A Fully polynomial Approximation Scheme for Integer Knapsack

K. Subramani
Department of Computer Science and Electrical Engineering,
West Virginia University,
Morgantown, WV
ksmani@csee.wvu.edu

1 Problem Statement

You are given a knapsack of capacity W and n objects $o_1, o_2, \ldots, o_n$ having weights $w_1, w_2, \ldots, w_n$ and profit values $p_1, p_2, \ldots, p_n$. The goal is to 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 constraint i.e. the sum of the weights of the objects in the knapsack is at most W. Associated with object o_i is a binary variable x_i, which is set to 1 if the o_i is in the knapsack and 0 otherwise. Accordingly, the knapsack (0/1) problem can be succinctly expressed as:

$$\max \sum_{i=1}^{n} p_i . x_i$$

$$s.t. \sum_{i=1}^{n} w_i . x_i \leq W, x_i \in \{0, 1\} \tag{1}$$

Integer Knapsack can be solved optimally using the Dynamic Programming technique used in [ACG$^+$00]. We call this algorithm EXACT-KNAPSACK; note that EXACT-KNAPSACK takes time proportional to $O(n. \sum_{i=1}^{n} p_i)$, which is exponential in the size of the input [1]. Our approximation algorithm consists of the following two steps:

1. Scale down the profit values, using an appropriate scaling factor;
2. Compute the optimal knapsack value, in the scaled problem, using EXACT-KNAPSACK.

We then argue that the subset corresponding to the scaled solution produces a value that is not very far away from the true optimal solution. Let $r > 1$ denote the degree with which we want to approximate the given instance.

Observation: 1.1 *We assume that the $w_i \leq W, \forall i$; otherwise we can discard the objects having weight greater than W.*

Observation: 1.2 $p_i \leq p_{max} \forall i = 1, 2, \ldots, n$

2 Analysis of approximation quality

The key point is that there is a loss in precision, as we compute the optimal solution in the scaled-down version of knapsack. Consider some arbitrary object o_i; because we are rounding down, $2^t . p'_i$ can be less than p_i but by at most 2^t. This is because $p'_i + 1 \geq \frac{p_i}{2^t} \Rightarrow 2^t . p'_i + 2^t \geq p_i$. Now focus on the optimal set O for the initial

[1]The size of the input to this problem is $\sum_{i=1}^{n} \log p_i$ while is a logarithmically related to $\sum_{i=1}^{n} p_i$.

```
Function APPROX-KNAPSACK $(p_1, \ldots, p_n; w_1, \ldots, w_n; , r, W)$
1: Let $t = \lfloor \log \frac{r-1}{r}.\frac{p_{max}}{n} \rfloor$, where $p_{max}$ is the profit value of the object with maximum profit.
2: for $(1 = 1$ to $n)$ do
3:     Compute $p'_i = \lfloor \frac{p_i}{2^t} \rfloor$
4: end for
5: Let $H$ be the solution returned by EXACT-KNAPSACK when provided with input $(p'_1, \ldots, p'_n; w_1, \ldots, w_n; , r, W)$
6: return$(H)$
```

Algorithm 1.1: A FPTAS for Knapsack

(unscaled) problem [2]. Let $p(O)$ indicate the profit of set O in the initial version i.e. $p(O)$ is represents the sum of the profits of the elements of set O. Likewise, let $p'(O)$ indicate the profit of set O in the scaled version of the problem. Each object in O could lose at most 2^t as a result of scaling down; there are at most n objects in O and hence we must have

$$2^t p'(O) + n.2^t \geq p(O)$$

Let H denote the set returned by the Dynamic Programming algorithm (1.1) on the scaled down version of the problem. In the scaled version, the profit of H is at least as good as the profit of O [3], i.e. $p'(H) \geq p'(O)$. This means that $2^t.p'(H) \geq 2^t.p'(O)$, which implies that $2^t.p'(H) \geq p(O) - n.2^t$. But $p(H) \geq 2^t.p'(H)$! Hence, $p(H) \geq p(O) - n.2^t$. Substituting for 2^t, we obtain, $p(H) \geq p(O) - n.\frac{r-1}{r}.\frac{p_{max}}{n} \Rightarrow p(H) \geq p(O) - p_{max}.\frac{r-1}{r}$. However, the optimal solution must be at least as large as p_{max} (See Observation (1.1). Thus, we can write $p(H) \geq p(O) - p(O)\frac{r-1}{r}$, which on simplification results in $p(H) \geq \frac{1}{r}p(O)$ i.e. the heuristic provides a solution whose value is at least $\frac{1}{r}$ of the optimal!

3 Analysis of Running Time

The running time of the Dynamic Programming algorithm on the scaled-down version of the problem is $O(n. \sum_{i=1}^{n} p'_i)$, which can be simplified to

$$O(n. \sum_{i=1}^{n} \frac{p_i}{2^t}) \tag{2}$$

Using Observation (1.2) and substituting for 2^t, we have

$$O(n. \sum_{i=1}^{n} n.\frac{p_{max}}{2^t}) = O(n^2.\frac{p_{max}}{\frac{r-1}{r}.\frac{p_{max}}{n}} = O(n^3.\frac{r}{r-1}) \tag{3}$$

Also see [ACG+00]. Observe that the running time is polynomial in n and $\frac{1}{r-1}$ and this is therefore an FPTAS (fully polynomial-time approximation scheme); as r tends to 1 the quality of the approximation improves; however the running time starts increasing.

References

[ACG+00] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation.* Springer-Verlag, 2000.

[2] It is important to note that the set O exists only for the purpose of analysis; we do not actually compute it!
[3] since H is the optimal set returned by EXACT-KNAPSACK()