

Approximation Algorithm for Extensible Bin Packing

Lasse Nielsen

10th April 2008

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 2 |
| 1.1 | Notation | 2 |
| 1.2 | Defining the Extensible Bin Packing problem | 2 |
| 2 | Simplifying the problem | 3 |
| 3 | Creating an FPTAAS | 6 |
| 3.1 | Step1: Rounding | 6 |
| 3.2 | Step 2: Redefining the prolem | 7 |
| 3.3 | Step 3: Weakening | 9 |
| 3.4 | Step 4a: Solving | 9 |
| 3.5 | Step 4b: Using polynomial time | 10 |
| 4 | Summing up | 11 |
| 4.1 | Other results | 11 |

References

[CL] Approximation Algorithms for Extensible Bin Packing
by E. G. Coffman, JR and George S. Lueker
from Journal of Scheduling 9: 63-69 2006.

[APPROX] Approximation Algorithms
by Vijay V. Vazirani
Georgia Institute of Technology
College of Computing

1 Introduction

1.1 Notation

Most of the notation in this project will be introduced when it is used, but there are a couple of implicit ... that will be introduced.

The term $\ln(x)$ will always represent the base-2 logarithm to x .

When we mention the *objective value* of a solution to a minimization problem, we refer to the value of the function we try to minimize called on the solution.

1.2 Defining the Extensible Bin Packing problem

The Extensible Bin Packing problem is very similar to the regular Bin Packing problem, because the objective is to pack a number of items with given rational sizes into a number of bins, but the difference is that the number of bins is defined as part of the problem instance, and not as part of the solution. This can mean that we can't be sure that we can pack the given objects in the given amount of bins, if the bins are unit sized. Therefore we allow the bins to be filled more than their unit capacity, but this will be punished by increasing the cost of the bin.

If we denote the sum of the sizes of the objects in bin k by $\ell(k)$, we can describe the cost of each bin as follows.

Definition 1.1 (The cost of a bin)

If $\ell(k) \leq 1$ then the cost of bin k is 1. If $\ell(k) > 1$ then the cost of bin k is $\ell(k)$.

This gives the following formal definition of the cost of bin k .

$$\max(1, \ell(k)).$$

The objective of the problem is to find a way to distribute the objects in the bins such that the cost of the m bins is minimal.

This should give an intuitive understanding of the extensible bin packing problem, and we will now define the problem formally.

Definition 1.2 (The Extensible Bin Packing Problem)

The input is given by m , n and x_1, x_2, \dots, x_n , where $m \in \mathbb{N}_+$ is the number of bins, $n \in \mathbb{N}_+$ is the number of objects to pack in the m bins, and $x_i \in \mathbb{Q}_+$ is the size of the i 'th object.

A solution to the problem instance m , n , x_1, x_2, \dots, x_n is a mapping ϕ from $\{1, 2, \dots, n\}$ to $\{1, 2, \dots, m\}$, and it should be interpreted such that the i 'th element will be placed in bin k if and only if $\phi(i) = k$, so a solution simply places each object in a bin. We can now define the cost of a solution in the following way.

Definition 1.3 ($\text{Cost}(\phi)$)

The objective value of a solution ϕ is denoted as

$$\text{Cost}(\phi) = \sum_{j=1}^m \max(1, \sum_{\phi(i)=j} x_i) = \sum_{j=1}^m \max(1, \ell(j))$$

Given an instance of the extensible bin packing problem, the objective is to find a solution with minimal cost.

Just for fun, we will now describe this problem as an integer programming problem.

