39

The Knapsack Problem

Rene Beier and Berthold Vöcking

Max-Planck-Institut für Informatik, Saarbrücken, Germany
RWTH Aachen, Aachen, Germany

In two months, the next rocket will leave Earth heading to the space station. The space agency, being a little short of money, offers to carry out scientific experiments for other research institutions at the space station. For each experiment, some equipment needs to be lifted up to the station. However, there is a limit on the weight the rocket can carry. Apart from the obligatory food rations, the rocket is able to carry up to 645 kg of scientific equipment. The space agency receives several offers from different research institutions stating how much they are willing to pay for the execution of an experiment and specifying the weight of the necessary equipment. The space agency wants to figure out which of these experiments should be chosen in order to maximize the profit.

This scenario exemplifies a classic optimization problem, the so-called knapsack problem: Suppose we have a knapsack that has a certain capacity described by a weight threshold T. We are given a set of n items, each bearing a weight and a profit. The task is to choose a subset of the items that should go into the knapsack. Only subsets of total weight at most T can be chosen. The objective is to make the profit as large as possible. That is, one seeks for a subset such that the sum of the profits of the items in the subset is as large as possible under the constraint that the sum of the weights of these items is at most T.

In the introductory example the knapsack corresponds to the rocket. The weight threshold is $T = 645$. The items correspond to the experiments. To make the example more concrete we assume that the space agency can choose from $n = 8$ experiments listed with the following weights and profits.

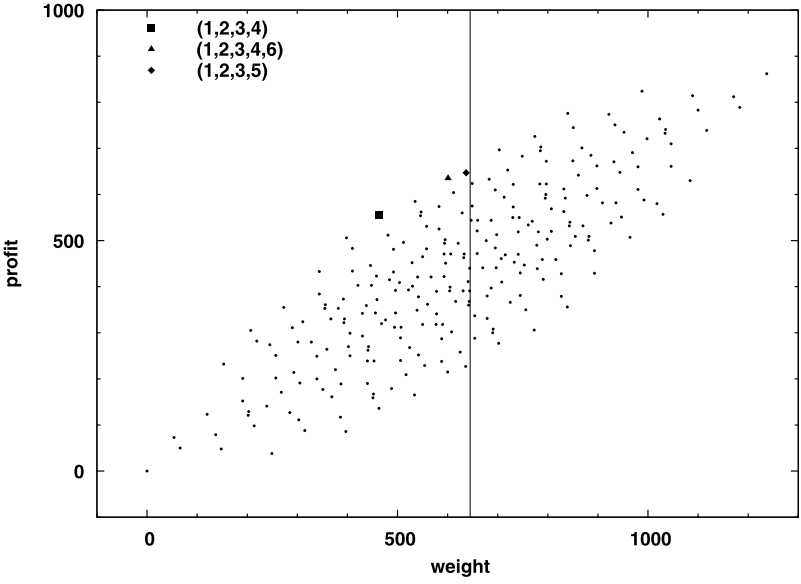
Item	1	2	3	4	5	6	7	8
Weight in kg	153	54	191	66	239	137	148	249
Profit in 1000 Euro	232	73	201	50	141	79	48	38
Profit density	1.52	1.35	1.05	0.76	0.59	0.58	0.32	0.15

B. Vöcking et al. (eds.), *Algorithms Unplugged*,
DOI 10.1007/978-3-642-15328-0_39,

How can we find a feasible set of items that maximizes the profit? Intuitively we could choose those items first that achieve the largest profit per unit weight. This ratio between profit and weight is called profit density. We have calculated the profit density for all items. The table lists the items from left to right in such a way that this ratio is decreasing.

