A Robust APTAS for the Classical Bin Packing Problem

Leah Epstein[1] and Asaf Levin[2]

[1] Department of Mathematics, University of Haifa, 31905 Haifa, Israel.
Email: lea@math.haifa.ac.il
[2] Department of Statistics, The Hebrew University, Jerusalem, Israel.
levinas@mscc.huji.ac.il.

Abstract. Bin packing is a well studied problem which has many applications. In this paper we design a robust APTAS for the problem. The robust APTAS receives a single input item to be added to the packing at each step. It maintains an approximate solution throughout this process, by slightly adjusting the solution for each new item. At each step, the total size of items which may migrate between bins must be bounded by a constant factor times the size of the new item. We show that such a property cannot be maintained with respect to optimal solutions.

1 Introduction

Consider the classical online bin packing problem where items arrive one by one and are assigned irrevocably to bins. Items have sizes bounded by 1 and are assigned to bins of size 1 so as to minimize the number of bins used. The associated offline problem assumes that the complete input is given in advance.

We follow [12] and allow the "online" algorithm to change the assignment of items to bins whenever a new item arrives, subject to the constraint that the total size of the moved items is bounded by β times the size of the arriving item. The value β is called the *Migration Factor* of the algorithm. We call algorithms that solve an offline problem in the traditional way *static*, whereas algorithms that receive the input items one by one, assign them upon arrival, and can do some amount of re-packing using a constant migration factor are called *dynamic* or *robust*. An example we introduce later shows that an optimal solution for bin packing cannot be maintained using a dynamic (exponential) algorithm. Consequently, we focus on polynomial-time approximation algorithms.

In our point of view, the main advantage in obtaining an APTAS for the classical bin packing problem with a bounded migration factor is that such type of schemes possess a structure. Hence we are able to gain insights into the structure of the solution even though it results from exhaustive enumeration of a large amount of information.

Sanders, Sivadasan and Skutella [12] studied the generalization of the online scheduling problem where jobs that arrive one by one are assigned to identical parallel machines with the objective of minimizing the makespan. In their generalization they allow the current assignment to be changed whenever a new job

arrives, subject to the constraint that the total size of moved jobs is bounded by β times the size of the arriving job. They obtained a dynamic polynomial time approximation scheme for this problem extending an earlier polynomial time approximation scheme of Hochbaum and Shmoys [7] for the static problem. They noted that this result is of particular importance if considered in the context of sensitivity analysis. While a newly arriving job may force a complete change of the entire structure of an optimal schedule, only very limited local changes suffice to preserve near-optimal solutions.

For an input X of the bin packing problem we denote by $OPT(X)$ the minimal number of bins needed to pack the items of X, and let $SIZE(X)$ denote the sum of all sizes of items. Clearly, $OPT(X) \geq SIZE(X)$. For an algorithm $\mathcal{B}$ we denote by $\mathcal{B}(X)$ the number of bins used by $\mathcal{B}$.

It is known that no approximation algorithm for the classical bin packing problem can have a cost within a constant factor r of the minimum number of required bins for $r < \frac{3}{2}$ unless $\mathcal{P} = \mathcal{NP}$. This leads to the usage of the standard quality measure for the performance of bin packing algorithms which is the *asymptotic approximation ratio* or *asymptotic performance guarantee*. The asymptotic approximation ratio for an algorithm $\mathbf{A}$ is defined to be

$$\mathcal{R}(\mathbf{A}) = \limsup_{n\to\infty} \sup_{X} \left\{ \frac{\mathbf{A}(X)}{OPT(X)} \middle| OPT(X) = n \right\}.$$

The natural question, which was whether this measure allows to find an approximation scheme for bin packing, was answered affirmatively by Fernandez de la Vega and Lueker [3]. They designed an algorithm whose output never exceeds $(1+\varepsilon)OPT(I) + g(\varepsilon)$ bins for an input I and a given $\varepsilon > 0$. The running time was linear in n, but depended exponentially on ε, and such a class of algorithms is considered to be an APTAS (Asymptotic Polynomial Time Approximation Scheme). The function $g(\varepsilon)$ depends only on ε and grows with $\frac{1}{\varepsilon}$.

