CS 598CSC: Approximation Algorithms	Lecture date: February 9, 2011
Instructor: Chandra Chekuri	Scribe: Kyle Fox (2009)

In this lecture we explore the KNAPSACK problem. This problem provides a good basis for learning some important procedures used for approximation algorithms that give better solutions at the cost of higher running time.

1 The Knapsack Problem

1.1 Problem Description

In the KNAPSACK problem we are given a knapsack capacity B, and set N of n items. Each item i has a given size $s_i \geq 0$ and a profit $p_i \geq 0$. Given a subset of the items $A \subseteq N$, we define two functions, $s(A) = \sum_{i \in A} s_i$ and $p(A) = \sum_{i \in A} p_i$, representing the total size and profit of the group of items respectively. The goal is to choose a subset of the items, A, such that $s(A) \leq B$ and $p(A)$ is maximized. We will assume every item has size at most B. It is not difficult to see that if all the profits are 1, the natural greedy algorithm of sorting the items by size and then taking the smallest items will give an optimal solution. Assuming the profits and sizes are integral, we can still find an optimal solution to the problem relatively quickly using dynamic programming in either $O(nB)$ or $O(nP)$ time, where $P = \sum_{i=1}^{n} p_i$. Finding the details of these algorithms was given as an exercise in the practice homework. While these algorithms appear to run in polynomial time, it should be noted that B and P can be exponential in the size of the input assuming the numbers in the input are not written in unary. We call these *pseudo-polynomial time algorithms* as their running times are polynomial only when numbers in the input are given in unary.

polynomial in numeric value of input, rather than length (~ value = exp{length})

1.2 A Greedy Algorithm

Consider the following greedy algorithm for the KNAPSACK problem which we will refer to as GREEDYKNAPSACK. We sort all the items by the ratio of their profits to their sizes so that $\frac{p_1}{s_1} \geq \frac{p_2}{s_2} \geq \cdots \geq \frac{p_n}{s_n}$. Afterward, we greedily take items in this order as long as adding an item to our collection does not exceed the capacity of the knapsack. It turns out that this algorithm can be arbitrarily bad. Suppose we only have two items in N. Let $s_1 = 1$, $p_1 = 2$, $s_2 = B$, and $p_2 = B$. GREEDYKNAPSACK will take only item 1, but taking only item 2 would be a better solution. As it turns out, we can easily modify this algorithm to provide a 2-approximation by simply taking the best of GREEDYKNAPSACK's solution or the most profitable item. We will call this new algorithm MODIFIEDGREEDY.

maximization problem: want to show p(A_opt) <= 2*p(A)

Theorem 1 MODIFIEDGREEDY *has an approximation ratio of* 1/2 *for the* KNAPSACK *problem.*

Proof: Let k be the index of the first item that is not accepted by GREEDYKNAPSACK. Consider the following claim:

Claim 2 $p_1 + p_2 + \ldots p_k \geq \text{OPT}$. *In fact,* $p_1 + p_2 + \cdots + \alpha p_k \geq \text{OPT}$ *where* $\alpha = \frac{B - (s_1 + s_2 + \cdots + s_{k-1})}{s_k}$ *is the fraction of item* k *that can still fit in the knapsack after packing the first* $k - 1$ *items.*

The proof of Theorem 1 follows immediately from the claim. In particular, either $p_1 + p_2 + \cdots + p_{k-1}$ or p_k must be at least OPT/2. We now only have to prove Claim 2. We give an LP relaxation of the KNAPSACK problem as follows: Here, $x_i \in [0, 1]$ denotes the fraction of item i packed in the knapsack.

$$\begin{aligned} \text{maximize } & \sum_{i=1}^{n} p_i x_i \\ \text{subject to } & \sum_{i=1}^{n} s_i x_i \leq B \\ & x_i \leq 1 \text{ for all } i \text{ in } \{1 \ldots n\} \\ & x_i \geq 0 \text{ for all } i \text{ in } \{1 \ldots n\} \end{aligned}$$

Let OPT$'$ be the optimal value of the objective function in this linear programming instance. Any solution to KNAPSACK is a feasible solution to the LP and both problems share the same objective function, so OPT$' \geq$ OPT. Now set $x_1 = x_2 = \cdots = x_{k-1} = 1$, $x_k = \alpha$, and $x_i = 0$ for all $i > k$. This is a feasible solution to the LP that cannot be improved by changing any one tight constraint, as we sorted the items. Therefore, $p_1 + p_2 + \cdots + \alpha p_k = \text{OPT}' \geq \text{OPT}$. The first statement of the lemma follows from the second as $\alpha \leq 1$. □

1.3 A Polynomial Time Approximation Scheme

Using the results from the last section, we make a few simple observations. Some of these lead to a better approximation.