Our first algorithm follows the idea above and sorts all items by decreasing profit density, e.g., by using the algorithms from Chap. 2 or Chap. 3. Starting with the empty knapsack we add items one by one in this order as long as the capacity of the knapsack is not exceeded. In our example, we would pack items $1, 2, 3$, and 4 as their cumulative weight is 464, which is still below the threshold. Adding item 5 would result in a total weight of $464 + 239 = 703$, which would overload the knapsack. The four packed items yield a profit of 556. Is this the maximal profit that we can achieve? Not quite. By adding item 6 we obtain a knapsack packing with a total weight of 601 that is still feasible and has a total profit of 635. Is this now the most profitable subset? Unfortunately not.

Of course, in order to guarantee optimality we could test all possible combinations of packing the knapsack. To better illustrate let us draw all possible packings in a weight–profit diagram, irrespective of the weight threshold. For instance, we draw the point $(601, 647)$ for the packing with items $\{1, 2, 3, 4, 6\}$.



Each point represents a subset of items. How many points do we have to draw? We must decide whether to include each item or not; hence there are two possibilities per item. As the choice for each item is independent from the choice made for the other items, there are 2^n possibilities for n items. Thus, in our example, we would need to test $2^8 = 256$ different packings.

The packings with weights greater than the threshold T are not feasible. These subsets correspond to points in the diagram that lie to the right of the vertical line. Points to the left of or exactly on the vertical line correspond to feasible packings. Among those feasible packings we choose the one with maximum profit. In our example, it is the point with coordinates 637 and 647, corresponding to the packing containing items $1, 2, 3$, and 5. This is the *optimal solution*.

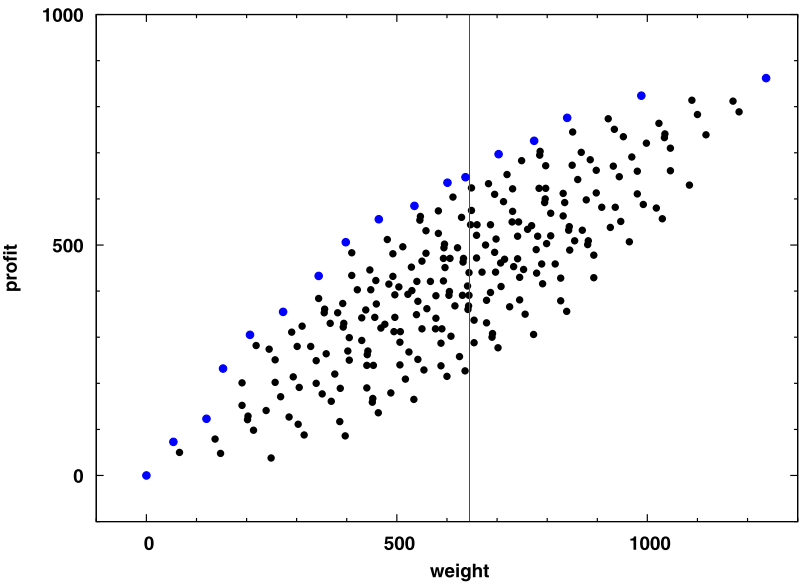
This approach of finding the best solution is practical only for a small number of items as the computational effort increases rapidly when increasing the number of items. Each additional item doubles the number of packings that need to be tested. When the space agency, for example, can choose among 60 experiments the number of possible packings grows to

$$2^{60} = 1{,}152{,}921{,}504{,}606{,}846{,}976.$$

Assuming that a modern computer can test 1,000,000,000 packings per second, it would still take more than 36 years to finish the computation. Of course, we do not want to delay the start of the rocket for such a long time.

Pareto-Optimal Solutions

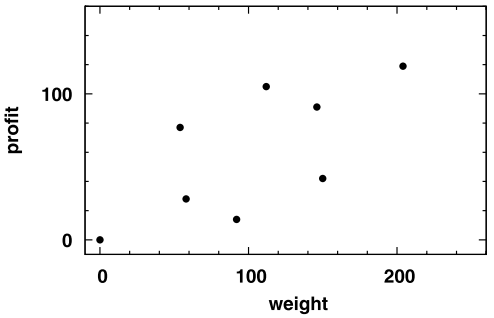
How can we find the optimal solution faster? The key to a more efficient algorithm is the following observation: A packing cannot be optimal if there exists another packing with lower weight and higher profit. Let's go back to our example.