In order to do so, we will have to consider ϕ as a collection of integer variables $y_{i,k}$ where

$$y_{i,k} = \begin{cases} 1 & \text{if } \phi(i) = k \\ 0 & \text{if } \phi(i) \neq k \end{cases}$$

Now the constraints simply have to state that ϕ must be a function from $\{1, 2, \dots, n\}$ to $\{1, 2, \dots, m\}$. This yields the following integer programming problem.

$$\begin{aligned} & \text{minimize } \sum_{k=1}^m \max \left(1, \sum_{i=1}^n y_{i,k} \cdot x_i \right) = \sum_{k=1}^m \max(1, \ell(k)) \\ & \text{subject to } \sum_{k=1}^m y_{i,k} = 1 \text{ for } i = 1 \dots n \\ & \text{where } y_{i,k} \in \{0, 1\} \text{ for } i = 1 \dots n \text{ and } k = 1 \dots m \end{aligned}$$

2 Simplifying the problem

In this section, we will show that we are allowed to make some assumptions about the instances of this problem, because all instances that fall outside of these assumptions can be solved by solving a smaller instance where the assumptions are fulfilled.

We start by making the following observation, because this will ease the argumentation of the assumptions.

Observation 2.1 We observe, that given a solution ϕ to an instance with objects x_1, x_2, \dots, x_n , the cost of ϕ can be described as the sum of the sizes of all the objects, plus the free space in the bins that are not filled up to level at least one. This gives the alternate definition of the cost of a solution

$$\text{Cost}(\phi) = \sum_{i=1}^n x_i + \sum_{\ell(k) < 1} 1 - \ell(k).$$

This definition is useful in some cases when we need to argue that one solution will be at least as good as another solution, because we are allowed to move an object from one bin to another, if it doesn't create more unused space.

Now we will introduce each of the assumptions as a lemma that proves that it is allowed to make that assumption.

Lemma 2.2 If we can find an FPTAAS for instances where all objects are greater than $\frac{\epsilon}{1+\epsilon}$, then we can find an FPTAAS for all instances.

The reason for this is that if an instance I contains objects that are not greater than $\frac{\epsilon}{1+\epsilon}$, then we can first exclude all of these objects, which yields an instance I' that the assumed FPTAAS gives us a solution ϕ' for the remaining objects. Then we can add the remaining objects using the greedy algorithm of always adding the next element in the least filled bin, and this results in a valid solution ϕ to the original instance. The time used for this is clearly polynomial in $|I|$ and $\frac{1}{\epsilon}$, but it also approximates the optimal solution by the same factor as the assumed FPTAAS. The reason for this is the following.

If the cost of ϕ is the same as the cost of ϕ' , then the result is of course as good an approximation as the original FPTAAS guarantees, because any solution to I gives a solution to I' with no greater cost, simply by removing the *small* objects.

If adding the *small* objects does increase the cost of the solution, then we know that all bins have less than $\frac{\epsilon}{1+\epsilon}$ free space. Therefore Observation 2.1 yields that $\text{OPT}(I) > \text{Cost}(\phi) - m \cdot \frac{\epsilon}{1+\epsilon}$, and since we always have that $\text{OPT}(I) \geq m$, we get that $\frac{\text{Cost}(\phi) - \text{OPT}(I)}{\text{OPT}(I)} \leq \frac{m \cdot \frac{\epsilon}{1+\epsilon}}{m} \leq \frac{\epsilon}{1+\epsilon} \leq \epsilon$.

Lemma 2.3 If we can find an FPTAAS for instances where all objects are smaller than 1, then we can find an FPTAAS for all instances.

First we observe that if there is only one bin, then finding the optimal solution is trivial, because there is only the one solution of assigning all the objects to that bin.

Then we observe that given an instance I with an object that is not smaller than 1, and where there is more than one bin, then we can create a new instance I' by removing one bin and one object of size $c \geq 1$. We can then observe that any solution ϕ' to I' yields a solution ϕ to I with $\text{Cost}(\phi) \leq \text{Cost}(\phi') + c$, and Observation 2.1 yields that any solution ϕ to I yields a solution ϕ' to I' with $\text{Cost}(\phi') \leq \text{Cost}(\phi) - c$ by eliminating the bin that the chosen object is assigned to, and moving the other objects in that bin to any other bin. We can therefore observe that $\text{OPT}(I) = \text{OPT}(I') + c$.

