CMPUT 675: Approximation Algorithms **Winter 2005**

Lecture 7: Feb 4

Lecturer: Mohammad R. Salavatipour *Scribe: Xiaozhen Niu*

7.1 Bin Packing (continued)

Today we will continue working on Bin Packing problem. We will develop an asymptotic PTAS for this problem. Recall the definition of the problem:

Input: there are items $\{1, \ldots, n\}$, each has a size of $S_i \in Q^+$, and $0 \leq S_i \leq 1$.

Question: Find the minimum number of unit-size bins into which these items can be packed.

The decision version of this problem is NP-hard. In fact, using a simple reduction from the Partition problem we showed the following lower bound:

Theorem 7.1 *There is no $(3/2 - \epsilon)$-approximation algorithm for bin packing for any $\epsilon \geq 0$ unless $P = NP$.*

Our goal today is to prove the following theorem:

Theorem 7.2 *For any $\epsilon > 0$, there is a polynomial time algorithm $\mathcal{A}_\epsilon$ such that returns a solution for the bin packing problem with size at most $(1+\epsilon) \cdot OPT + 1$, where OPT is the cost of optimal solution.*

7.1.1 The Obvious Greedy Algorithm (First-Fit)

Consider the most obvious greedy algorithm for this problem, which we call it First-Fit (FF). In the First-Fit algorithm, we consider an ordering of the bins, and start putting items into bins one by one, by trying the first bin into which the item fits. Here is the formal description of the FF algorithm:

```
FF Algorithm
For $i \leftarrow 1$ to $n$ do
    Let $j$ be the first bin such that $i$ can fit into
    Put item $i$ into bin $j$
```

Theorem 7.3 *The cost of the solution of FF is at most $2OPT + 1$.*

Proof: Let I be the given instance, and define $S(I) = \sum_i S_i$ called size of the instance. Clearly $S(I) \leq Opt(I)$. The crucial point int he proof is that in the solution of FF algorithm, at most one bin can be less than half-full. Otherwise, suppose there were two bins i and j $(i < j)$ that are both less than half-full. Then all the items placed into bin j could have been put into bin i and since we consider bins in order, bin i should have been considered for those items before bin j. This is a contradiction! Therefore, all the bins, except possibly the last one are at least half-full. Thus: $\frac{1}{2}(FF(I) - 1) \leq S(I) \leq OPT$, now we conclude that $FF(I) \leq 2OPT + 1$. ■

Intuitively, it seems better if we first try to put the larger items into bins and then use the smaller ones to fill the remaining empty parts of partly used bins. This in fact gives an improvement over the FF algorithm.

Improved FF: Order the items first in non-increasing order of their sizes, i.e. assume that $S_1 \geq S_2 \geq \ldots S_n > 0$. Run the FF algorithm on the items in this order. Then:

Theorem 7.4 *The improved FF algorithm has a cost at most* $\frac{11}{9}OPT + 4$.

What if we only have a fixed number of distinct sizes, i.e. most items have equal size? Can we solve the problem in this case more efficiently, perhaps exactly?

7.1.2 Dynamic Programming Bin-Packing Algorithm

Suppose we have only a fixed number of distinct sizes items. Let's say we have a total r different sizes, for some fixed $r << n$; so many items have equal sizes.

Definition 7.5 *A vector* $(x_1 \ldots x_r)$ *is called a* ***configuration*** *if the total size of picking* x_i *copies of items of size* S_i *is* ≤ 1, *i.e., they can fit into one bin.*

Therefore, in every feasible solution, the items placed in every bin form a configuration. The number of different configurations is clearly at most $O(n^r)$, because every number x_i is between 0 and n. Since r is a fixed constant, $O(n^r)$ is polynomial in n (although can be huge). We also define $\mathcal{C}$ to be the set of all configurations.

We are going to use dynamic programming to solve the problem exactly. We have a table of size $(n+1)^r$ and $A[x_1, \ldots, x_r]$ is the minimum number of bins for packing a set of items with x_i copies of items of size s_i, for $1 \leq i \leq r$. Clearly, $A[0, \ldots, 0] = 0$ and if $(x_1, \ldots, x_r)$ is a (non-zero) configuration then $A[x_1, \ldots, x_r] = 1$. These are going to be the initial values of this table. Now we show the dynamic programming bin-packing algorithm:

Dyn-Prog Bin Packing

```
    $A[0,\ldots,0] \leftarrow 0$;
    For All configurations $(x_1,\ldots,x_r) \in C$ do
        $A[x_1,\ldots,x_r] \leftarrow 1$;
    For $i_1 \leftarrow 0$ to $n$ do
        For $i_2 \leftarrow 0$ to $n$ do
            ...
                For $i_r \leftarrow 0$ to $n$ do
                    $current \leftarrow \infty$;
                    For all $(x_1,\ldots,x_r) \in C$ do
                        if $(i_1 \geq x_1 \& i_2 \geq x_2 \& \ldots i_r \geq x_r)$ then
                            if $current \geq A[i_1 - x_1,\ldots,i_r - x_r] + 1$ then
                                $current = A[i_1 - x_1,\ldots,i_r - x_r] + 1$;
                        $A[x_1,\ldots,x_r] = current$;
```

Because $|\mathcal{C}| \in O(n^r)$ and for each entry of the table we spend $O(n^r)$ time, the total running time for this algorithm is $O(n^{2r})$. Thus, we can solve the bin packing problem in time $O(n^{2r})$ which is polynomial for each fixed r.

