynomial Approximation Scheme for the Knapsack Problem. *APROX'98*, 1998, pp. 123–144.

[17] R. Khandekar. Lagrangian relaxation based algorithms for convex programming problems. *PhD thesis, Indian Institute of Technology Delhi*, 2004. In http:// www.cse.iitd.ernet.in/∼rohitk.

[18] 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.

[19] 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.

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

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

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

[23] F.D. Murgolo. An Efficient Approximation Scheme for Variable-Sized Bin Packing. *SIAM J. Comput.* 16(1), 149–161, 1987.

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

[25] S.A. Plotkin, D.B. Shmoys, Eva Tardos. Fast Approximation Algorithms for Fractional Packing and Covering Problems. In *Proc. of FOCS*, 1995.

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

[27] H. Shachnai, T.Tamir and O. Yehezkely. Approximation Schemes for Packing with Item Fragmentation. To appear in *Theory of Computing Systems*. Preliminary version in Proc. of WAOA 2005.

[28] F. Sourd, Preemptive scheduling with position costs. To appear in *Algorithmic Operations Research*.

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

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

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

[32] N. E. Young. Sequential and Parallel Algorithms for Mixed Packing and Covering. In I *Proc. of FOCS*, 538–546, 2001.

סכימות קרוב לבעיות אריזה עם פצולים

עומר יחזקאלי

סכימות קרוב לבעיות אריזה עם פצולים

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת התואר מגיסטר למדעים במדעי המחשב

עומר יחזקאלי

הוגש לסנט הטכניון – מכון טכנולוגי לישראל

חשון התשס״ז חיפה נובמבר 2006

המחקר נעשה בהנחיית פרופסור הדס שכנאי בפקולטה למדעי המחשב.

אני מודה לטכניון ולקרן ע״ש סם כהן וינדהוק על התמיכה הכספית הנדיבה בהשתלמותי.

תוכן העניינים

תקציר

בבעית אריזת התאים (*BP - bin packing*) נתון אוסף פריטים $a_1, a_2, \ldots, a_n$ ולכל פריט הגודל $s(a_i) \in (0,1]$. המטרה היא לארוז את *כל* הפריטים בתאים (bins) בעלי קיבולת אחסון 1. בעית ה- BP היא בעית NP-hard ידועה.

עבודה זו עוסקת בשני וריאנטים של BP שבהם ניתן לפצל את הפריטים, מתוך מטרה לצמצם את מספר התאים הנדרשים לאריזת הקלט. אוסף זה של בעיות נקרא אריזת תאים עם פצולים (*BPF - bin packing with fragmentation*).

לכאורה, כאשר ניתן לפצל את הפריטים, הופכת הבעיה לטריויאלית, מאחר וניתן לארוז את הפריטים (בסדר שרירותי כלשהו), וכאשר לא ניתן לארוז את הפריט הנוכחי, מאחר ואין לו די מקום בתא הנוכחי, נפצל את הפריט כך שהתא יתמלא לחלוטין, ונמשיך בתהליך האריזה על-ידי שימוש בתא חדש. ואכן, אם זהו המצב, הבעיה פשוטה.

אולם, אם מוטלת מגבלה על אופן בצוע הפצולים, הבעיה נשארת קשה, כמו בשתי הבעיות המטופלות בעבודה זו:

1. **בעית אריזת התאים עם פצולים המשמרים את נפח הקלט** (*BP-SPF - Bin Packing with Size Preserving Fragmentation*). בבעיה זו מוטלת מגבלה על מספר הפצולים המקסימלי שניתן לבצע. כאשר מספר הפצולים המותרים שווה לאפס, מתקבלת בעיית BP הקלאסית, ואילו כאשר מספר הפיצולים המותר גדול/שווה לנפח הכולל של הפריטים, הבעיה חוזרת להיות טריויאלית, כפי שהוסבר לעיל.
2. **בעית אריזת תאים עם פיצולים המגדילים את נפח הקלט** (*BP-SIF – Bin Packing with Size Increasing Fragmentation*). בבעיה זו נוספת לכל פריט מעטפת (Header ו/או Footer) שגודלה $0 < \Delta < 1$. לפיכך, כאשר אנו מפצלים פריט לשניים, שני תתי הפריטים הנוצרים יעטפו כל-אחד במעטפת נפרדת, ובכך יגדל הנפח הכולל שעלינו לארוז.