We can therefore create an FPTAAS by eliminating bins, and objects that are not smaller than 1, until there is only one bin left, or until all the objects are smaller than 1. If only one bin is left, we can easily find the optimal solution, otherwise we can use the assumed FPTAAS on the derived problem and find a solution ϕ' . We can then add the removed bins, and place one of the removed objects in each of those bins.

The running time of this is clearly polynomial, and if $\text{Cost}(\phi') = \text{OPT}(I') \cdot \delta$ then $\text{Cost}(\phi) = \text{Cost}(\phi') + C \leq \text{OPT}(I') \cdot \delta + C \leq (\text{OPT}(I') + C) \cdot \delta = \text{OPT}(I) \cdot \delta$ where C is the sum of the big elements that were removed in I' .

Therefore the described method is an FPTAAS.

Lemma 2.4 If we can find an FPTAAS for instances where there is an object with size $c \geq 1$ or the sum of the sizes of all the objects is less than $2m$ then we can find an FPTAAS for all instances.

The reason for this is that if we can't use the assumed FPTAAS, then all objects has size less than 1 and the sum of the sizes of all the objects is not less than 2. In this case, we can see, that using the greedy method of always placing the next object in the least filled bin will leave all bins filled to at least level 1, and therefore Observation 2.1 yields that this will return an optimal solution.

Lemma 2.5 If all objects has size less than 1 and the sum of the sizes of all the objects is less than $2m$ then there is an optimal solution in which all bins have cost less than 3.

The reason for this is that if ϕ' is an optimal solution with a bin whose cost is not less than 3, then there is another bin whose cost is not greater than 2. We can now select any object in the overfilled bin. Since all objects have sizes less than 1, we can move this object to the other bin, and the cost of that bin will still be less than 3. We can repeat this untill the cost of the overfilled bin is less than 3, and we can do this for all the overfilled bins in ϕ' , and obtain a new solution ϕ with at most the same cost¹ and where all bins have cost less than 3.

Now we have introduced a series of lemmas, and using these, it is obvious that we only need to consider instances where all objects have sizes between $\frac{\epsilon}{1+\epsilon}$ and 1, and where the sum of the sizes of all the objects are less than $2m$. Furthermore we know that there will be an optimal solution to these instances where each bin costs less than 3.

¹It must be the same cost, otherwise ϕ' wouldn't be an optimal solution

3 Creating an FPTAAS

Assume that we are given an instance I and an ε .

There are several steps in the algorithm, and we will now go through them one by one, and in each step argue that it can be done in polynomial time, and that the result will be a good enough approximation.

3.1 Step1: Rounding

In this step, we define a set of sizes, and round all the sizes of the objects to these sizes. This gives us a bounded set of sizes, with a given amount of objects of each size. This of course implies, that we can't be sure to get the same optimal value.

We will first introduce a lemma, which we will use to bound the number of values we will round the sizes to.

Lemma 3.1 The least integer x such that $\left\lfloor \frac{(1+\varepsilon)^x}{\varepsilon} \right\rfloor \cdot \varepsilon^2 \geq 1$ is not greater than $\left\lceil \frac{1}{\varepsilon} \cdot \ln\left(\frac{1}{\varepsilon}\right) + \frac{1}{\varepsilon} \right\rceil$. This is proved by the following implications.

$$x \geq \frac{1}{\varepsilon} \cdot \ln\left(\frac{1}{\varepsilon}\right) + \frac{1}{\varepsilon} \quad \Rightarrow \quad (1)$$

$$x \geq \frac{\ln\left(\frac{1}{\varepsilon}\right)}{\ln(1+\varepsilon)} + \frac{1}{\ln(1+\varepsilon)} \quad \Leftrightarrow \quad (2)$$

$$x \cdot \ln(1+\varepsilon) \geq \ln\left(\frac{1}{\varepsilon}\right) + 1 \quad \Leftrightarrow \quad (3)$$