Two later modifications simplified and improved this seminal result. The first modification allows to replace the function $g(\varepsilon)$ by 1 (i.e. one additional bin instead of some function of ε), see [17], Chapter 9. The second one by Karmarkar and Karp [10] allows to develop an AFPTAS (Asymptotic Fully Polynomial Time Approximation Scheme). This means that using a similar (but much more complex) algorithm, it is possible to achieve a running time which depends on $\frac{1}{\varepsilon}$ polynomially. The dependence on n is much worse than linear, and is not better than $\Theta(n^8)$. In this case the additive term remains $g(\varepsilon)$. Karmarkar and Karp [10] also designed an algorithm which uses at most $OPT(I) + \log^2[OPT(I)]$ bins for an input I.

Related work. The classical online problem was studied in many papers, see the survey papers of [2, 1]. It was first introduced and investigated by Ullman [15]. The currently best results are an algorithm of asymptotic performance ratio 1.58889 given by Seiden [14] and a lower bound of 1.5401 [16]. From this lower bound we can deduce that in order to maintain a solution which is very close to optimal, the algorithm cannot be online in the usual sense. Several attempts were

made to give a semi-online model which allows a small amount of modifications to the solution produced by the algorithm. We next review these attempts.

Gambosi, Postiglione and Talamo [5, 6] introduced a model where a constant number of items (or small items grouped together) can be moved after each arrival of an item. They presented two algorithms. The first moves at most three items on each arrival and has the performance guarantee $\frac{3}{2} = 1.5$. The second algorithm moves at most seven items on each arrival and the performance guarantee $\frac{4}{3} \approx 1.33333$. The running times of these two algorithms are $\Theta(n)$ and $\Theta(n \log n)$ respectively, where n is the number of items.

Ivkovic and Lloyd [9] gave an algorithm which uses $O(\log n)$ re-packing moves (these moves are again of a single item or a set of grouped small items). This algorithm is designed to deal with departures of items as well as arrivals, and has performance guarantee $\frac{5}{4}$. Ivkovic and Lloyd [9, 8] considered an amortized analysis as well, and show that for every $\varepsilon > 0$, the performance guarantee $1 + \varepsilon$ can be maintained, with $O(\log n)$ amortized number of re-packing moves if ε is seen as a constant, and with $O(\log^2 n)$ re-packing moves if the running time must be polynomial in $\frac{1}{\varepsilon}$. However, the amortized notion here refers to a situation that for most new items no re-packing at all is done, whereas for some arrivals the whole input is re-packed.

Galambos and Woeginger [4] adapted the notion of bounded space online algorithms (see [11]), where an algorithm may have a constant number of active bins, and bins that are no longer active, cannot be activated. They allow complete re-packing of the active bins. It turned out that the same lower bound as for the original (bounded space) problem holds for this problem as well, and re-packing only allowed to obtain the exact best possible competitive ratio having three active bins, instead of in the limit.

Outline. We review the adaptation of the algorithm of Fernandez de la Vega and Lueker [3], as it appears in [17], in Section 2. We then state some further helpful adaptations that can be made to the static algorithm. In Section 3 we describe our dynamic APTAS and prove its correctness. This algorithm uses many ideas from [3], however the adaptation into a dynamic APTAS requires careful changes to the scheme. We show that the number of bins used by our APTAS never exceeds $(1 + \varepsilon)OPT(X) + 1$ where X is the list of items that has been considered so far. The running time is $O(n \log n)$ where n is the number of items, since the amount of work done upon arrival of an item is a function of ε times $\log n$. In the full version we show an example in which there is no optimal solution that can be maintained with a constant migration factor.

2 Preliminaries

We review a simple version of the very first asymptotic polynomial time approximation scheme. This is the algorithm of Fernandez de la Vega and Lueker [3]. The algorithm is *static*, i.e., it considers the complete set of items in order to compute the approximate solution. Later in this section we adapt it and present

another version of it, which is still static. This new version is used in our dynamic APTAS that is presented in the next section.

We are given a value $0 < \varepsilon < 1$ such that the asymptotic performance guarantee should be at most $1 + \varepsilon$, and $\frac{1}{\varepsilon}$ is an integer. Consider an input I for the bin packing problem. We define an item to be *small*, if its size is smaller than $\frac{\varepsilon}{2}$. Other items are *large*.

