6.891 Approximation Algorithms	February 11, 2000
Lecture 2	
Lecturer: David P. Williamson	*Scribe: Mihai Badoiu*

1 Bin Packing

We consider the bin packing problem.

Bin Packing

- **Input:** Set of items $\{1, 2, \ldots, n\}$ such that item i has size s_i and $1 \geq s_1 \geq s_2 \geq \ldots \geq s_n > 0$, $s_i \in Q^+$.
- **Goal:** Find a minimum number of bins of size 1 into which all items can be packed. (i.e., partition $\{1, 2, \ldots, n\}$ into k sets $B_1, \ldots, B_k$ such that $\forall j \sum_{i \in B_j} s_i \leq 1$ and k is minimum.)

1.1 A lower bound

We show a connection between the bin packing problem and partition, an NP-complete problem:

Partition

- **Input:** Set of items $\{1, 2, \ldots, n\}$ such that item i has size s_i and $1 \geq s_1 \geq s_2 \geq \ldots \geq s_n > 0$, $s_i \in Q^+$.
- **Goal:** Can we partition $\{1, 2, \ldots, n\}$ into 2 sets A and B such that $\sum_{i \in A} s_i = \sum_{i \in B} s_i$?

Claim 1 *There is no α-approximation algorithm for bin packing with $\alpha < \frac{3}{2}$ unless $P = NP$.*

Proof: Given an α-approximation algorithm for bin packing, for $\alpha < \frac{3}{2}$ you can solve partition exactly. Consider an instance I of partition problem. Scale the size of items such that $\sum_i s_i = 2$ and consider this as an instance I' of bin packing. If all items of I' can fit in two bins, then we have a "yes" answer to I. Otherwise, the items of I' need 3 bins and the answer to I is "no".

OPT for I' is 2 or more. So, if $\alpha < \frac{3}{2}$, then we can determine the value of OPT which implies that we can solve I. Thus, there cannot exist a α-approximation algorithm for bin packing for $\alpha < \frac{3}{2}$ unless P = NP. $\square$

However, the situation is not so bad. There exist approximation algorithms which guarantee better approximation ratios, but they also entail some small additive factor. We state below some results of this kind.

Theorem 2 *(Johnson '74) There exists a polynomial-time algorithm FFD such that*

$$FFD(I) \leq \frac{11}{9} OPT(I) + 4$$

for all instances I for the bin packing problem.

Today, we shall prove the following theorems.

Theorem 3 *(Fernandez de la Vega, Lueker '81) There exists a polynomial-time algorithm FL such that*

$$FL(I) \leq (1+\epsilon) OPT(I) + 1$$

for all bin packing instances I and any $\epsilon > 0$.

Theorem 4 *(Karmarkar, Karp '82) There exists a polynomial-time algorithm KK such that*

$$KK(I) \leq OPT(I) + O(\log^2(OPT(I))$$

for all bin packing instances I, where n is the size of instance I.

It is still an open problem whether there exists a polynomial time algorithm A such that

$$A(I) \leq OPT(I) + 1.$$

A theorem that we should have mentioned at the end of the previous lecture concerning the vertex cover problem is the following:

Theorem 5 *(Håstad '97) There is no α-approximation algorithm for vertex cover with $\alpha < \frac{7}{6}$ unless $P = NP$.*

We leave it as an exercise to the reader to determine whether theorems of this sort have meaning when bounds such as those in the claim above (that there is no α-approximation algorithm for bin packing with $\alpha < 3/2$) can be so easily overcome by including additive terms. Could there exist a $1.1OPT + 3$ approximation algorithm for vertex cover? Why or why not?

1.2 Easy Case

We start by considering an easy case when there are only r different sizes of pieces and r is constant. For this case we will show how to solve the problem in polynomial time.