Observation 3 *If for all i, $s_i \leq \epsilon B$,* GREEDYKNAPSACK *gives a $(1 - \epsilon)$ approximation.*

This follows from the deductions below:

$$\begin{aligned} \text{For } 1 \leq i \leq k,\ p_i/s_i &\geq p_k/s_k \\ \Rightarrow p_1 + p_2 + \cdots + p_k &\geq (s_1 + s_2 + \cdots + s_k) p_k/s_k \\ \Rightarrow p_k &\leq s_k(p_1 + p_2 + \cdots + p_k)/B \\ &\leq \epsilon(p_1 + p_2 + \cdots + p_k) \\ &\leq \epsilon(p_1 + p_2 + \cdots + p_{k-1})/(1 - \epsilon) \\ \Rightarrow p_1 + p_2 + \cdots + p_{k-1} &\geq (1 - \epsilon)\text{OPT} \end{aligned}$$

The third line follows because $s_1 + s_2 + \cdots + s_k > B$, and the last from Claim 2.

Observation 4 *If for all i, $p_i \leq \epsilon$*OPT*,* GREEDYKNAPSACK *gives a $(1 - \epsilon)$ approximation.*

This follows immediately from Claim 2.

Observation 5 *There are at most $\lceil \frac{1}{\epsilon} \rceil$ items with profit at least ϵ*OPT *in any optimal solution.*

We may now describe the following algorithm. Let $\epsilon \in (0, 1)$ be a fixed constant and let $h = \lceil \frac{1}{\epsilon} \rceil$. We will try to guess the h most profitable items in an optimal solution and pack the rest greedily.

GUESS H + GREEDY(N, B):
For each $S \subseteq N$ such that $|S| \leq h$:
  Pack S in knapsack of size at most B
  Let i be the least profitable item in S. Remove all items $j \in N - S$ where $p_j > p_i$.
  Run GREEDYKNAPSACK on remaining items with remaining capacity $B - \sum_{i \in S} s_i$
Output best solution from above

Theorem 6 GUESS H + GREEDY *gives a* $(1 - \epsilon)$ *approximation and runs in* $O(n^{\lceil 1/\epsilon \rceil + 1})$ *time.*

Proof: For the running time, observe that there are $O(n^h)$ subsets of N. For each subset, we spend linear time greedily packing the remaining items. The time initially spent sorting the items can be ignored thanks to the rest of the running time.

For the approximation ratio, consider a run of the loop where S actually is the h most profitable items in an optimal solution and the algorithm's greedy stage packs the set of items $A' \subseteq (N - S)$. Let OPT' be the optimal way to pack the smaller items in $N - S$ so that $\text{OPT} = p(S) + \text{OPT}'$. Let item k be the first item rejected by the greedy packing of $N - S$. We know $p_k \leq \epsilon\text{OPT}$ so by Claim 2 $p(A') \geq \text{OPT}' - \epsilon\text{OPT}$. This means the total profit found in that run of the loop is $p(S) + p(A') \geq (1 - \epsilon)\text{OPT}$. □

Note that for any fixed choice of epsilon, the algorithm above runs in polynomial time. This type of algorithm is known as a *polynomial time approximation scheme* or PTAS. We say a maximization problem Π has a PTAS if for all $\epsilon > 0$, there exists a polynomial time algorithm that gives a $(1 - \epsilon)$ approximation ($(1 + \epsilon)$ for minimization problems). In general, one can often find a PTAS for a problem by greedily filling in a solution after first searching for a good basis on which to work. As described below, KNAPSACK actually has something stronger known as a *fully polynomial time approximation scheme* or FPTAS. A maximization problem Π has a FPTAS if for all $\epsilon > 0$, there exists an algorithm that gives a $(1-\epsilon)$ approximation ($(1+\epsilon)$ for minimization problems) and runs in time polynomial in both the input size and $1/\epsilon$.