Algorithm FL [3]:

1. Items are partitioned into two sets according to their size. The multiset of large items is denoted L and the multiset of small items is denoted T. We have $I = L \cup T$.
2. A linear grouping is performed to the large items. Let n be the number of large items in the input ($n = |L|$), and let $a_1 \geq \ldots \geq a_n$ be these items. Let $m = \frac{2}{\varepsilon^2}$. We partition the sorted set of large items into m consecutive sequences S_j ($j = 1, \ldots, m$) of $k = \lceil \frac{n}{m} \rceil = \lceil \frac{n\varepsilon^2}{2} \rceil$ items each (to make the last sequence be of the same cardinality, we define $a_i = 0$ for $i > n$). I.e., $S_j = \{a_{(j-1)k+1}, \ldots, a_{(j-1)k+k}\}$ for $j = 1, 2, \ldots, m$. For $j \geq 2$, we define a modified sequence $\hat{S}_j$ which is based on the sequence S_j as follows. $\hat{S}_j$ is a multiset which contains exactly k items of size $a_{(j-1)k+1}$, i.e., all items are rounded up to the size of the largest element of S_j. The set S_1 is not rounded and therefore $\hat{S}_1 = S_1$. Let L' be the union of all multisets $\hat{S}_j$ and let $L'' = \bigcup_{j=2}^{m} \hat{S}_j$.
3. The input L'' is solved optimally.
4. A packing of the complete input is obtained by first replacing the items of $\hat{S}_j$ in the packing by items of S_j (the items of S_j are never larger than the items of $\hat{S}_j$, and so the resulting packing is feasible), and second, using k bins to pack each item of S_1 in a separate bin. Last, the small items are added to the packed bins (with the original items without rounding) using Any Fit Algorithm. Additional bins can be opened for small items if necessary. Step 3 can be executed in polynomial time by solving an integer programming in a fixed dimension.

Lemma 1. *Algorithm FL is a polynomial time algorithm.*

We next analyze the performance guarantee of Algorithm FL. For two multisets A, B, we say that A is *dominated* by B and denote $A \leq B$ if there exists an injection $h : A \to B$ such that for all $a \in A$, $h(a) \geq a$.

Lemma 2. *If A and B are multisets such that $A \leq B$, then $OPT(A) \leq OPT(B)$.*

Theorem 1. *Algorithm FL is an APTAS.*

Algorithm Revised FL: We now design a new static adaptation Algorithm FL which is later generalized into a dynamic APTAS. We modify only Step 2 as follows:

- The multisets S_j are defined similarly to before, with the following changes. The multisets do not need to have the same size, but their cardinalities need to be monotonically non-increasing. I.e., for all j, $|S_j| \geq |S_{j+1}|$. Moreover,

we require that if $n\varepsilon^2 \geq 8$, then $|S_1| \leq \lceil \frac{\varepsilon^2 n}{2} \rceil$ and $|S_m| \geq \frac{|S_1|}{4}$, and otherwise each set has a single element.
- The rounding is done as follows. Given a multiset S_j which consists of elements $c_1 \geq \ldots \geq c_k$, the elements are rounded up into two values. Let $1 < s \leq k$, then all elements $c_1, \ldots, c_{s-1}$ are rounded into c_1, and the elements $c_s, \ldots, c_k$ are rounded into c_t for some $2 \leq t \leq s$.

The proof of Theorem 1 extends easily to this adaptation as well. Specifically, the amount of distinct sizes in the rounded instance is constant (which depends on ε) and so is the number of patterns. The mapping is defined similarly, and the set S_1 still satisfies $|S_1| \leq \lceil \frac{\varepsilon^2 n}{2} \rceil$ so it is small enough to be packed into separate bins. If the small items which are are added using Any Fit cause the usage of additional bins, the situation is exactly the same as before. Therefore, we establish the following theorem.

Theorem 2. *Algorithm Revised FL is an APTAS.*

In the sequel we show how to maintain the input grouped as required by Algorithm Revised FL. We also show that the difference between packing of two subsequent steps is small enough that it can be achieved by using a constant (as a function of ε) migration factor as in [12].

3 APTAS with $f(\frac{1}{\varepsilon})$ Migration Factor

In this Section we describe our dynamic APTAS for bin packing with the additional property that the migration factor of the scheme is $f(\frac{1}{\varepsilon})$ (i.e., it is a function f of the term $\frac{1}{\epsilon}$), and therefore a constant migration factor for fixed value of ε.