$$\ln((1+\varepsilon)^x) \geq \ln\left(\frac{1}{\varepsilon}\right) + 1 \quad \Leftrightarrow \quad (4)$$

$$\ln((1+\varepsilon)^x) + \ln\left(\frac{1}{\varepsilon}\right) \geq 2 \cdot \ln\left(\frac{1}{\varepsilon}\right) + 1 \quad \Leftrightarrow \quad (5)$$

$$\ln((1+\varepsilon)^x) + \ln\left(\frac{1}{\varepsilon}\right) \geq \ln\left(\frac{1}{\varepsilon^2}\right) + 1 \quad \Leftrightarrow \quad (6)$$

$$\ln((1+\varepsilon)^x) - \ln(\varepsilon) \geq \ln\left(\frac{1}{\varepsilon^2} + 1\right) \quad \Leftrightarrow \quad (7)$$

$$\ln\left(\frac{(1+\varepsilon)^x}{\varepsilon}\right) \geq \ln\left(\frac{1}{\varepsilon^2} + 1\right) \quad \Leftrightarrow \quad (8)$$

$$\frac{(1+\varepsilon)^x}{\varepsilon} \geq \frac{1}{\varepsilon^2} + 1 \quad \Rightarrow \quad (9)$$

$$\left\lfloor \frac{(1+\varepsilon)^x}{\varepsilon} \right\rfloor \geq \frac{1}{\varepsilon^2} \quad \Leftrightarrow \quad (10)$$

$$\left\lfloor \frac{(1+\varepsilon)^x}{\varepsilon} \right\rfloor \cdot \varepsilon^2 \geq 1 \quad (11)$$

Step (1)-(2) is the most interesting, but it follows from the fact that $\ln(1+\varepsilon) \geq \varepsilon$ for $\varepsilon \in [0; 1]$. We also use that $\ln(y+1) \leq \ln(y) + 1$ when $y \geq 1$ in step (5)-(6). \blacksquare

We let $\rho(\varepsilon)$ denote the least integer such that $\left\lfloor \frac{(1+\varepsilon)^{\rho(\varepsilon)}}{\varepsilon} \right\rfloor \cdot \varepsilon^2 \geq 1$. If we let $s_i = \left\lfloor \frac{(1+\varepsilon)^i}{\varepsilon} \right\rfloor \cdot \varepsilon^2$ for $i = 1, 2, \dots, \rho(\varepsilon) - 1$ and $s_{\rho(\varepsilon)} = 1$, then we can round all the objectsizes to the nearest s_i , which yields a new instance I' .

We know that the new instance I' has at most $\rho(\varepsilon)$ different object-sizes, and because of Lemma 3.1, we know that $\rho(\varepsilon) \leq \left\lceil \frac{1}{\varepsilon} \cdot \ln\left(\frac{1}{\varepsilon}\right) + \frac{1}{\varepsilon} \right\rceil$.

The last thing we need to show about the new instance is that a solution ϕ to I yields a solution ϕ' to I' such that $\mathbf{Cost}(\phi') \leq (1 + \varepsilon)^2 \cdot \mathbf{Cost}(\phi)$ and a solution ϕ' to I' yields a solution ϕ to I such that $\mathbf{Cost}(\phi) \leq (1 + \varepsilon)^2 \cdot \mathbf{Cost}(\phi')$.

This is done by proving, that the rounding can change the sizes by at most a factor $(1 + \varepsilon)^2$.

Lemma 3.2 If an object of size c is rounded to s_i then $c \leq (1 + \varepsilon)^2 \cdot s_i$ and $s_i \leq (1 + \varepsilon)^2 \cdot c$. We devide the rounding cases into two groups, objects with size less than s_1 , and objects with size between s_i and s_{i+1} for some i . The reason why we don't need to consider objects with size greater than $s_{\rho(\varepsilon)}$ is that we can assume that no object has size greater than 1, because of Lemma 2.3.

If an object has size $c \leq s_1$, then it will be rounded to s_1 .