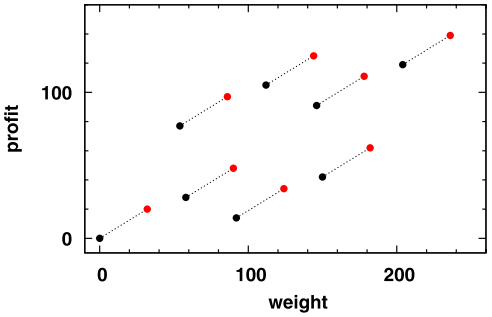
None of the black points can be the optimal solution since for each of them we can find at least one other point that is better, i.e., that has both lower weight and higher profit. We say that this better point *dominates* the black point. The blue points in the diagram are exactly those points that are not dominated by any other point. These points are called *Pareto-optimal*. Hence, a packing is Pareto-optimal if there exists no other packing of less weight that yields a higher profit. In our example, only 17 of the 256 packings are Pareto-optimal. The optimal solution must be among those 17 packings. Observe that the property of being Pareto-optimal is independent of the choice of the weight threshold. In particular, for any given threshold we can find the optimal solution among those 17 packings.

So far we have ignored in the discussion the fact that two packings can have the same weight or the same profit. In order to correctly handle those cases we define that packing A dominates packing B if A is at least as good as B in weight and in profit and additionally if A is strictly better than B in at least one of the two criteria.

But how can we compute a list of all Pareto-optimal packings efficiently, namely without testing all 2^n packings? Consider the following small example starting with three items. We plotted all $2^3 = 8$ different packings in a weight–profit diagram.



Let us assume that we know the set of Pareto-optimal packings for these three items, and we consider an additional fourth item. For each of the eight packings we can generate a new packing by adding the fourth item. This way, we obtain eight additional packings. Each black point generates a new red point with a horizontal and vertical shift corresponding to the weight and profit of the fourth item. Hence, the set of red points is just a shifted copy of the set of black points.



What can we say about the Pareto-optimality of these 16 points? We exploit that we already know the set of Pareto-optimal packings for three items. A black point dominated by some other black point is by definition not Pareto-optimal. The same holds for each red point dominated by some other red point. In other words, a black point that is not Pareto-optimal with respect to the black point set (3 items) cannot become Pareto-optimal with respect to all 16 points (4 items). The same applies to red points that are not Pareto-optimal with respect to the red-point set. As a consequence, only points from the following two sets can potentially be Pareto-optimal.

A: black points that are Pareto-optimal with respect to the black point set, and
B: red points that are Pareto-optimal with respect to the red point set.

Observe that the points in B are just shifted copies of the points in A. Now consider a point p from A. Suppose p is dominated by a red point q. If q does not belong to B then there must be a red point q' in B that dominates q and thus dominates p as well. Hence, in order to check the Pareto-optimality of a point from A (with respect to both black and red points), one only needs to check if this point is not dominated by a point from B. Analogously, in order to check the Pareto-optimality of a point from B, one only needs to check if it is not dominated by a point from A.

Now we have a procedure for computing the set of Pareto-optimal packings when adding one additional item: First construct all red points that are generated from Pareto-optimal black points. Then delete all red points dominated by a black point. Finally, delete all black points dominated by a red point.

The Nemhauser–Ullmann Algorithm

The following algorithm was invented by Nemhauser and Ullmann in 1969. It uses the arguments above in an iterative fashion, adding one item after the other. That is, it starts with the empty set of items and adds items one by one until it finally obtains the set of Pareto-optimal packings for all n items.