We use the following notations. The number of large items among the first t items is denoted $n(t)$. The size of the ith arriving item is b_i. The value m is defined as in the previous section $m = \frac{2}{\varepsilon^2}$. We denote by $OPT(t)$ the number of bins used by an optimal solution for the first t items. We assume that after the first t items, we have a feasible solution that uses at most $(1+\varepsilon)OPT(t)+1$ bins and show how to maintain such a solution after the arrival of a new item of index $t+1$. Later on we describe several structural properties that our solution satisfies, and show how to maintain these properties as well. These structural properties help us to establish the desired migration factor.

Similarly to Algorithm Revised FL we treat small items and large items differently. Recall that an item is small, if its size is smaller than $\frac{\varepsilon}{2}$. When a small item arrives, we use Any Fit Algorithm to find it a suitable bin.

It remains to consider the case where the $t+1$-th item is a large item, i.e., $b_{t+1} \geq \frac{\varepsilon}{2}$. We keep the following structural properties throughout the algorithm.

1. If $n(t) \geq 4m+1$ then the sorted list of large items which arrived so far is partitioned into $M(t)$ consecutive sequences $S_1(t), \ldots, S_{M(t)}(t)$, where $4m+1 \leq M(t) \leq 8m+1$ such that $|S_1(t)| \geq |S_2(t)| = |S_3(t)| = \cdots = |S_{M(t)}(t)|$.

Otherwise, if $n(t) \leq 4m$ then the sorted list of large items which arrived so far is partitioned into $n(t)$ consecutive sequences each of them has a single large element $S_1(t), \ldots, S_{n(t)}(t)$.
2. Denote $K(t) = |S_{M(t)}(t)|$, then $|S_1(t)| \leq 4K(t)$.
3. There are two special subsets of $S_1(t)$ denoted by $S'_1(t)$ and $S''_1(t)$ (these sets might be empty at some times). Each of these two sets contains at most $K(t)$ items. The sets are special in the sense that we treat them as separate sets while rounding, and in the rounded up instance $S'_1(t)$ and $S''_1(t)$ are rounded. However in the analysis the cost of the solution is bounded allowing each element of $S_1(t)$ to be packed in its own bin.

The time index t can be omitted from the notation of a set or a parameter if the time it belongs to is clear from the context. Therefore, when e.g. we discuss the set S_1 this is the set $S_1(t)$ that is associated with the discussed time. The size of the set S_1 is defined according to algorithm Revised FL. We have $n \geq MK > 4mK$, and therefore $S_1 \leq 4K < \frac{n}{m} = \frac{n\varepsilon^2}{2}$.

Note that for $n \leq 4m$ each element has its own list, and therefore we solve optimally the instance of the large items excluding the largest item.

We turn now to discuss *steps*, where each step is an arrival of a large item. We partition the steps into three types, which are, *regular steps*, *creation steps* and *union steps*. An *insertion of a new large item* operation takes place in all types of steps. A *creation of new sets* operation takes place only in creation steps. A *union of sets* operation takes place only in union steps.

We also maintain the following property. During regular steps or creation steps, no set is rounded to a pair of values but for each set S_j ($j \geq 2$), the items of S_j are rounded up to the largest size of any element of S_j. However, during union steps, each set is rounded up to a pair of values (as in Algorithm Revised FL).

Lemma 3. *The arrival of a small item and allocating it according to Any Fit Algorithm maintains the structural properties.*

We next define a series of operations on the sequences so the properties are maintained after a new item arrives, and the migration factor of the resulting solution is constant. When a new item arrives, we first apply the *insertion of a new large item* operation. Afterwards if the current step is a creation step, we apply the *creation of new sets* operation, whereas if the current step is a union step, we apply the *union of sets* operation.

When we bound the migration factor, note that changes to the allocation of items into bins are made only when large items arrive. Hence, the size of the arriving item of index $t+1$ is at least $\frac{\varepsilon}{2}$. Therefore, if we can prove that the allocation is changed only for a set of items that we allocated to a fixed number of bins (their number is a function of ε) then we get a constant migration factor throughout the algorithm.

Insertion of a new large item. When a large item arrives then if $n \leq 4m+1$ we add a new list that contains the new item as its unique element. Note that in this case the resulting set of lists satisfies the structural properties.