We know from Lemma 2.2, that $c \geq \frac{\varepsilon}{1+\varepsilon}$.

Therefore we get that $c \cdot (1 + \varepsilon)^2 \geq \frac{\varepsilon}{1+\varepsilon} \cdot (1 + \varepsilon)^2 = \varepsilon \cdot (1 + \varepsilon) = \varepsilon^2 \cdot \frac{1+\varepsilon}{\varepsilon} \geq \varepsilon^2 \cdot \left\lfloor \frac{1+\varepsilon}{\varepsilon} \right\rfloor = s_1$, which is what we needed to prove.

If an object has size c between s_i and s_{i+1} for some i , then $(1 + \varepsilon)^2 \cdot c \geq (1 + \varepsilon)^2 \cdot s_i$ and the following calculation proves that $(1 + \varepsilon)^2 \cdot s_i \geq s_{i+1} + 1$, so $(1 + \varepsilon)^2 \cdot c \geq s_{i+1}$ follows by transitivity.

$$\begin{aligned}
s_i \cdot (1 + \varepsilon)^2 &= \left\lfloor \frac{(1 + \varepsilon)^i}{\varepsilon} \right\rfloor \cdot \varepsilon^2 \cdot (1 + \varepsilon)^2 \\
&\geq \left(\frac{(1 + \varepsilon)^i}{\varepsilon} - 1 \right) \cdot \varepsilon^2 \cdot (1 + \varepsilon)^2 \\
&= \frac{(1 + \varepsilon)^i}{\varepsilon} \cdot \varepsilon^2 \cdot (1 + \varepsilon)^2 - (1 + \varepsilon)^2 \cdot \varepsilon^2 \\
&= \frac{(1 + \varepsilon)^{i+1}}{\varepsilon} \cdot \varepsilon^2 + (1 + \varepsilon)^{i+1} \cdot \varepsilon^2 - (1 + \varepsilon)^2 \cdot \varepsilon^2 \\
&\geq \frac{(1 + \varepsilon)^{i+1}}{\varepsilon} \cdot \varepsilon^2
\end{aligned}$$

The last inequality is a consequence of $i \geq 1$.

Because of this rounding-inaccuracy, we get that $\mathbf{OPT}(I') \leq (1 + \varepsilon)^2 \cdot \mathbf{OPT}(I)$, and if ϕ' is a solution to I' , then the solution ϕ to I which packs the same objects in the same bin will fulfill $\mathbf{Cost}(\phi) \leq (1 + \varepsilon)^2 \cdot \mathbf{Cost}(\phi')$.

The last comment to this step is that the running time clearly is linear in the number of objects to be rounded, and therefore polynomial in the inputsize.

3.2 Step 2: Redefining the prolem

By now, we have reached a new instance I' . We know that there are $\rho(\varepsilon)$ different objectsizes in I' , and that $\mathbf{OPT}(I') \leq \mathbf{OPT}(I) \cdot (1 + \varepsilon)^2$ and a solution ϕ' to I' , then it yields a solution ϕ to I such that $\mathbf{Cost}(\phi) \leq \mathbf{Cost}(\phi') \cdot (1 + \varepsilon)^2$.

We will let c_j denote the number of objects in I' with size s_j .

Because of the limited amount of object sizes, we can calculate all the possible ways to pack a bin with cost less than 3.

The following lemma proves that there can be at most $\binom{\rho(\varepsilon) + \lceil 3 \cdot \frac{1}{\varepsilon} \rceil}{\rho(\varepsilon)}$ ways of packing a bin. This is however not polynomial in $\frac{1}{\varepsilon}$, so we will have to avoid creating all the bins, if we wish to obtain an FPTAAS, but we will address this issue as the last part of the algorithm.