7.1.3 An Asymptotic PTAS for Bin-Packing

Let's go back to the greedy FF algorithm and assume assume that all the sizes are less then or equal to $\frac{\epsilon}{2}$ where $0 \leq \epsilon \leq 1$ is a small number. Then by an argument similar to the proof of Theorem 7.3 all the bins (except the last one) are "almost" full, i.e. the total size of the items in each (except possibly the last one) is greater or equal to $1 - \frac{\epsilon}{2}$. (we call it $(1 - \frac{\epsilon}{2})$-full). This implies that $(1 - \frac{\epsilon}{2})(FF(I) - 1) \leq S_I \leq OPT$. So the cost of the solution is:

$$\begin{aligned} FF(I) &\leq \frac{1}{(1-\frac{\epsilon}{2})} S_I + 1 && (7.1)\\ &\leq (1+\epsilon) S_I + 1 && (7.2)\\ &\leq (1+\epsilon) OPT + 1 && (7.3) \end{aligned}$$

So if all the items are very small we know what to do. The main problem is in the large items. For the general case, the basic steps of the algorithm includes:

1. Consider the small (size $< \frac{\epsilon}{2}$) items and large items (size $> \frac{\epsilon}{2}$) separately.
2. First pack the large items into some number of bins, let say L bins.
3. Use FF algorithm to pack the small items on the current L bins, and possibly some new bins.

Notice that in the last step, we do not open a new bin unless all the other bins are $\geq (1 - \frac{\epsilon}{2})$-full. So if we do open a new bin, then all the bins (except the last one) are $(1 - \frac{\epsilon}{2})$-full and by (7.1) the cost of the solution is at most $(1 + \epsilon)OPT + 1$. Therefore, we know that if Step 3 opens new bins we are fine. The only case that has to be considered is when Step 3 does not use any new bins. Also, we have to specify how to do Step 2.

Let I_L be the instance on large items only and let l be the number of them. For these items, we like to use the dyn-prog algorithm given in the previous section. For this, we need to have a constant number of distinct sizes. To achieve this we will partition the large items into groups and then round the sizes up so that only very few distinct sizes exist. This is the overview of the algorithm. Below we explain how to handle I_L in more details.

• Consider the large items in non-increasing order of their sizes. For some k (to be defined later), put every k consecutive items into one group, i.e. we are partitioning the large items into groups of size k each. So we get $\lceil \frac{l}{k} \rceil$ groups: $G_1, G_2, \ldots, G_{\lceil \frac{l}{k} \rceil}$ and every item in group G_i is greater than (or equal to) every item in group G_{i+1}, for every $1 \leq i < \lceil \frac{l}{k} \rceil$.

• Discard G_1 for the moment. For G_1, at most we need k bins, so we can easily fit all those items into k bins at the end.

• Round up the items in group G_i to the largest value in that group (for every $2 \leq i \leq \lceil \frac{l}{k} \rceil$). We call this instance I'. So we get $\lceil \frac{l}{k} \rceil - 1$ groups and all the items in each group have the same size.

Lemma 7.6 *The cost of* $OPT(I') \leq OPT(I_L)$.

Proof: We can discard all the items in the last group $G_{\leq \lceil \frac{l}{k} \rceil}$ and round all the items in group G_i (where $1 \leq i < \lceil \frac{l}{k} \rceil$) down to the smallest value in that group. Now we call this instance I''. Clearly, $Opt(I'') \leq Opt(I_L)$. Also, both I' and I'' have the same number of groups and the same number of items in each group. Furthermore, all the items in the i'th group of I' (which are coming from G_{i+1} of I_L) are smaller than all the items in the i'th group of I'' (which are coming from G_i of I_L). Therefore: $Opt(I') \leq Opt(I'')$. Combining these two inequalities we get the lemma. ■

Now we are ready to describe the asymptotic PTAS algorithm for bin packing. For a given error parameter ϵ, we will set $k = \lceil \epsilon S_{I_L} \rceil$ where I_L is the instance on the large items. The algorithm AP (asymptotic PTAS) includes following steps:

- Split all items into small ($\leq \frac{\epsilon}{2}$) and large ($> \frac{\epsilon}{2}$).
- Let $k = \lceil \epsilon S_{I_L} \rceil$,
- Construct I' as above from I_L.
- Apply dynamic programming algorithm to I', let us say b is the optimal solution for the instance I'.
- Pack items in G_1 into k bins (because there are only k items in G_1).
- Use FF algorithm to pack the small items.

Analysis of AP:

From the definition of I_L and k, every item in I_L has size at least $\frac{\epsilon}{2}$. So:

$$S_{I_L} = \sum_{\text{item } \mathrm{i} \in \mathrm{I_L}} S_i \geq \sum_{i=1}^{l} \frac{\epsilon}{2} = \frac{l\epsilon}{2}.$$

Therefore:

$$\frac{l}{k} = \frac{l}{\epsilon S_{I_L}} \leq \frac{l}{\epsilon \frac{l\epsilon}{2}} \leq 2/\epsilon^2$$

Thus, the running time of this algorithm is $O(n^{(4/\epsilon^2)})$ (according to the dynamic programming algorithm).

Let $AP(I)$ be the cost of the AP algorithm on instance I. We have already shown that if FF opens new bins then the cost of the algorithm is at most $(1+\epsilon)OPT + 1$. So let's assume that FF does NOT use any new bins. Then, the total number of bins used by the algorithm is:

$$\begin{aligned} OPT(I') + k &\leq OPT(I_L) + \lceil \epsilon S_{I_L} \rceil \\ &\leq OPT(I) + \epsilon OPT(I) + 1 \\ &\leq (1+\epsilon)OPT(I) + 1 \end{aligned}$$

Note: Here we assume k bins are used for group G_1 which is equal (by definition) to $\lceil \epsilon S_{I_L} \rceil$. In addition, we assumed that when using FF algorithm we do not open any new bins (this case was dealt with before).