Let $(a_1, a_2, \ldots, a_r)$ denote a set of pieces with a_i pieces of size s_i. A set of pieces $(a_1, a_2, \ldots, a_r)$ such that $\sum_{i=1}^{r} a_i s_i \leq 1$ is called a configuration. Let C be the set of all possible configurations. Given n pieces in total, a rough upper bound on the size of the set C is $|C| = O(n^r)$. This is because every component a_i is bounded by n. We use this fact to solve the problem in polynomial time using dynamic programming: we create a table B where $B(i_1, i_2, \ldots, i_r)$ contains the minimum number of bins needed to pack this configuration. Clearly we can define the base cases: $B(0, \ldots, 0)$ is 0, and for each configuration $(a_1, \ldots, a_r)$, $B(a_1, \ldots, a_r)$ is 1. We can now use dynamic programming to fill in the rest of the table. Suppose we attempt to find the minimum number of bins needed for an instance $(n_1, \ldots, n_r)$.

DP

```
B(0,0,...,0) ← 0
for all (a_1, a_2, ..., a_r) ∈ C
    B(a_1, a_2, ..., a_r) ← 1
for i_1 ← 0 to n_1
    for i_2 ← 0 to n_2
    ...
        for i_r ← 0 to n_r
            M ← ∞
            for all (a_1, a_2, ..., a_r) ∈ C
                if ∀j, i_j ≥ a_j
                    if M > B(i_1 − a_1, i_2 − a_2, ..., i_r − a_r) + 1
                    M ← B(i_1 − a_1, i_2 − a_2, ..., i_r − a_r) + 1
            B(i_1, i_2, ..., i_r) ← M.
```

The running time of this algorithm, **DP**, is clearly $O(n^{2r})$: the nested loop is executed at most $O(n^r)$ times, and within the loop we iterate through all $O(n^r)$ configurations.

1.3 First Fit Algorithm

This is among the most intuitive algorithms one can think of for the bin packing problem. Consider some ordering on empty bins.

FF

For $i \leftarrow 1$ to n
  Let j be the first bin such that i fits in bin j
  Put i in bin j

Theorem 6 $FF(I) \leq 2OPT(I) + 1$ *for all instances* I.

Proof: Let $SIZE(I)$ denote $\sum_i s_i$. Then it is easy to see the following lower bound for $OPT(I)$:

$$SIZE(I) \leq OPT(I).$$

We now claim that at most one bin can be half-full in the output of $FF(I)$. This is so because if there were two bins half full, then the last item added to the latter bin should have been added to the first bin. Thus,

$$\frac{1}{2}(FF(I) - 1) \leq SIZE(I)$$

which implies

$$FF(I) \leq 2SIZE(I) + 1$$

i.e.,

$$FF(I) \leq 2OPT(I) + 1.$$

$\square$

1.4 A "PTAS" for Bin Packing

We now give a "PTAS" for the bin packing problem ("PTAS" in quotes, since we need an additive factor).

Suppose size of items $s_i \leq \frac{\epsilon}{2}$ for all i in an instance I, where $\epsilon < 1$. Using arguments similar to the proof of Theorem 6, we can show that all bins except perhaps one must contain items of total size at least $1 - \frac{\epsilon}{2}$. Thus,

$$\left(1 - \frac{\epsilon}{2}\right)(FF(I) - 1) \leq SIZE(I)$$

which implies that

$$\begin{aligned} FF(I) &\leq \frac{1}{1 - \frac{\epsilon}{2}} SIZE(I) + 1 \\ &\leq (1 + \epsilon) SIZE(I) + 1 \\ &\leq (1 + \epsilon) OPT(I) + 1. \end{aligned}$$

We can use this idea to develop the following algorithm for bin packing:

1. Split the items into large (size $> \frac{\epsilon}{2}$) and small (size $\leq \frac{\epsilon}{2}$) pieces.

2. Use some algorithm to pack large pieces into b bins.

3. Use FF to pack the small pieces (possibly opening new bins).

In step 3 above, if FF doesn't open any new bin, then the algorithm uses at most b bins, and if b is close to the optimal number of bins for the large pieces alone, it is close to the optimal for the instance with all pieces. If FF opens new bins, then by the argument above, we can prove that all bins except perhaps one must contain items of total size at least $1 - \frac{\epsilon}{2}$. Thus by the inequalities above, at most $(1 + \epsilon)OPT(I) + 1$ bins are used.

So, if we can get a PTAS for bin packing problem when all items are large, then we have a PTAS for the general problem as well.

