CS 391I/491I – Approximation Algorithms
Lecture Notes

April 3-5, 2001

Gayane Goltukhchyan

KNAPSACK Problem

1. Problem Statement

Given a knapsack of capacity W and n objects $o_1, o_2 \ldots\ldots o_n$ having weights $w_1, w_2 \ldots\ldots w_n$ and profits values $p_1, p_2 \ldots\ldots p_n$, select some subset of these objects to be placed in the knapsack, so that the profit of the objects in the knapsack is maximized, while not violating its capacity constraints.

Let us formulate IP to solve the Knapsack problem. In order to do that associate with object o_i a binary variable x_i such that

$$x_i = \begin{cases} 1 & \text{if object } o_i \text{ is in the knapsack} \\ 0 & \text{otherwise} \end{cases}$$

The problem can be expressed as follows:

$$\text{MAX} \sum_{i=1}^{n} p_i x_i$$

$$\text{s.t.} \quad \sum_{i=1}^{n} w_i x_i \le W$$

$$x_i \in \{0,1\}$$

Consider now situation where fractional solution is allowed. In that case we know that polynomial algorithm exists: we relax the problem to LP problem and get solution M_{LP}. In case if we do not want to use LP to solve the problem, we can use *Greedy* strategy.

2. Greedy Algorithm.

Consider quantity $\frac{p_i}{w_i}$. Assume we order these quantities in the non-increasing order:

Step 1. $\frac{p_1}{w_1} \ge \frac{p_2}{w_2} \ge \ldots\ldots\ldots \frac{p_n}{w_n}$

Step 2. Use *Greedy* strategy and get M_{greedy}.

Let us show that Greedy strategy though good for fractional Knapsack fails for {0,1} Knapsack.

Example,

$$W = 5 \quad \frac{p_1}{w_1} = \frac{16}{4}, \frac{p_2}{w_2} = \frac{7}{2}, \frac{p_3}{w_3} = \frac{10}{3}$$

Greedy strategy produces solution with profit P = 16, whereas optimal solution is P = 17.

3. Exact Algorithm for MAXIMUM-KNAPSACK problem.

We can solve Knapsack {0,1} using Dynamic Programming technique, which crucial observation is that optimal structure has within it optimal substructure.

Dynamic programming can be applied to any problem for which an optimal solution of the problem can be derived by composing optimal solutions of a limited set of 'subproblems', regardless of how these solutions have been obtained. This is generally called the *principle of optimality*.

For any k with $1 \le k \le n$ and for any p with $0 \le p \le \sum_{i=1}^{n} p_i$ we consider the problem of finding a subset of $\{x_1, x_2, \ldots x_k\}$ which minimizes the total size among all those subsets having total profit equal to p and the total size at most W.

Denote with $M^*(k, p)$ an optimal solution of this problem and with $S^*(k, p)$ the corresponding optimal size. Assume that whenever $M^*(k, p)$ is not defined,

$S^*(k, p) = 1 + \sum_{i=1}^{n} p_i$.

Let $P = \sum_{i=1}^{n} p_i$ and

$$M^*(k, p) = \begin{cases} \text{smallest weight subset of } \{x_1, x_2 \ldots\ldots x_k\} \text{ to achieve a profit p} \\ \text{undefined, otherwise} \end{cases}$$

We clearly can see that

$M^*(1,0) = \varnothing$

$M^*(1, p_1) = \{x_1\}$

$M^*(1, p) = \text{undefined for any positive integer } p \ne p_i$
$p \ne p_1$

In general for any k with $2 \le k \le n$ and for any p with $0 \le p \le \sum_{i=1}^{n} p_i$ following relationship holds:

$$M^*(k,p) = \begin{cases} M^*(k-1,p-p_k) \cup \{x_k\}, \text{ if } p_k \le p, M^*(k-1,p-p_k) \text{ is defined,} \\ \qquad\qquad S^*(k-1,p) \text{ is at least } S^*(k-1,p-p_k) + w_k, \\ \qquad\qquad \text{and } S^*(k-1,p-p_k) + w_k \le W \\ M^*(k-1,p) \quad \text{otherwise} \end{cases}$$

That is, the best subset of $\{x_1, x_2, \ldots x_k\}$ that has total profit p is either the best subset of $\{x_1, x_2, \ldots x_{k-1}\}$ that has total profit $p - p_k$ plus item x_k or the best subset of $\{x_1, x_2, \ldots x_{k-1}\}$ that has total profit p. Since the best subset of $\{x_1, x_2, \ldots x_k\}$ that has total profit p must either contain x_k or not,, one of these two choices must be the right one.

The table can be filled based on above formula to help to understand relationships as well as to store data.

	1	2	3	4	5				P
1	∅	∅	$\{x_1\}$	u	u				u
2	∅	∅	$\{x_1\}$	u	$\{x_1, x_2\}$				u
3	∅	∅	$\{x_1\}$	u	$\{x_3\}$				u
n-1	∅	∅	$\{x_1\}$	u					
n	∅	∅	$\{x_1\}$	u					

Here p_1= 3, p_2= 1, p_3= 5, u – undefined.
The right most value in the last row that is not undefined is the optimal solution.

Running Time

Running time of the Algorithm is O(n * P). It depends on the input value of P. This type of algorithms is called Pseudo-polynomial Algorithms. Each p_i requires $\log p_i$ bits to be represented, so running time becomes exponential in the size of the input.

3. Approximation Algorithm

Step 1. Apply Greedy Algorithm
Step 2. Return $M_H = MAX(M_{greedy}, p_{MAX})$

2-Approximation

Let j be the first index at which *Greedy* Algorithm stops.

$$\overline{p}_j = \sum_{i=1}^{j-1} p_i$$

$$\overline{w}_j = \sum_{i=1}^{j-1} w_i$$

We can claim that $OPT < \overline{p}_j + p_j$.

Greedy Algorithm for fractional Knapsack $< \overline{p}_j + p_j$

Greedy Algorithm for {0,1} Knapsack < Greedy Algorithm for fractional Knapsack

OPT for {0,1} Knapsack $\leq$ Greedy Algorithm for fractional Knapsack $\leq \overline{p}_j + p_j$

Case 1:

$$p_j < \overline{p}_j \Rightarrow M_H = M_{Greedy} \Rightarrow OPT < M_{Greedy} < 2\overline{p}_j \Rightarrow OPT < 2\overline{p}_j$$

Case 2:

$$p_j > \overline{p}_j \Rightarrow p_{MAX} > \overline{p}_j \Rightarrow OPT < 2p_{MAX} = 2M_H \Rightarrow M_H \geq \frac{1}{2} OPT$$