Otherwise (i.e., $n \geq 4m + 2$) we first compute the list to which it belongs, and add it there. The list to which it belongs, S_j, is defined as follows. If the new item is larger than an existing item in S_1, then this list is S_1. Otherwise, we find the set S_{j+1} of smallest index such that the new item is larger than all its elements, and the list to which the item belongs is S_j. We *move up* items from S_i to S_{i-1} for all $2 \leq i \leq j$, this operations is defined as follows. We move the largest item of S_i to S_{i-1} and afterwards we change the value which the size of the items of S_i is rounded up to, into the size of the new largest item of S_i. When we consider the effect this operation has on the feasibility of the integer program, we can see that the right hand side does not change (the size of S_r is not affected for all values of r such that $2 \leq r \leq M$), however new patterns arise as the size of the rounded up instance is smaller (and so we can pack more items to a bin in some cases). The additional patterns mean new columns of the feasibility constraint matrix. Note that adding the new large item into its list takes $O(\log n)$ time (as we need to maintain sorted lists of the large elements).

Theorem 3. *[Corollary 17.2a, [13]] Let A be an integral $m \times d$ matrix such that each sub-determinant of A is at most Δ in absolute value, let $\hat{u}$ and u' be column m-vectors, and let v be a row d-vector. Suppose* $\max\{vx|Ax \leq \hat{u}; x \text{ is integral}\}$ *and* $\max\{vx|Ax \leq u'; x \text{ is integral}\}$ *are finite. Then, for each optimum solution y of the first maximum there exists an optimum solution y' of the second maximum such that $||y - y'||_\infty \leq d\Delta\,(||\hat{u} - u'||_\infty + 2)$.*

Lemma 4. *Let A be the constraint matrix of the feasibility integer program. Let d be the number of columns of A and let Δ be the maximum value in absolute value of a sub-determinant of A. Then, throughout the algorithm $d \cdot \Delta$ is bounded by a constant (for a fixed value of ε).*

The proof of the following lemma is similar to the analysis of [12].

Lemma 5. *Assume that before the arrival of the current large item, there is a feasible solution y to the feasibility integer program of the rounded up instance L''. After we apply an* insertion of a new large item *operation, if the feasibility integer program is feasible then there is a solution to it y' such that it is suffices to re-pack the items that reside in a constant number of bins.*

Proof. Denote by $Ay = u$ the constraints of the feasibility integer program of the rounded up instance. The matrix A is the constraint matrix and u is the right hand side. Note that the columns of A correspond to patterns and the rows correspond to different sizes of items. The constraint that corresponds to an item size a has the following meaning. The amount of the items that we allocate along all possible patterns of items with size a is exactly the number of items in the rounded up instance with size a.

We first assume that $n(t) \geq 4m+1$. Now consider the change in the constraint matrix A when a new large item of size b_{t+1} arrives. Let A' denote the modified A, which is the feasibility constraint matrix for all the items in L'' and the one extra new element of S_1. First, the cardinality of S_1 increases by 1, (at the end

of the move up operation), and this does not change the constraint matrix as we do not have a constraint associated with S_1. Next, we consider the decrease in the size of rounded up items. Note that all the patterns that were feasible in the previous stage clearly remain feasible (given a set of items that can be packed in a single bin, decreasing the size of some of the items still allows to pack them into a single bin). Therefore, the matrix A of the previous stage is a sub-matrix of the matrix after we apply the insertion of a new large item operation. The difference is a possible addition of columns (that correspond to patterns that were infeasible before we decrease the size of some items and before we add the new item, however these patterns are now feasible).

To bound the change in u, denote the new right hand side by u'. The set S_1 is not represented in A, therefore there is no change and $u' = u$. Let $\hat{u} = u$. Then, in the case where $n(t) \geq 4m + 1$ the change in the right hand side is bounded by a constant, i.e., $||u' - \hat{u}||_\infty = 0$.