שתי בעיות אלו מופיעות במגוון רחב של יישומים, בכללם:

- **פרוטוקולי תקשורת:** בפרוטוקול DOCSIS, לדוגמה, לכל הודעה נוסף header בגודל קבוע. ניתן לפצל הודעות (ולכל אחד מהחלקים נוסף header) והתשדורת עצמה מבוצעת ע"י שידור של מסגרות בעלות גודל קבוע. כאשר המטרה היא למזער את משך הזמן הנדרש להעברת אוסף נתון של הודעות, מתקבל קלט לבעית BP-SIF.

- **בעיות תזמון:** בבעיות תזמון שבהן נתון זמן יעד לבצוע אוסף של ג'ובים, וכמו כן ניתן לבצע הפקעות (preemption), כאשר המטרה היא לצמצם את מספר המעבדים הנחוצים על מנת לסיים עד זמן היעד אנו מקבלים:
  א. במקרה שבו התחלת או המשך הבצוע של ג'וב דורשת איתחול, נקבל קלט לבעית BP-SIF.
  ב. במקרה שבו התחלה או המשך הבצוע של ג'וב כרוכה בעלות חיצונית, ועלות התזמון מוגבלת לערך נתון, מתקבל קלט לבעית BP-SPF.

- **תכנון מעגלי VLSI:** פעולתה של יחידה לוגית מתחילה בהעתקת ערכים לכניסות הקלט של היחידה. ערכים אלה יוכלו להיות ערכי פלט של יחידה לוגית אחרת. באופן כללי, יתכן שיש להעתיק ערך פלט אחד של יחידה לוגית מסוימת למספר רב של כניסות קלט של יחידה לוגית אחרת. מאחר ומספר כתובות הזכרון אליהן ניתן לגשת בסבב עבודה בודד הוא חסום, וברצוננו למזער את משך זמן האתחול, מתקבל קלט לבעית BP-SIF.

לדוגמה, אם עלינו להעתיק את הערך של הפלט A לחמש כניסות הקלט: B,C,D,E,F וניתן לגשת ל-5 כתובות זיכרון במהלך סבב העתקה בודד, יידרש פיצול של תהליך ההעתקה (פיצול אפשרי: B=C=D=E=A; F=A;) וידרשו שני סבבים לבצוע המשימה. במידה שיש הרבה פלטים שכל אחד מהם יש להעתיק להרבה כניסות קלט, נקבל קלט ל- BP-SIF שבו מספר הכתובות אליהן ניתן לגשת בסבב הוא גודל התאים, גודל ה- Header הוא 1, וגדלי הקבוצות של כניסות הקלט (כניסות קלט שמאותחלות לאותו ערך שייכות לאותה קבוצה של כניסות קלט) הם גדלי הפריטים.

- **בעיות אריזה גמישות:** לעיתים עלות האריזה של הפריטים היא גבוהה, והפריטים עצמם ניתנים לפיצול, למשל, בעית שינוע חומרי בנין (כגון חול) הנתונים בשקים. חסכון בעלות הובלה של משאית אחת הוא מהותי, ואילו את שקי החול ניתן לפצל בקלות. מאחר והמשקל הכולל שמשאית רשאית להוביל הוא חסום אנו מקבלים קלט לבעית BP-SPF (בהנחה שמשך הזמן שאנו מוכנים להשקיע בפיצול השקים חסום).

בעבודה זו אנו מוכיחים כי שתי הבעיות (BP-SPF ו- BP-SIF) קשות לקרוב, ולא תתכן עבורם סכימת קרוב פולינומיאלית (Polynomial Time Approximation Scheme – PTAS), בהנחה ש- $P \neq NP$. בפרט, אנו מראים כי לא יתכן אלגוריתם קרוב לבעיות הנ"ל המבטיח קרוב של $(1+\epsilon)$, כאשר $0<\epsilon<0.5$.