To do this, we will use the dynamic programming routine **DP**, which we developed earlier, to get a PTAS for bin packing when pieces are large. To apply this routine, we must have a constant number of piece sizes. To get a small number of piece sizes, we must round them. Rounding them down is not a good idea because once we pack them in a bin of size 1, we cannot "expand" them to their original size because the size of bin is fixed. So, we round the sizes up, and we do this via a "grouping" of pieces.

Given an instance I, arrange the pieces in decreasing order of size. Place l consecutive pieces in one group (starting from the largest size piece). Thus, the pieces are divided into $\lceil \frac{n}{l} \rceil$ groups, namely, $G_1, \ldots, G_{\lceil \frac{n}{l} \rceil}$, where G_1 contains the largest pieces and so on.

Produce another instance I' from I as follows : Discard G_1. For all other groups G_i, round the size of pieces to size of the largest piece in G_i. Thus, we have at most $\lceil \frac{n}{l} \rceil - 1$ distinct sizes.

Lemma 7 $OPT(I') \leq OPT(I)$

Proof: $OPT(I') \leq OPT(I)$ because each piece in I' can be mapped to a unique piece in I of size at least as large, e.g., pieces in G_2 in I' can be mapped to pieces in G_1 in I. So, any packing for I gives a packing for I' also. □

So, the algorithm is

FL

Separate the pieces into large ($> \frac{\epsilon}{2}$) and small ($\leq \frac{\epsilon}{2}$) pieces
$l \leftarrow \lceil \epsilon SIZE(I) \rceil$
Group the large pieces to get another instance I' as above
Pack G_1 into l bins
Apply **DP** to I' to pack it optimally into b bins
Apply FF to small pieces

Note that the number of large piece sizes in FL is at most

$$\frac{n}{l} \leq \frac{n}{\epsilon SIZE(I)} \leq \frac{n}{\epsilon \frac{n\epsilon}{2}} \leq \frac{2}{\epsilon^2}$$

because $SIZE(I) = \sum_i s_i \geq \sum_i \frac{\epsilon}{2} = \frac{n\epsilon}{2}$. Thus the running time of FL is $O(n^{4/\epsilon^2})$.

Theorem 8 *(Fernandez de la Vega, Lueker '81)* $FL(I) \leq (1+\epsilon)OPT(I) + 1$

Proof: We have already proved the theorem for the case when FF adds new bins. So, assume that FF doesn't add new bins. In this case,

$$\begin{aligned} FL(I) &\leq b + l \\ &\leq OPT(I) + \lceil \epsilon SIZE(I) \rceil, \qquad (1) \\ &\leq (1+\epsilon)OPT(I) + 1, \qquad (2) \end{aligned}$$

where (1) follows since $b = OPT(I') \leq OPT(I)$ and (2) follows since $OPT(I) \geq SIZE(I)$. $\square$

1.5 A-better-than-PTAS for Bin Packing

We turn to linear programming relaxation techniques to get an algorithm for bin packing with only an additive error term. First we formulate bin packing as an integer program. Suppose an instance I consists of m distinct piece sizes and there are n_i pieces of size s_i in I. Let N be the number of all possible configurations. Let $A_1, A_2, \ldots$ be an enumeration of all possible configurations. Let a_{ij} denote the i^{th} component of A_j, i.e., $A_j = (a_{1j}, a_{2j}, \ldots, a_{mj})$.

For each configuration A_j, we introduce a variable x_j which denotes the number of bins having configuration A_j. So an integer programming formulation of the bin packing problem is:

$$\text{Min} \quad \sum_{j=1}^{N} x_j$$

subject to

$$\sum_{j=1}^{N} a_{ij} x_j \geq n_i \quad \text{for all } i = 1, \ldots, m$$

$$x_j \in N$$

We relax this to a linear program by replacing the constraints $x_j \in N$ by $x_j \geq 0$. Note that the LP has a large number of variables. However, it can be solved efficiently, as the following theorem shows:

Theorem 9 *(Karmarkar, Karp '82) The LP above can be solved to within an additive error of at most 1 in time polynomial in m and $\log\left(\frac{n}{s_m}\right)$, where s_m is the size of smallest piece in the instance.*

We shall ensure that $s_m \geq \frac{1}{SIZE(I)}$, by filling in the smaller pieces afterwards with FF. If FF opens a new bin, this gives a solution which uses at most

$$\left(1 + \frac{2}{SIZE(I)}\right) SIZE(I) + 1 \leq OPT(I) + 3 \quad \text{bins.}$$

Also, since $s_m \geq \frac{1}{SIZE(I)}$, Theorem 9 implies that the LP can be solved in polynomial time.