Lemma 3.3 The number of ways to select up to A objects of B types is at most $\binom{A+B}{A}$. To prove this, we must observe, that a set of at most A objects of B different types can be described by integers $0 \leq a_1 \leq a_2 \leq \dots \leq a_B < A$, and that all ways of selecting $0 \leq a_1 \leq a_2 \leq \dots \leq a_B < A$ yields a unique way of selecting at most A elements of B types. So we only need to figure out in how many ways $0 \leq a_1 \leq a_2 \leq \dots \leq a_B < A$ can be selected. To do this, we observe that selecting a_1, \dots, a_B is equivalent to selecting $0 \leq b_1 < b_2 < \dots < b_B < A + B - 1$ by setting $b_i = a_i + (i - 1)$. The number of ways of selecting $0 \leq b_1 < b_2 < \dots < b_B < A + B - 1$ is exactly the number of ways of selecting B numbers from $A + B - 1$ numbers, and this is by definition $\binom{A+B-1}{B} \leq \binom{A+B}{B} = \binom{A+B}{A}$. ■

Let us define $\xi(\varepsilon)$ to be the number of ways to pack a bin with cost less than 3 using objects of the given sizes. We can denote these bins $b_1, b_2, \dots, b_{\xi(\varepsilon)}$. Since Lemma 2.5 yields that there is an optimal solution to I' where no bins have greater cost than 3, all the bins in this optimal solution must be among $b_1, b_2, \dots, b_{\xi(\varepsilon)}$.

One way of finding the optimal solution to I' would be to find all ways of combining m of the found bins, and find one with a minimal cost that uses all the required objects. This would yield a PTAS, because we can assume that the input is linear in m , and there are $\binom{m+\xi(\varepsilon)}{m} \in O(m^{\xi(\varepsilon)})$ ways of selecting m of the $\xi(\varepsilon)$ types of bins. This is the same strategy as is used in [APPROX, Lemma 9.4].

The article doesn't do this, in stead it proceeds by defining the problem instance as a new integer programming problem, which describes that we wish to select m bins of the $\xi(\varepsilon)$ kinds of bins we have found, such that all the necessary objects are used, and such that the cost is minimal.

This results in the following integer programming problem, where $b_i(j)$ denotes the number of times an object of size s_j occurs in the bin b_i .

Definition 3.4

We will now define I' as a new integer programming problem.

In this problem, we need to find z_i for $i = 1 \dots \xi(\varepsilon)$, such that the following is fulfilled.

$$\begin{aligned} & \text{minimize } \sum_{i=1}^{\xi(\varepsilon)} z_i \cdot \max \left(1, \sum_{j=1}^{\rho(\varepsilon)} b_i(j) \cdot s_j \right) = \sum_{i=1}^{\xi(\varepsilon)} z_i \cdot \max(1, \ell(b_i)) \\ & \text{subject to } \sum_{i=1}^{\xi(\varepsilon)} z_i \cdot b_i(j) \geq n_j \text{ for } j = 1 \dots \rho(\varepsilon) \\ & \text{and } \sum_{i=1}^{\xi(\varepsilon)} z_i = m \\ & \text{with } z_i \in \mathbb{N}_0 \text{ for } i = 1 \dots \xi(\varepsilon) \end{aligned}$$

z_i corresponds to the amount of times we use bin b_i , and therefore the above problem is equivalent to I' , in the cases where all bins have cost at most 3.

3.3 Step 3: Weakening

By now, we have defined I' as a new integer programming problem, and by finding a solution with objective value C to this we can find a solution ϕ our original problem with $\text{Cost}(\phi) \leq C \cdot (1 + \varepsilon)^2$.

We will need to make a couple of generalizations to the problem, to make it into a Linear Programming problem.

One of the constraints in the problem is that we need to use exactly m bins, but we can observe, that if we use more bins, then we can move the objects in the exceeding bins to the first m bins, and remove the exceeding bins, Observation 2.1 yields that the new solution will not have greater cost. We can therefore change the constraint

$$\sum_{i=1}^{\xi(\varepsilon)} z_i = m \quad \text{to} \quad \sum_{i=1}^{\xi(\varepsilon)} z_i \geq m.$$

Now we need to allow z_i to be non-integer values, and this could improve the optimal solution, but since all the integer value solutions are also allowed, we know that the optimal solution can't be worse.

