22C:253 Lecture 4

Scribe: Rajiv Raman

September 23, 2002

Call each object j with $p_j \geq t\epsilon$ a *large* object and call the remaining objects, *small* objects. For each large object j, we have rounded p_j down to $p'_j = t\epsilon(1+\epsilon)^i$ for some integer $i \geq 0$ such that $p_j \in [t\epsilon(1+\epsilon)^i, t\epsilon(1+\epsilon)^{i+1})$. Now, notice that $p_j/(1+\epsilon) \leq p'_j \leq p_j$. As mentioned in the previous lecture, we can compute, in polynomial time, an optimal solution of the restricted version of Bin packing. If we expand the size of each bin in this packing from t to $t(1+\epsilon)$, then we get a packing of the large objects restored to their original sizes. To pack the small objects, each of size in the range $(0, t\epsilon)$, we use a greedy algorithm. In other words, we try to fit each small object into an existing bin, opening a new bin only when no old bins have space left for the object. So now we have a bin packing of the original instance of the problem. Let $\alpha(I, t, \epsilon)$ denote the number of bins used in this packing.

Lemma 1 $\alpha(I, t, \epsilon) \leq BINS(I, t)$

Proof: There are two cases depending on whether the greedy algorithm used to pack the small objects, used any new bins or not. First, suppose that the greedy algorithm used no new bins. This means that $\alpha(I, t, \epsilon)$ is equal to the optimal number of i bins used to pack the restricted instance of the problem. Note that in this instance we are packing only large objects and each of these objects has shrunk in size from p_j to p'_j. This implies that $\alpha(I, t, \epsilon) \leq BINS(I, t)$.

Suppose the greedy algorithm did use new bins. This means except the last opened bin, all other bins are full at least to the extent t. Hence, in *any* bin packing of the original instance with size-t bins, we must use at least $\alpha(I, t, \epsilon)$ bins. This implies that $\alpha(I, t, \epsilon) \leq BINS(I, t)$. □

We would like to view $\alpha(I, t, \epsilon)$ as a quickly computable approximation for $BINS(I, t)$. Specifically, we replace the query "Is $BINS(I, t) \leq m$?" by "Is $\alpha(I, t, \epsilon) \leq m$?" The algorithm now is essentially doing a binary search in $[LB, 2LB]$ for $\min\{t \mid \alpha(I, t, \epsilon) \leq m\}$.

Since $\alpha(I, t, \epsilon) \leq BINS(I, t)$ this implies the following:

1. If query $BINS(I, t) \leq m$ has a **YES** answer, then the query $\alpha(I, t, \epsilon)$ also has a **YES** answer.
2. If query $BINS(I, t) \leq m$ has a **NO** answer, then the query $\alpha(I, t, \epsilon) \leq m$ may have a **YES** or **NO** answer. However, if $\alpha(I, t, \epsilon) \leq m$ has a **YES** answer we know that $BINS(I, t(1+\epsilon))$ has a **YES** answer.

Let $t^* = \min\{t \mid \alpha(I, t, \epsilon) \leq m\}$. The above remarks imply that $t^* \leq OPT \leq t^*(1+\epsilon)$. Recall that $OPT = \min\{t \mid \alpha(I, t, \epsilon) \leq m\}$. So, if we could find t^*, we could return $a = t^*(1+\epsilon)$ as the answer and we would have $a \leq OPT(1+\epsilon)$. However, even though we can answer the query "Is $\alpha(I, t, \epsilon) \leq m$" in polynomial time, we still cannot do the binary search in polynomial time. Here is how we get around this problem.

In each iteration of the search, we shrink the search interval by 1/2. The search interval is originally $[LB, 2 \cdot LB]$ and so after k iterations, the search interval is of size $LB/2^k$. We stop when the interval size is at most ϵLB. This implies that

$$\frac{LB}{2^k} \leq \epsilon LB < \frac{LB}{2^{k-1}}$$

and hence, $k \geq \log_2 \frac{1}{\epsilon} > k - 1$. This implies that $k = \lceil \log_2 \frac{1}{\epsilon} \rceil$.