Although the number of variables in the LP are very large, any extreme point will contain at most m non-zero variables. We can also assume that we can get an extreme point solution to the LP. So, if we round up the non-zero variables in an optimal solution to the LP, the number of bins used will be at most $OPT(I) + m$. However, it is possible that $m = n$, in which case this isn't saying much.

To do better than this, we introduce the following scheme, that rounds *down* the LP solution, rather than rounding up:

1. Given LP solution x^* for instance I, pack $\lfloor x_j^* \rfloor$ bins according to configuration A_j. Denote the set of pieces packed by this as I_{int}, and the remaining pieces as $I - I_{int}$.

2. Recurse on $I - I_{int}$.

To prove that this is a good approximation algorithm, we must show how to make progress in every iteration. We begin by letting $LP(I)$ denote the value of LP on instance I.

Claim 10 $LP(I - I_{int}) + LP(I_{int}) \leq LP(I)$

Proof: We know that $\lfloor x^* \rfloor$ is feasible for the LP on I_{int} and $(x^* - \lfloor x^* \rfloor)$ is feasible for the LP on $I - I_{int}$. Thus,

$$LP(I - I_{int}) + LP(I_{int}) \leq \sum_j (x_j^* - \lfloor x_j^* \rfloor) + \sum_j \lfloor x_j^* \rfloor = \sum_j x_j^* = LP(I).$$

$\square$

Lemma 11 $SIZE(I - I_{int}) \leq m$

Proof: Because x^* is an extreme point, at most m entries of x^* are non-zero. So at most m entries of $x^* - \lfloor x^* \rfloor$ are non-zero, and furthermore, each of them is less than one. Since $x^* - \lfloor x^* \rfloor$ is feasible for the LP of the bin packing instance $I - I_{int}$, by our previous rounding-up argument we can pack this instance into m bins. Thus the total size of the instance must be no more than m. $\square$

The claim above shows that we make progress if $SIZE(I) > m$. To make progress in general, we will use a new type of "geometric grouping" each time before we solve the linear program. We will show that the grouping has the effect of making $m = SIZE(I)/2$, so that by the claim above, the size of the instance will fall by a factor of 2 each time we recurse on the "fractional" part of the instance.

To perform geometric grouping, arrange the pieces in decreasing order of size as before. Starting from the first piece, add pieces to a group until the group size becomes ≥ 2. Then, start the next group and so on until all items have been allocated to some group. Let the groups thus obtained be $G_1, \ldots, G_r$, where G_i has p_i pieces.

Claim 12 *$p_i \geq p_{i-1}$ for $i = 2, \ldots, r-1$.*

Proof: The claim follows from the fact that the items sizes in G_{i-1} are no larger than that of any item in G_i. $\square$

We obtain a new grouping instance I' from I as follows

Obtaining I' from I

> Discard G_1 and G_r
> For $i \leftarrow 2$ to $r-1$
>   discard $p_i - p_{i-1}$ smallest pieces of G_i
>   round the remaining p_{i-1} pieces of G_i to the size of the largest piece of G_i

Lemma 13 *$LP(I') \leq LP(I)$.*

Proof: This follows from the same kind of reasoning as in Lemma 7. $\square$

We can now state the overall algorithm.

KK

> Let I_0 be large pieces, i.e., pieces of size $\geq \frac{1}{SIZE(I)}$
> $k \leftarrow 0$
> While ($SIZE(I_k) \geq 1$)
>   perform geometric grouping to get I'_k
>   pack the discards of the grouping using FF
>   Run LP on I'_k : pack the integral part $I'_{k,int}$ using the solution to the LP.
> $I_{k+1} \leftarrow I'_k - I'_{k,int}$
> $k \leftarrow k+1$
> Pack remainder I_k in 1 bin
> Use FF to pack small pieces

Observe that the number of groups in I' is at most $\frac{SIZE(I)}{2}$ since each group has size at least 2. In I', all pieces in the same group get the same size. So,

$$\text{number of piece sizes in } I' \quad \leq \frac{SIZE(I)}{2}$$

which implies by Lemma 11

$$SIZE(I' - I'_{int}) \leq \frac{SIZE(I)}{2}.$$

Thus, the size of an instance goes down by a factor of two after grouping, solving the linear program, rounding down, and recursing on the remaining fractional part. We stop the algorithm when $SIZE(I)$ becomes less than 1. Thus, there can be at most $O(\log(SIZE(I)))$ iterations of the algorithm.

