22C:253 Lecture 3

Scribe: Samir Jain

September 9, 2002

KNAPSACK:
Input: Objects $a_1, a_2, \ldots, a_n$ with positive integer sizes given by $size(a_i)$, $1 \leq i \leq n$, positive integer profits given by $profit(a_i)$, $1 \leq i \leq n$, and a knapsack of capacity $B \in Q^+$.
Output: A collection of objects with total size at most B and maximum total profit.

Note that the total size of a collection simply refers to the sum of sizes of the objects in the collection and similarly, the total profit of a collection refers to the sum of profits of the objects in the collection. In other words, we want to find a set $A' \subseteq \{a_1, a_2, \ldots, a_n\}$ such that $\sum_{a_i \in A'} size(a_i) \leq B$ and $\sum_{a_i \in A'} profit(a_i)$ is maximized.

Simple Dynamic Algorithm For Knapsack Let $P = max\{profit(a_i) \mid 1 \leq i \leq n\}$. Since we have n objects, nP is the maximum total profit possible. Recall that profits are positive integers. For each $p \in \{1, 2, 3, ..., nP\}$, let A_p denote the size of a smallest set with profit equal to p. Suppose we have computed A_p for every $p \in \{1, 2, ..., nP\}$ then we just have to find largest p such that $A_p \leq B$. To compute the A_p's, we first compute $A_{i,p}$ where, $1 \leq i \leq n$, $p \in \{1, 2, ..., nP\}$ and $A_{i,p}$ is defined as the size of a smallest subset of $\{a_1, a_2, \ldots, a_i\}$ with profit equal to p. Note that "size of a subset" here does not refer to the cardinality of the subset, it refers to the sum of the sizes of the objects in the set. Computing $A_{i,p}$ is essentially filling an $n \times nP$ table. Also note that $A_p = A_{n,p}$ for each p.

It is obvious that for each $p \in \{1, 2, \ldots, nP\}$,

$$A_{1,p} = \begin{cases} 0 & \text{if } p \neq profit(a_i) \\ size(a_i) & \text{if } p = profit(a_i) \end{cases}$$

For each i, $2 \leq i \leq n$, and $p \in \{1, 2, 3, ..., nP\}$

$$A_{i,p} = \begin{cases} \min\{A_{i-1,p-profit(a_i)} + size(a_i), A_{i-1,p}\} & \text{if } profit(a_i) < p \\ \min\{size(a_i), A_{i-1,p}\} & \text{if } profit(a_i) = p \\ A_{i-1,p} & \text{otherwise} \end{cases}$$

This step of computing $A_{i,p}$ for any i and p takes $O(1)$ and since we have an $n \times nP$ table to fill the total time taken is $O(n^2P)$.

Now note that this is not a polynomial time algorithm in general. For example, assume that each of the object sizes can be represented in c bits for some constant c. Further assume that $P = profit(a_i)$ for all i and $P = 2^n$. Then the input can be represented in $O(n^2)$ bits, while the running time of the algorithm is $O(n^2 \cdot 2^n)$. Specifically, such an algorithm is said to run in

pseudopolynomial time. The notion of pseudopolynomial running time can be precisely defined as follows. Let I be an instance of problem and I_u denote I expressed in unary. If an algorithm for the problem runs in time polynomial in $|I_u|$ then the running time is said to be pseudopolynomial.

We now convert this dynamic programming algorithm into FPTAS (Fully Polynomial Time Approximation Scheme) for KNAPSACK. Like with other constructions of FPTASs and PTASs here we trade off accuracy for running time.

Algorithm

1. "Shrink" the profits by assigning to each a_i assign a new profit:

   $$profit'(a_i) = \lfloor \frac{profit(a_i)}{K} \rfloor$$

   for some fixed positive K to be chosen later.

2. Solve KNAPSACK using the new profits and using the dynamic programming algorithm discussed earlier.

3. Report the subset produced as the solution.

Analysis

Let O be an optimal solution of KNAPSACK and let O' be solution produced by above algorithm. Clearly,

$$OPT = \sum_{a_i \in O} profit(a_i) \qquad \text{and} \qquad \sum_{a_i \in O'} profit(a_i) \leq OPT$$

Since O' is optimal in shrunken instance, we have

$$\sum_{a_i \in O'} \lfloor \frac{profit(a_i)}{K} \rfloor \geq \sum_{a_i \in O} \lfloor \frac{profit(a_i)}{K} \rfloor.$$

We can bound the lefthand side (LHS) above as follows:

$$LHS \equiv \sum_{a_i \in O'} \lfloor \frac{profit(a_i)}{K} \rfloor \leq \sum_{a_i \in O'} \frac{profit(a_i)}{K} = \frac{profit(O')}{K}.$$

Similarly, we can bound from below the righthand side (RHS) by

$$RHS \equiv \sum_{a_i \in O} \lfloor \frac{profit(a_i)}{K} \rfloor \geq \sum_{a_i \in O} (\frac{profit(a_i)}{K} - 1) \geq \sum_{a_i \in O} \frac{profit(a_i)}{K} - n = \frac{OPT}{K} - n.$$

Thus,

$$\frac{profit(O')}{K} \geq \frac{OPT}{K} - n$$

and multiplying both sides by K we get

$$profit(O') \geq OPT - Kn.$$

For a given $\epsilon > 0$, choose K such that $Kn = P\epsilon$. This implies that

$$profit(O') \geq OPT - P\epsilon$$

and without loss of generality we can assume that $P \leq OPT$. Therefore,

$$profit(O') \geq OPT - \epsilon \cdot OPT = (1 - \epsilon)OPT.$$

Running Time Analysis The dynamic programming algorithm takes $O(n^2 \lfloor \frac{P}{K} \rfloor)$ time now and since $K = \frac{P\epsilon}{n}$, the running time of the algorithm is $O(n^3/\epsilon)$. Therefore the algorithm is polynomial in n and polynomial in $1/\epsilon$.

So we have presented an FPTAS for knapsack. The precise definition of a PTAS and an FPTAS is the following. For a *minimization problem* Π, an algorithm A is a PTAS if, for every $\epsilon > 0$ and for every instance I of Π, A produces solution s such that

$$Cost_{\Pi}(I, s) \leq (1 + \epsilon)OPT.$$

The running time of the algorithm is polynomial in the size of the input, but depends arbitrarily on ϵ. A FPTAS additionally satisfies the requirement that the running time depends polynomially on $1/\epsilon$. A PTAS and an FPTAS are defined similarly for a maximization problem except that the algorithm produces a solution s such that

$$Cost_{\Pi}(I, s) \geq (1 - \epsilon)OPT.$$