Otherwise, $n(t) \leq 4m$ and the feasibility integer program has a new row that corresponds to the new large item and whose right hand side value is 1. Note that all the patterns that were feasible in the previous stage clearly remain feasible. Therefore, the matrix A of the previous stage is a sub-matrix of the matrix after we apply the insertion of a new large item operation. The difference is a possible addition of columns that pack the new item as well, and a new row that corresponds to the new item. To bound the change in u, denote the new right hand side by u'. Let $\hat{u}$ be a right hand side equal to u' beside one entry that is in the component that correspond to the new row of A where it equals zero. In this case $||u' - \hat{u}||_\infty = 1$. In the remainder of this proof we do not distinguish between the case where $n \geq 4m + 1$ and the case where $n \leq 4m$.

We can extend y to a vector $\hat{y}$ that is a feasible solution of $A'\hat{y} = \hat{u}$. To do so, we define the entries of y that correspond to the new columns in A' compared to A to be zero. In the other components (whose columns exist in A) the value of $\hat{y}$ is exactly the value of y.

In order to prove the claim it is enough to show that there is a feasible solution y' such that $||\hat{y} - y'||_\infty$ is a constant (then we re-pack the items from the bins that correspond to the difference between $\hat{y}$ and y'). Recall that we assume that $Ay = \hat{u}$ is feasible integer program. Therefore, the assumptions of Theorem 3 are satisfied. Therefore, by Theorem 3, there is a feasible integer solution y' such that $||\hat{y} - y'||_\infty \leq d\Delta\left(||\hat{u} - u'||_\infty + 2\right)$. We would like to bound by a constant the right hand side of the last inequality.

By Lemma 4, d and Δ are bounded by a constant. We have already bounded $||\hat{u} - u'||_\infty$ by the constant 1 and this completes the proof. □

Therefore, the moving up operation causes constant migration. However in order to prove the performance guarantee of the algorithm we need to show how to maintain the structural properties. To do so, note that the only sets whose cardinality increases during the moving up operation is S_1, and therefore we need to show how to deal with cases where S_1 is too large.

Creation of new sets: After S_1 has exactly $3K$ items we start a new operation that we name *creation of new sets* that lasts for K steps (in each such

step a new large item arrives and we charge the operation done in the step to this new item). We consider the items $c_1 \geq c_2 \geq \cdots \geq c_{3K}$ of S_1. We create new sets S'_1 and S''_1 where eventually $S'_1 = \{c_{K+1}, \ldots, c_{2K}\}$ and $S''_1 = \{c_{2K+1}, \ldots, c_{3K}\}$. In each step we will have already rounded i items from S'_1 and from S''_1 to its target value and i is increased by 1 each step. So after i steps of this creation of new sets operation, the rounded up instance has i copies of the items c_{K+1} and c_{2K+1}. Then, the resulting instance can be solved in polynomial time where we put each item of S_1 that has not already rounded up to either c_{K+1} or to c_{2K+1} in a separate bin. Rounding up two items at each step results in a constant change in the right hand side of the feasibility integer programming and therefore increase the migration factor within an additive constant factor only as we prove in Lemma 6 below. At the end of K steps we declare the sets S'_1 and S''_1 as the new S_2 and S_3 and increasing M by two. Each of the new sets contains exactly K elements and the new S_1 contains exactly $2K$ elements, and therefore if $M(t) \leq 8m - 1$ we are done while keeping a constant migration factor. Otherwise, next time the creation new sets operation takes place we will violate the second structural property, and therefore we currently initiate the *union of sets* operation that lasts for K steps as well. Note that we never apply both the creation of new sets and the union of sets operations at the same step.

The moving up procedure during the insertion of a new large item operation will increase S_1 further, and this case we also apply this procedure to S'_1 and S''_1 and thus decreasing the value to which we round up the items that belong to S'_1 and S''_1. So in fact during the creation of new sets operation S_1 is partitioned into five sets $S^1_1, S'_1, S^2_1, S''_1, S^3_1$ such that the items in S^1_1 are the largest items of S_1, S^2_1 contains the items with size that is smaller than the items in S'_1 but they are larger than the items in S''_1, and S^3_1 contains the other items of S_1. Then, in each step we increase the size of S^1_1, S'_1 and S''_1 by one item each, whereas the size of S^2_1 and S^3_1 is decreased by one. This means that during the *move up* operation we will move up items also in these collection of the five subsets of S_1.