These changes results in the following linear programming problem.

Definition 3.5

We will now define the generalization of I' as a linear programming problem.

In this problem, we need to find z_i for $i = 1 \dots \xi(\varepsilon)$, such that the following is fulfilled.

$$\begin{aligned} & \text{minimize} \quad \sum_{i=1}^{\xi(\varepsilon)} z_i \cdot \max \left(1, \sum_{j=1}^{\rho(\varepsilon)} b_i(j) \cdot s_j \right) = \sum_{i=1}^{\xi(\varepsilon)} z_i \cdot \max(1, \ell(b_i)) \\ & \text{subject to} \quad \sum_{i=1}^{\xi(\varepsilon)} z_i \cdot b_i(j) \geq n_j \text{ for } j = 1 \dots \rho(\varepsilon) \\ & \text{and} \quad \sum_{i=1}^{\xi(\varepsilon)} z_i \geq m \\ & \text{with } z_i \geq 0 \text{ for } i = 1 \dots \xi(\varepsilon) \end{aligned}$$

We are now ready for actually solving the problem.

3.4 Step 4a: Solving

By now, we have obtained a new linear programming problem, such that a solution ϕ' to this with objective value C will result in a solution ϕ to the original problem, with $\text{Cost}(\phi) \leq C \cdot (1 + \varepsilon)^2$.

Of course we can't necessarily find an optimal integer solution, but we can find an optimal solution in polynomial time, and let us call this solution ϕ'' .

Since the number of inequalities in the linear programming problem is $\rho(\varepsilon) + 1$, we can assume that the solution we find will have at most $\rho(\varepsilon) + 1$ non-integer z_i -values.

We can therefore create an integer-solution ϕ' to I' by rounding all z_i values up.

This will of course increase the cost of the solution, but because all bins have cost at most 3, and we change at most $\rho(\varepsilon) + 1$ z_i -values, we know that

$$\begin{aligned}
\text{Cost}(\phi') &= \sum_{i=1}^{\xi(\varepsilon)} \lceil z_i \rceil \cdot \max \left(1, \sum_{j=1}^{\rho(\varepsilon)} b_i(j) \cdot s_j \right) \\
&= \sum_{i=1}^{\xi(\varepsilon)} z_i \cdot \max \left(1, \sum_{j=1}^{\rho(\varepsilon)} b_i(j) \cdot s_j \right) + \sum_{i=1}^{\xi(\varepsilon)} (\lceil z_i \rceil - z_i) \cdot \max \left(1, \sum_{j=1}^{\rho(\varepsilon)} b_i(j) \cdot s_j \right) \\
&\leq \sum_{i=1}^{\xi(\varepsilon)} z_i \cdot \max \left(1, \sum_{j=1}^{\rho(\varepsilon)} b_i(j) \cdot s_j \right) + \sum_{i=1}^{\xi(\varepsilon)} (\lceil z_i \rceil - z_i) \cdot 3 \\
&\leq \sum_{i=1}^{\xi(\varepsilon)} z_i \cdot \max \left(1, \sum_{j=1}^{\rho(\varepsilon)} b_i(j) \cdot s_j \right) + (\rho(\varepsilon) + 1) \cdot 3 \\
&\leq \text{OPT}(I') + (\rho(\varepsilon) + 1) \cdot 3
\end{aligned}$$

Therefore we can use ϕ' to obtain a solution ϕ with

$$\begin{aligned}
\text{Cost}(\phi) &\leq (1 + \varepsilon)^2 \cdot (\text{OPT}(I') + 3 \cdot (\rho(\varepsilon) + 1)) \\
&\leq (1 + \varepsilon)^2 \cdot ((1 + \varepsilon)^2 \cdot \text{OPT}(I) + 3 \cdot (\rho(\varepsilon) + 1)) \\
&\leq (1 + \varepsilon)^2 \cdot \left((1 + \varepsilon)^2 \cdot \text{OPT}(I) + 3 \cdot \left(\left\lceil \frac{1}{\varepsilon} \cdot \ln\left(\frac{1}{\varepsilon}\right) + \frac{1}{\varepsilon} \right\rceil + 1 \right) \right).
\end{aligned}$$

The only problem is that solving this problem directly, or even defining this problem uses exponential time in $\frac{1}{\varepsilon}$, because of the number of constraints. We will now proceed to solve this problem in a way that won't need all the constraints to be defined, and has a timecomplexity that is polynomially in $\frac{1}{\varepsilon}$.

