A brief survey on two exact algorithms for the knapsack problem

Marcos Daniel Baroni

November 19, 2014

This brief study was inspired by a comment about backtracking approach for knapsack problem by Lueker in [2] saying:

> "*When applied to randomly generated data, this approach, which always yields the exact optimum, seems to run very rapidly even for large values of N; in fact, it seems possible that its expected time is polynomial in N. A proof of this would be very interesting, but probably difficult*".

The knapsack problem is briefly introduced in Section 1. The backtrack algorithm for the knapsack problem and some experimental results is presented in Section 2. A sparse dynamic programming approach for the knapsack problem is presented in Section 3.

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

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 $(x_1, x_2, \ldots, x_n)$ where x_i are chosen from some finite set S_i and (b) shares a property $P_n(x_1, \ldots, x_n)$ such that $P_{k+1}(x_1, \ldots, x_k, x_{k+1})$ implies $P_k(x_1, \ldots, x_k)$ for $0 \leqslant k \leqslant n$.

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

In the case of knapsack problem, each partial solution $(x_1, \ldots, x_k)$ can be viewed as a subproblem whose variables $x_1, \ldots, x_k$ are fixed while $x_{k+1}, \ldots, x_n$ are free. For each subproblem a relaxed version can be derived replacing the integrality contraints of the free variables by the linear relaxation constraint $x_i \in [0, 1]$. This relaxation is called *linear relaxation*. The linear relaxation can be used to prune its respective partial solution: whenever its optimal value is not better than the best optimal solution found so far, the respective partial solution is pruned and the algorithm backtracks.

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 $(x_1, \ldots, x_k)$
Output: the LP upper bound p_{lp}

```
1 b_lp ← b;
2 p_lp ← 0;
3 for i ← 1 to k do                          /* compute current profit and weight */
4 |  b_lp ← b_lp + x_i · w_i; p_lp ← p_lp + x_i · p_i;
5 end
6 while b_left ⩾ w_i do                      /* greedily filling knapsack */
7 |  b_lp ← b_lp + w_i; p_lp ← p_lp + p_i; i ← i + 1;
8 end
9 p_lp ← p_lp + p_i · b_lp / w_i             /* fitting the split item */
```

Figure 1 shows experimental results on the number of partial solutions computed by the backtrack algorithm. The continuous line is the average number over 1000 random instances, the dotted line is the total number of possible solutions and the dashed line is the conjectured curve for the observed experimental result; (a) shows the number of computed solutions to find the optimal solution and (b) shows the number of computed solutions to prove its optimality.

3 The Nemhauser-Ullmann algorithm

In order to reduce the search space of a knapsack problem instance, a domination concept can be used which is usually attributed to Weingartner and Ness [4].

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 [3] 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 $\quad S'(i) \leftarrow S(i-1) \cup \big\{s \cup \{i\} \,\big|\, s \in S(i-1)\big\}$;
4 $\quad S(i) \leftarrow \big\{s \in S'(i) \,\big|\, \mathsf{dominates}\big(s, S'(i)\big)\big\}$;
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 2 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 $O\big(q(i)\big)$ time. The total running time of the algorithm is then $O\big(n \cdot q(n)\big)$. 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 $O(n^4)$.

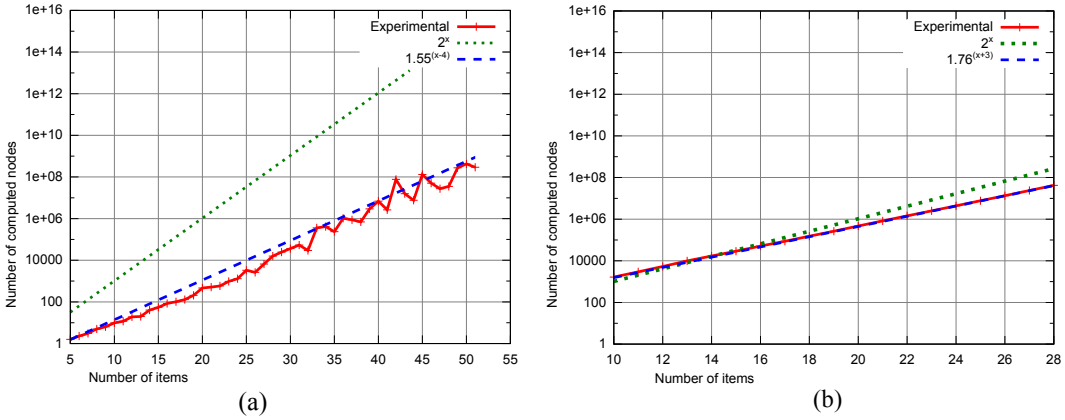

Figure 1: Number of computed nodes (partial solutions) (a) to find the optimal solution and (b) to prove its optimality.

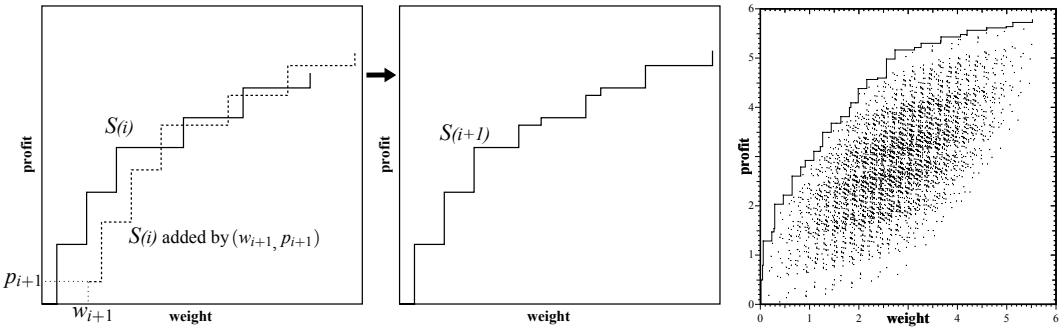

Figure 2: Graphical representation of dominating sets for (a) an intermediate solution $S(i)$, (b) its next solution $S(i+1)$. and (c) an optimal solution for a small random instance.

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] G. S. Lueker. *On the average difference between the solutions to linear and integer knapsack problems.* Springer, 1982.

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

[4] 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.