Lemma 6. *Assume that before we apply the creation of new sets operation and after we finish the insertion of a new large item operation, there is a feasible solution y' to the feasibility integer program of the rounded up instance L''. After we apply the creation new sets operation, if the feasibility integer program is feasible then there is a solution to it y'' such that it is enough to re-pack the items that reside in a constant number of bins.*

Union of sets: When the number of sets reaches $8m + 1$ we start the following operation that lasts for K steps (where again a step means an arrival of a large item). First we declare each pair of consecutive sets as a new set. That is for $2 \leq j \leq 4m + 1$ we let $S_j(t + 1)$ be $S_{2j-1}(t) \cup S_{2j-2}(t)$, but we still do not change the way the rounding is performed. So in the resulting partition into sets, each set has exactly $2K$ items, and we declare this the new value of K, i.e., $K(t+1) = 2K(t)$, and the number of sets now is $M = 4m+1$. However, each set is rounded to a pair of values, and this is something we will recover in the next steps. While the rounding of each set is to two values, denote by S'_j the items

that we round to the largest item of S_j and by $S''_j = S_j \setminus S'_j$. In each step for all $j \geq 2$ we move the largest item of S''_j to S'_j. Thus, we round this moved item to the largest element of S_j and do not change the value to which we round up the items of S''_j. Both these changes do not increase the migration factor by much as we prove below in Lemma 7. At the end of this procedure we end up with a collection of subsets each rounded to a common value and finish the union of sets operation.

Lemma 7. *Assume that before we apply the union of sets operation and after we finish the insertion of a new large item operation, there is a feasible solution y' to the feasibility integer program of the rounded up instance L''. After we apply the union of sets operation, if the feasibility integer program is feasible then there is a solution to it y'' such that it is enough to re-pack the items that reside in a constant number of bins.*

We now describe the algorithm we apply each time a new item arrives denoted as **Algorithm Dynamic APTAS**. If the new arriving item is small then we use Any Fit Algorithm to pack it into an existing bin or open a new bin for it if it cannot fit into any other existing bin (when we need to pack a small item, we consider the original sizes of large items that are already packed and not their rounded sizes). In the case where the new small items causes an addition of a new bin, we maintain the solution to the feasibility integer program by adding a bin whose pattern is the *empty pattern* that does not pack any large item.

Otherwise, the item is large and we are allowed (while still getting a constant migration factor) to re-pack the items of a constant number of bins. We consider the optimal rounded-up solution y before the current item has arrived. Note that y is the integer solution after the previous large item was added with the possibility of introducing empty patterns bins in case we *open* new bins to small items.

Then, after we apply the insertion of a new large item operation, we use Lemma 5 and look for a feasible packing of the resulting new rounded-up instance, y' that is close to y (i.e., the norm infinity of their difference is at most a constant that is given by the proof of Lemma 5). The restriction that y' is close to y is given by linear inequalities, and therefore we can solve the resulting feasibility integer program in polynomial time (for fixed value of ε). If the resulting integer program is infeasible, then we must open a new bin for the new rounded up instance, and then we can put the new item in the new bin and the rest of the items as they were packed in the solution before the insertion of a new large item operation occurs. Otherwise, we obtain such an integer solution y' that is close to y.

Similarly, if we need to apply either the creation of new sets operation or the union the sets operation we construct a solution y'' that is close to y' (the norm infinity of their difference is a constant). This restriction is again given by linear inequalities and therefore we can solve the resulting feasibility integer program in polynomial time (for fixed value of ε). If the resulting integer program is infeasible, then we must open a new bin for the new rounded up instance, and

then we can put the new item in the new bin and the rest of the items as they were packed in the solution before the insertion of a new large item operation occurs. Otherwise, we obtain such an integer solution y'' that is close to y. If we did not apply the creation of new sets operation or the union the sets operation, then we denote $y'' = y'$.

Note that during creation steps our algorithm packs the items from $S_1' \cup S_1''$ according to their packing in the feasibility integer program. I.e., the integer program has a row for S_1' and a row for S_1''. However, in the analysis of the performance guarantee of the algorithm in Theorem 4 below we bound the cost of the solution by a different solution that packs each item of S_1 in its own bin (this is also with for the items of $S_1' \cup S_1''$). Such a solution is not better than our resulting solution as it corresponds to a solution to the integer program that choose such patterns for its covering of the elements of $S_1' \cup S_1''$.