3.5 Step 4b: Using polynomial time

At this point, we have proved that we just need to solve the following linear programming problem.

$$\begin{aligned}
&\text{minimize } \sum_{i=1}^{\xi(\varepsilon)} z_i \cdot \max \left(1, \sum_{j=1}^{\rho(\varepsilon)} b_i(j) \cdot s_j \right) = \sum_{i=1}^{\xi(\varepsilon)} z_i \cdot \max(1, \ell(b_i)) \\
&\text{subject to } \sum_{i=1}^{\xi(\varepsilon)} z_i \cdot b_i(j) \leq n_j \text{ for } j = 1 \dots \rho(\varepsilon) \\
&\text{and } \sum_{i=1}^{\xi(\varepsilon)} z_i \geq m \\
&\text{with } z_i \geq 0 \text{ for } i = 1 \dots \xi(\varepsilon)
\end{aligned}$$

The problem is, that the number of constraints is exponential in $\frac{1}{\varepsilon}$, so we can't even check if a solution fulfills the constraints in polynomial time.

The article states that this can be fixed by considering the dual problem, and it can be expressed as below.

$$\begin{aligned} & \text{maximize } t \cdot m + \sum_{j=1}^{\rho(\varepsilon)} u_j \cdot n_j \\ & \text{subject to } t + \sum_{i=1}^{\rho(\varepsilon)} u_j \cdot b_i(j) \leq \max \left(1, \sum_{j=1}^{\rho(\varepsilon)} b_i(j) \cdot s_j \right) \text{ for } i = 1, 2, \dots, \xi(\varepsilon) \\ & \text{with } t, u_j \geq 0 \text{ for } i = 1 \dots \rho(\varepsilon) \end{aligned}$$

The trick is now, that we don't need to consider the constraints, but by using the structure of the constraints, we can rewrite them to a *strong separation problem* that can be solved in polynomial time using dynamic programming.

We recall that the bins $b_1, b_2, \dots, b_{\xi(\varepsilon)}$ was defined as all the ways we could select objects from the $\rho(\varepsilon)$ sizes, such that the sum of the sizes was less than 3. The general idea is, that using this structure, we can create a dynamic programming formulation that solves this strong separation problem in polynomial time, and therefore we can solve the entire problem in polynomial time. The exact method of solving this is not explained in the [CL], and I have not had time to investigate this.

4 Summing up

We have now seen, how we given an instance I of an Extensible Bin Packing Problem, and an ε can find a solution with cost at most $(1 + \varepsilon)^2 \cdot (\text{OPT}(I) + 3 \cdot (\lceil \frac{1}{\varepsilon} \cdot \ln(\frac{1}{\varepsilon}) + \frac{1}{\varepsilon} \rceil + 1))$.

We have seen that the sunningtime of the algorithm is polynomial in the input size, and it is mentioned in Step 4b, that it can also become polynomial in $\frac{1}{\varepsilon}$.

It is also mentioned in Step 2, how this method could be changed to a PTAS for the Extensible Bin Packing Problem.

4.1 Other results

It is mentioned in [CL], that unless $\text{NP}=\text{P}$, it is impossible to find an FPTAS for the Extensible Bin Packing Problem.

It is also mentioned, that the simple greedy algorithm, of first sorting the elements by non-increasing size, and then sequentially assigning the elements to a bin with minimal cost obtains a factor $\frac{13}{12}$ approximation.