1.4 Rounding and Scaling

Earlier we mentioned exact algorithms based on dynamic programming that run in $O(nB)$ and $O(nP)$ time but noted that B and P may be prohibitively large. If we could somehow decrease one of those to be polynomial in n without losing too much information, we might be able to find an approximation based on one of these algorithms. Let $p_{\max} = \max_i p_i$ and note the following.

Observation 7 $p_{\max} \leq \text{OPT} \leq np_{\max}$

Now, fix some $\epsilon \in (0, 1)$. We want to scale the profits and round them to be integers so we may use the $O(nP)$ algorithm efficiently while still keeping enough information in the numbers to allow for an accurate approximation. For each i, let $p'_i = \lfloor \frac{n}{\epsilon} \frac{1}{p_{\max}} p_i \rfloor$. Observe that $p'_i \leq \frac{n}{\epsilon}$ so now the sum of the profits P' is at most $\frac{n^2}{\epsilon}$. Also, note that we lost at most n profit from the scaled optimal solution during the rounding, but the scaled down OPT is still at least $\frac{n}{\epsilon}$. We have only lost an ϵ fraction of the solution. This process of rounding and scaling values for use in exact algorithms has use in a large number of other maximization problems. We now formally state the algorithm ROUND&SCALE and prove its correctness and running time.

ROUND&SCALE(N, B):
For each i set $p'_i = \lfloor \frac{n}{\epsilon} \frac{1}{p_{\max}} p_i \rfloor$
Run exact algorithm with run time $O(nP')$ to obtain A
Output A

Theorem 8 ROUND&SCALE *gives a* $(1-\epsilon)$ *approximation and runs in* $O(\frac{n^3}{\epsilon})$ *time.*

Proof: The rounding can be done in linear time and as $P' = O(\frac{n^2}{\epsilon})$, the dynamic programing portion of the algorithm runs in $O(\frac{n^3}{\epsilon})$ time. To show the approximation ratio, let $\alpha = \frac{n}{\epsilon} \frac{1}{p_{\max}}$ and let A be the solution returned by the algorithm and A^* be the optimal solution. Observe that for all $X \subseteq N$, $\alpha p(X) - |X| \leq p'(X) \leq \alpha p(X)$ as the rounding lowers each scaled profit by at most 1. The algorithm returns the best choice for A given the scaled and rounded values, so we know $p'(A) \geq p'(A^*)$.

$$p(A) \geq \frac{1}{\alpha} p'(A) \geq \frac{1}{\alpha} p'(A^*) \geq p(A^*) - \frac{n}{\alpha} = \text{OPT} - \epsilon p_{\max} \geq (1-\epsilon)\text{OPT}$$

□

It should be noted that this is not the best FPTAS known for KNAPSACK. In particular, [1] shows a FPTAS that runs in $O(n \log(1/\epsilon) + 1/\epsilon^4)$ time.

2 Other Problems

The following problems are related to Knapsack; they can also be viewed as special cases of the set covering problems discussed previously. Can you see how?

2.1 Bin Packing

BINPACKING gives us n items as in KNAPSACK. Each item i is given a size $s_i \in (0, 1]$. The goal is to find the minimum number of bins of size 1 that are required to pack all of the items. As a special case of SET COVER, there is an $O(\log n)$ approximation algorithm for BINPACKING, but it is easy to see that one can do much better. In fact most greedy algorithms give a factor of 2 or better. There is an algorithm for BINPACKING that guarantees a packing of size $(1+\epsilon)\text{OPT} + 1$.

2.2 Multiple Knapsack

MULTIPLEKNAPSACK gives us m knapsacks of the same size B and n items with sizes and profits as in KNAPSACK. We again wish to pack items to obtains as large a profit as possible, except now we have more than one knapsack with which to do so. The obvious greedy algorithm based on MAXIMUM COVERAGE yields a $(1 - 1/e)$ approximation; can you do better?

References

[1] E. L. Lawler. Fast Approximation Algorithms for Knapsack Problems. *Mathematics of Operations Research*, 4(4): 339–356, 1979.