1 The knapsack problem

In this report we consider the 0/1 knapsack problem which can be defined as follows: given n positive weights w_i, n positive profits p_i and a positive number b which is the knapsack capacity, this problem calls for choosing a subset $s \in [n]$ such that

$$max \sum_{i \in s} p_i x_i \qquad \text{subject to} \qquad \sum_{i \in s} w_i x_i \leqslant b$$

The x constute a 0/1 valued vector. To facilitate greedy procedures we can consider the items is sorted in nondecreasing order of *profit density*: $\frac{p_1}{w_1} \geqslant \frac{p_2}{w_2} \geqslant \dots \geqslant \frac{p_n}{w_n}$.

2 The backtracking approach

The backtrack procedure is a general technique used to solve combinatorial problems dealing with search for a set of solutions or an optimal solution satisfying some constraint.

In order to apply the backtrack method the desired solution must (a) be expressible as an n-tuple $(x_1, x_2, \dots, x_n)$ where x_i are chosen from some finite set S_i and (b) shares a property $P_n(x_1, \dots, x_n)$ such that $P_{k+1}(x_1, \dots, x_k, x_{k+1})$ implies $P_k(x_1, \dots, x_k)$ for $0 \leqslant k \leqslant n$.

The backtrack procedure consists of enumerating all solutions $(x_1, \dots, x_n)$ to the original problem by cosidering all partial solutions $(x_1, \dots, x_k)$ that satisfy P_k. If a partial solution $(x_1, \dots, x_k)$ does not satisfying P_k; hence by induction, no extended sequence $(x_1, \dots, x_k, \dots, x_n)$ can satisfy P_n.

A backtrack procedure for the knapsack problem can be developed using the linear relaxation of the problem for computing and upper bound [2]. The solution for linear relaxation of a given partial solution is given by the greedy algorithm (Algorithm 1).

Algorithm 1: Computes profit of LP-relaxation of a partial solution

Input : a partial solution $(x_1, \dots, x_k)$
Output: the LP upper bound p_{lp}

$b_{lp} \leftarrow b$;
$p_{lp} \leftarrow 0$;
for $i \leftarrow 1$ **to** k **do** `/* compute current profit and weight */`
  $b_{lp} \leftarrow b_{lp} + x_i \cdot w_i$; $p_{lp} \leftarrow p_{lp} + x_i \cdot p_i$;
end
while $b_{left} \geqslant w_i$ **do** `/* greedily filling knapsack */`
  $b_{lp} \leftarrow b_{lp} + w_i$; $p_{lp} \leftarrow p_{lp} + p_i$; $i \leftarrow i + 1$;
end
$p_{lp} \leftarrow p_{lp} + p_i . \frac{b_{lp}}{w_i}$ `/* fitting the split item */`

[3]

3 The Nemhauser-Ullmann algorithm

A brute force method to solve the knapsack problem is to enumerate all possible subsets over the n items. In order to reduce the search space, a domination concept can be used which is usually attributed to Weingartner and Ness [5].

Definition 1 (Domination) *A subset $S \in [n]$ with weight $w(S) = \sum_{i \in S} w_i$ and profit $p(S) = \sum_{i \in S} p_i$* ***dominates*** *another subset $T \subseteq [n]$ if $w(S) \leqslant w(T)$ and $p(S) \geqslant p(T)$.*

For simplicity assume that no two subsets have the same profit. Then no subset dominated by another subset can be an optimal solution to the knapsack problem, regardless of the specified knapsack capacity. Consequently, it suffices to consider those sets that are not dominated by any other set.

Using this concept Nemhauser and Ullmann [4] introduced an elegant algorithm (Algorithm 2) computing a list of all dominating sets in an iterative manner.

Algorithm 2: The Nemhauser-Ullmann Algorithm

Input : a KP instance
Output: list $S(n)$ of all dominating sets

1 $S(0) \leftarrow \emptyset$;
2 **for** $i \leftarrow 1$ **to** n **do**
3 $\quad S'(i) \leftarrow S(i-1) \cup \{ s \cup \{i\} \mid s \in S(i-1) \}$;
4 $\quad S(i) \leftarrow \{ s \in S'(i) \mid \mathsf{dominates}(s, S'(i)) \}$;
5 **end**

This algorithm can be viewed as a sparse dynamic programming approach which at each iteration duplicates all subsets in $S(i-1)$ and then adds item i to each of the duplicated subsets (line 3). The dominates procedure checks if subset s dominates all others subsets in $S'(i)$. The dominated subsets are then *filtered* (line 4). The result is the ordered sequence $S(n)$ of dominating subsets over the items $1, \ldots, n$.

Figure 1 graphically represents profits and weights for dominating sets of (a) an intermediate solution $S(i)$, (b) its next solution $S(i+1)$ and (c) an optimal solution for a small random instance.

If we denote $q(i)$ the upper bound on the number of dominating sets over items in $1, \ldots, i$, at each iteration the algorithm computes $S(i)$ over $S(i-1)$ in $O(q(i))$ time. The total running time of the algorithm is then $O(n \cdot q(n))$. Now the challenge in the analysis is to estimate the number of dominating sets.

Beier and Vöcking [1] addressed this analysis considering sets of items with adversary weights and randomly drawn profits. They could deduce that for the uniform distribution, for example, the expected number of dominating sets is $E[q] = O(n^3)$ leading to an expected running time of $O(n^4)$.

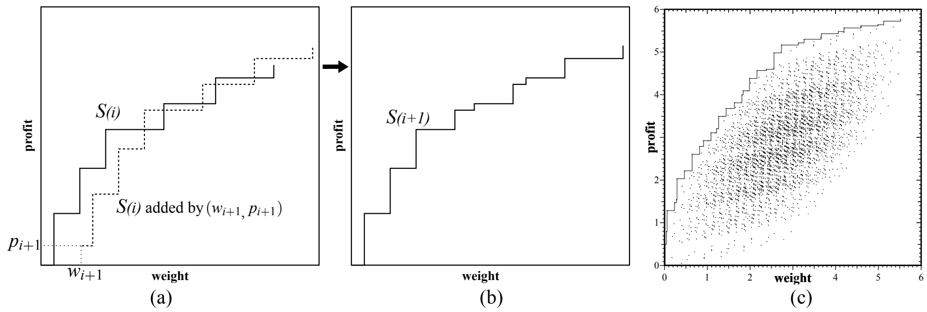

Figure 1: Graphical representation of dominating sets for (a) an intermediate solution $S(i)$, (b) its next solution $S(i+1)$. and (c) an optimal solution for a small random instance.

References

[1] R. Beier and B. Vöcking. Random knapsack in expected polynomial time. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 232–241. ACM, 2003.

[2] E. Horowitz and S. Sahni. *Fundamentals of computer algorithms.* Computer Science Press, 1978.

[3] G. S. Lueker. Average-case analysis of off-line and on-line knapsack problems. *Journal of Algorithms*, 29(2):277–305, 1998.

[4] G. L. Nemhauser and Z. Ullmann. Discrete dynamic programming and capital allocation. *Management Science*, 15(9):494–505, 1969.

[5] H. M. Weingartner and D. N. Ness. Methods for the solution of the multidimensional 0/1 knapsack problem. *Operations Research*, 15(1):83–103, 1967.