Note that the right end-point of every search interval corresponds to a **YES** answer to the query "Is $\alpha(I, t, \epsilon) \leq m$?" Furthermore, for any t smaller than the left end-point of the search interval, the query has a **NO** answer. So when we stop at a search interval $[a', b']$ we know that $a' \leq t^* \leq b'$. So we return the right end-point b' as the result of the binary search. Since $b' - a' \leq \epsilon \cdot LB$, we have that

$$b' \leq t^* + \epsilon \cdot LB \leq t^* + \epsilon t^* \leq t^*(1 + \epsilon).$$

The result b' of the binary search get multiplied by $(1 + \epsilon)$ before being finally returned, and so

$$b'(1 + \epsilon) \leq t^*(1 + \epsilon)^2 \leq OPT(1 + \epsilon)^2.$$

For $\epsilon < 1$, $\epsilon^2 < \epsilon$ and so for $\epsilon < 1$,

$$b'(1 + \epsilon) \leq OPT(1 + 3\epsilon).$$

Also note that for $\epsilon \geq 1$, we might as well use the simple greedy algorithm for MMS.

The running time of of this algorithm is

$$\mathbf{O}\Big(\left\lceil \log_2 \frac{1}{\epsilon} \right\rceil n^{2(\lceil \log_{1+\epsilon} \frac{1}{\epsilon} \rceil + 1)}\Big).$$

This is because there are $\lceil \log_2 \frac{1}{\epsilon} \rceil$ iterations of the binary search and in each iteration the restricted Bin packing problem is solved in $O(n^{2(\lceil \log_{1+\epsilon} \frac{1}{\epsilon} \rceil + 1)})$ time. Since the running time depends exponentially on $\frac{1}{\epsilon}$, the algorithm is a **PTAS** and not an **FPTAS**

Integer LP formulation

All the combinatorial optimization problems we have seen so far have simple integer linear programs formulations. Here are some examples.

SET COVER

Let x_i, $i = 1, \ldots, k$ be indicator variables that denote whether S_i is in the solution or not. In other words, $x_i = 1$ if S_i belongs to the solution, and $x_i = 0$ otherwise.

SET COVER consists of minimizing

$$\sum_{i=1}^{k} cost(S_i) \cdot x_i$$

subject to the constraint that each element in the universe if covered. This is equivalent to saying that for each element $j \in U$

$$\sum_{i: j \in S_i} x_i \geq 1.$$

Thus SET COVER is equivalent to the integer linear program (ILP):

$$\min \sum_{i=1}^{k} cost(S_i) \cdot x_i$$

$$\begin{aligned} \sum_{i:j \in S_i} x_i &\geq 1 \text{ for all } j = 1, 2, \ldots, n \\ x_i &\in 0, 1 \text{ for all } i = 1, 2, \ldots, k \end{aligned}$$

KNAPSACK
Define x_i, $i = 1, 2, \ldots, n$ as indicator variable for each object. Then the knapsack problem can be stated as the following maximization problem:

$$\max \sum_{i=1}^{n} profit(a_i) x_i$$

$$\begin{aligned} \sum_{i=1}^{n} size(a_i) \cdot x_i &\leq B \\ x_i &\in \{0, 1\} \text{ for all } i = 1, 2, \ldots, n \end{aligned}$$

The first constraint forces the total size of the objects chosen to be no greater than the capacity B of the knapsack.

MINIMUM MAKESPAN
Let x_{ij}, $i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, m$ be the indicator variables telling us if job i is assigned to machine j. Then

$$\sum_{i=1}^{n} x_{ij} \cdot p_i$$

is the completion time of machine j. Let T be a variable whose value is below by the completion time of all the machines. In other words,

$$T \geq \sum_{i=1}^{n} x_{ij} \cdot p_i, \text{ for all } j = 1, 2, \ldots, n.$$

Then the MINIMUM MAKESPAN problem is equivalent to:

$$\min \ T$$

$$\begin{aligned} \sum_{j=1}^{m} x_{ij} &= 1 \text{ for all } i = 1, 2, \ldots, n \\ \sum_{i=1}^{n} x_{ij} \cdot p_i &\leq T \text{ for all } j = 1, 2, \ldots, n \\ x_{ij} &\in \{0, 1\} \text{ for all } i = 1, 2, \ldots, n \text{ and } j = 1, 2, \ldots, m \end{aligned}$$

The first constraint ensures that every object is assigned to exactly one machine. In each of these examples, the last constraint forces the solutions to be integral. This constraint is what makes the problems ILP, rather than just LP.