To start the analysis of the algorithm, we need to bound the size of the discards. We show below that the total size of the discards is at most $O(\log(SIZE(I)))$. Thus FF can pack the discards in $O(\log(SIZE(I)))$ bins.

Lemma 14 *The total size of the discards from the geometric grouping is* $O(\log(SIZE(I)))$.

Proof: Clearly, $SIZE(G_1), SIZE(G_r) \leq 3$, since we stop filling a group when its size exceeds 2.

It is also easy to see that the total size of the $p_i - 1$ largest pieces in G_i is at most 2, because before the addition of the last piece, the total size of G_i did not exceed 2. Using an averaging argument, we have

$$\text{size of } p_i - p_{i-1} \text{ smallest pieces in } G_i \quad \leq \frac{2}{p_i - 1}\left(p_i - p_{i-1}\right).$$

So,

$$\begin{aligned}
\text{total size of discards} \quad &\leq \quad 6 + \sum_{i=2}^{r-1} \frac{2}{p_i - 1}\left(p_i - p_{i-1}\right) \\
&\leq \quad 6 + 2\sum_{i=2}^{r-1}\left[\frac{1}{p_i - 1} + \frac{1}{p_i - 2} + \ldots + \frac{1}{p_i - (p_i - p_{i-1})}\right] \\
&\qquad \left(\text{since } \tfrac{1}{p_i - 1} \leq \tfrac{1}{p_i - k} \text{ for any } k \geq 1\right) \\
&= \quad 6 + 2\sum_{j=n_1}^{n_{r-1}-1} \frac{1}{j} \\
&\leq \quad 6 + 2H_{n_{r-1}}
\end{aligned}$$

Since each piece has size $\geq \frac{1}{SIZE(I)}$ and each group has size at most 3, the number of pieces in each group is at most $3SIZE(I)$. In particular, $n_{r-1} \leq 3SIZE(I)$. So,

$$\begin{aligned}
\text{total size of discards} \quad &\leq \quad 6 + 2H_{n_{r-1}} \\
&= \quad O(\log(n_{r-1})) \\
&= \quad O(\log(SIZE(I)))
\end{aligned}$$

□

We can now complete the analysis of the algorithm. Clearly we require $O(\log^2(SIZE(I)))$ bins to pack the total discards ($O(\log(SIZE(I)))$ iterations with $O(\log(SIZE(I)))$ bins used in each iteration). The remainder of the analysis shows that by rounding down the LP solution each time, we lose nothing on the LP overall: the total number of bins used in packing the integral part each time is no more than the original LP value, which is a lower bound on $OPT(I)$.

Theorem 15 *(Karmarkar, Karp '82) $KK(I) \leq OPT(I)+O(\log^2(SIZE(I))$ for any bin packing instance I.*

Proof: Using Lemma 13 and the fact that there are at most $O(\log(SIZE(I)))$ iterations, we get

$$\begin{aligned}
KK(I) &= O(\log^2(SIZE(I))) + \sum_k LP(I'_{k,int}) \\
&\leq O(\log^2(SIZE(I))) + \sum_k (LP(I'_k) - LP(I'_k - I'_{k,int})) \qquad (3)\\
&\leq O(\log^2(SIZE(I))) + \sum_i (LP(I'_k) - LP(I_{k+1})) \\
&\leq O(\log^2(SIZE(I))) + \sum_i (LP(I_k) - LP(I_{k+1})) \qquad (4)\\
&= O(\log^2(SIZE(I))) + LP(I_0) \\
&\leq O(\log^2(SIZE(I))) + OPT(I),
\end{aligned}$$

where (3) follows by Claim 10 and (4) follows by Lemma 13.

□