עם זאת, בדומה לבעיית ה- BP, אנו מראים כי הבעיות הנ"ל ניתנות לקרוב דואלי ואסימפטוטי. אנו עושים זאת ע"י פתוח הסכימות הבאות:

1. סכימת קירוב פולינומיאלית דואלית (*Dual PTAS*) אשר בהנתן קלט I לאחת משתי הבעיות אורזת את כל הפריטים ב- OPT(I) תאים בגודל $1+\epsilon$ ((OPT(I) הנו מספר התאים הדרוש בפתרון אופטימלי).
2. סכימת קרוב פולינומיאלית אסימפטוטית (*APTAS - Asymptotic PTAS*) אשר אורזת את הפריטים ב- $(1+\epsilon)OPT(I)+1$ תאים.
3. סכימת קרוב אסימפטוטית, פולינומיאלית מלאה ודואלית (*Dual AFPTAS - Dual Asymptotic Fully PTAS*), אשר אורזת את הפריטים לתוך $OPT(I)+k$ תאים בגודל $(1+\epsilon)$, כאשר $k \leq 4/\epsilon^2+3$. (סכימת קרוב נקראת מלאה כאשר זמן הריצה שלה פולינומיאלי בגודל הקלט ובפרמטר הקרוב, כלומר פולינומיאלי ב- $1/\epsilon$).
4. סכימת קרוב אסימפטוטית ופולינומיאלית מלאה (*AFPTAS*) בעלת סיבוכיות זמן ריצה לינארית בגודל הקלט.

בפיתוח כל הסכימות הנ"ל אנו נסמכים על העובדה (המוכחת בעבודה) שלכל קלט לבעיות BP-SPF ו- BP-SIF ישנו פתרון אופטימלי שהמבנה שלו "פרימיטיבי". עובדה זו מאפשרת לנו לצמצם באופן ניכר את מרחב הפתרונות מתוכם עלינו לחפש את הפתרון האופטימלי.

בפתוח סכימת הקרוב הרביעית (AFPTAS) אנו משתמשים באספקט נוסף של קיום פתרון אופטימלי בעל מבנה "פרימיטיבי": אנו מגדירים בעיה חדשה, אותה אנו מכנים בעיית כיסוי פריטים *(item cover problem)*, אשר אין בה פיצולי פריטים, אך פתרונה שקול לחלוטין לפתרון הבעיות BP-SPF ו- BP-SIF. הגדרה זו של בעית כיסוי הפריטים מאפשרת פיתוח סכימת קרוב מהירה מאוד, אשר סיבוכיות זמן הריצה שלה היא לינארית בגודל הקלט.

טיפול ישיר בנושא הפיצולים הינו בעיתי (לצורך השגת סכימה פולינומיאלית מלאה), מאחר וגם בפתרונות בעלי מבנה פרימיטיבי מספר אפשרויות הפיצול הוא אקספוננציאלי ב- $1/\epsilon$, ועיגול הגדלים של תתי-הפריטים (אשר אינם נתונים מראש, והם נגזרים מגדלי הפריטים שאינם מתפצלים בקלט) עלול לגרום ל"גלישה" מהתאים. לפיכך, פתרון הבעיות ע"י פתרון בעיה שאינה מצריכה פיצולים מקל באופן מהותי על השגת סכימת קרוב פולינומיאלית מלאה.

לצורך השוואת סיבוכיות הזמן של הסכימה הפולינומיאלית המלאה עם תוצאות קודמות, אנו מנתחים את סיבוכיות הזמן שלה בפתרון בעית ה- BP-SPF כאשר תקציב הפצולים הוא אפס. במקרה זה, מתקבלת בעית BP הקלאסית. מניתוח זמן הריצה, עולה כי הסכימה הנ"ל משפרת את זמן הריצה הטוב ביותר הידוע בפתרון בעית BP.

טכניקה מענינת שפותחה בעבודה זו היא *Oblivious Shifting*, אשר מצליחה להזיח גדלים *לא ידועים* של תתי פריטים, כך שהתוצאה הסופית היא מספר חסום של סוגי גדלים ומספר חסום של תתי פריטים (הגדולים ביותר) שצריך לארוז בשנית בתאים חדשים. משימה זו, הנראית בלתי אפשרית בתחילה, מתבררת כאפשרית כאשר נתונים חסמים על הגודל הכולל של חלקים הנארזים יחדיו באותו תא, וכן גודלם המקורי של הפריטים לפני פיצולם.

זאת ועוד, אנו מדגימים בעבודה זו טכניקה להמרת מחלקה של תוכניות לינאריות מעורבות, הכוללות אילוצי אריזה ואילוצי כיסוי, לתוכניות הכוללות אילוצי כיסוי בלבד. המיוחד בטכניקה זו הוא שאינה גורמת להוספת מקדמים שליליים לאילוצים, כפי שקורה בהמרות סטנדרטיות של אילוצים. תכונה זו עשויה לאפשר שימוש במספר סכימות קרוב מודרניות לתוכניות לינאריות אשר דורשות מקדמים אי-שליליים כתנאי בסיסי להפעלתן.