It remains to show how to construct a solution to the bin packing instance using the vector y''. For each pattern we change the packing of $\max\{y_p - y_p'', 0\}$ bins that were packed according to pattern p. For pattern p we select such bins arbitrarily (from the bins that we pack according to pattern p). We complete the packing of the items to a packing that correspond to y''. This will pack all the large items and all the small items that were not packed in the bins we decided to re-pack. Then, we apply Any Fit Algorithm for the small unpacked items.

This algorithm re-packs a constant number of bins in case a large item arrives and it does not change the packing in case a small item arrives.

Corollary 1. *Algorithm Dynamic APTAS has a constant migration factor for fixed value of ε.*

The proof of the following theorem is based on the analysis of Algorithm Revised FL.

Theorem 4. *Algorithm Dynamic APTAS is a polynomial time algorithm that has a constant migration factor and uses at most $(1+\varepsilon)OPT(t)+1$ bins after t items arrives.*

Remark 1. The feasibility integer program to find vectors y' and y'' (in the notations of the algorithm), which are close to y, can be solved by only one integer program of a fixed size.

4 Concluding Remarks

A similar approach allows to prove the following result. Given bins of size $1+\varepsilon$ instead of size 1, it is possible to design a dynamic algorithm which uses at time t, at most $OPT(t)$ bins to pack the items. The algorithm works as follows. Items are partitioned into large (at least $\frac{\varepsilon}{4}$) and small (all other items). The sizes of large items are rounded up into powers of $1+\frac{\varepsilon}{4}$. This rounding is permanent and the original size is ignored in all steps of the algorithm. Feasible patterns of large items are defined similarly to [12]. At each arrival of a large item we check

whether the previous amount of bins still allows a feasible solution. It is possible to show that in such a case, a limited amount of re-packing is needed. Otherwise the new item is packed in a new bin. Small items are packed greedily using Any Fit Algorithm.

A question that is left open is whether there exists a robust AFPTAS for the classical bin packing problem. It would be interesting to find out which other problems can benefit from the study of robust approximation algorithms.

References

1. E. G. Coffman, M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: A survey. In D. Hochbaum, editor, *Approximation algorithms*. PWS Publishing Company, 1997.
2. J. Csirik and G. J. Woeginger. On-line packing and covering problems. In *A. Fiat and G. J. Woeginger, editors,* Online Algorithms: The State of the Art, pages 147–177, 1998.
3. 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.
4. G. Galambos and G. J. Woeginger. Repacking helps in bounded space online bin packing. *Computing*, 49:329–338, 1993.
5. G. Gambosi, A. Postiglione, and M. Talamo. On-line maintenance of an approximate bin-packing solution. *Nordic Journal on Computing*, 4(2):151–166, 1997.
6. G. Gambosi, A. Postiglione, and M. Talamo. Algorithms for the relaxed online bin-packing model. *SIAM Journal on Computing*, 30(5):1532–1551, 2000.
7. D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: theoretical and practical results. *Journal of the ACM*, 34(1):144–162, 1987.
8. Z. Ivkovic and E. L. Lloyd. Partially dynamic bin packing can be solved within $1+\varepsilon$ in (amortized) polylogarithmic time. *Information Processing Letters*, 63(1):45–50, 1997.
9. Z. Ivkovic and E. L. Lloyd. Fully dynamic algorithms for bin packing: Being (mostly) myopic helps. *SIAM Journal on Computing*, 28(2):574–611, 1998.
10. N. Karmarkar and R. M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (FOCS'82*, pages 312–320, 1982.
11. C. C. Lee and D. T. Lee. A simple online bin packing algorithm. *Journal of the ACM*, 32(3):562–572, 1985.
12. P. Sanders, N. Sivadasan, and M. Skutella. Online scheduling with bounded migration. In *Proc. of the 31st International Colloquium on Automata, Languages and Programming (ICALP2004)*, pages 1111–1122, 2004.
13. A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley & Sons, 1986.
14. S. S. Seiden. On the online bin packing problem. *Journal of the ACM*, 49(5):640–671, 2002.
15. J. D. Ullman. The performance of a memory allocation algorithm. Technical Report 100, Princeton University, Princeton, NJ, 1971.
16. A. van Vliet. An improved lower bound for online bin packing algorithms. *Information Processing Letters*, 43(5):277–284, 1992.
17. V. V. Vazirani. *Approximation Algorithms.* Springer-Verlag, 2001.