An efficient implementation maintains the set of Pareto-optimal points in a list that is sorted according to the weight of the points. The initial list L_0 contains only the point $(0, 0)$, which corresponds to the empty packing. Now we iteratively compute lists $L_1, L_2, \ldots, L_n$, where L_i denotes the list of Pareto-optimal points with respect to items 1 to i.

Using L_{i-1} and the ith item we compute L_i as follows. First we generate L'_{i-1} (the red point set), which is a shifted copy of L_i (the black point set). Each point in L_{i-1} has to be copied and shifted right and up by the weight and profit of the ith item, respectively. Now we merge the two lists, L_{i-1} and L'_{i-1}, filtering out points that are dominated. As both lists are sorted according to the weight of the points (and therefore also according to the profit – can you tell why?), this task can be achieved by scanning only once through both of these lists. Thus, the time needed to merge these lists is linear in the sum of the length of the two lists.

The algorithm MERGE merges two sorted lists of points L and L'.

```
1  procedure MERGE (L, L')
2  BEGIN
3     PMAX = −1; E = {}
4     REPEAT
5        Scan L for a point (w, p) with profit p > PMAX
6        Scan L' for a point (w', p') with profit p' > PMAX
7        If no point has been found in line 5 (finished scanning L)
8           Insert remaining points from L' into E; RETURN(E)
9        If no point has been found in line 6 (Finished scanning L')
10          Insert remaining points from L into E; RETURN(E)
11       IF (w < w') OR (w = w' AND p > p')
12          THEN insert (w, p) into E and set PMAX := p
13          ELSE insert (w', p') into E and set PMAX := p'
14 END
```

The resulting list L_n contains all Pareto-optimal points with respect to the n items. From this list, we choose the point with maximal profit whose weight is at most T. The packing belonging to this point is the optimal solution.

Is this algorithm better than simply testing all possible packings? Not in every case, as it is possible that all 2^n packings are Pareto-optimal. This can happen, for instance, when the profit density of all items has the same value c, i.e., $p_i = c * w_i$ for some constant c. However, this is not what one typically experiences. Usually, the number of Pareto-optimal solutions is much smaller than 2^n. In our introductory example we generated weights and profits of the eight items at random. Only 17 out of the 256 packings are Pareto-optimal. Mathematical and empirical analyses show that typically only a very small fraction of the packings are Pareto-optimal. For this reason, the described algorithm can handle instances of the knapsack problems with thousands of items in a reasonable amount of time.

Further Reading

1. H. Kellerer, U. Pferschy and D. Pisinger: *Knapsack Problems*. Springer, 2004.
   This nice textbook is completely devoted to algorithms for the knapsack problem. It discusses various practical variations of this problem and presents several algorithmic approaches to tackle these problems. It gives a comprehensive overview of the state of the art in this field.
2. S. Martello and P. Toth: *Knapsack Problems: Algorithms and Implementations*. Wiley, Chichester, 1990.
   This is an older textbook about the knapsack problem; nevertheless it gives some interesting insights into different approaches for solving the problem and its variations.
3. Wikipedia elaborates on the knapsack problem, too. In particular, it presents an algorithm using the dynamic programming paradigm. (An introduction to this paradigm applied to a different problem can be found in Chap. 31 of this book.) The same algorithm can be found in several introductory textbooks about algorithms as well. In many instances, however, it is much slower than the algorithm that we have presented here:
   http://en.wikipedia.org/wiki/Knapsack_problem
4. The knapsack problem is closely related to bi- or multicriteria optimization problems which optimize two or more criteria simultaneously. For example, one is given n objects, each of which comes with a profit and a weight, like in the knapsack problem, but there is no threshold on the weight. Instead one assumes that there is a decision-maker that, on the one hand, seeks for a subset of items giving a large profit. On the other hand, the decision-maker wants to select a subset of low weight. Of course, these are conflicting goals and the question is how to resolve them. Such problems arise in many practical applications. For example, consider a navigation system for traveling by car. (Such a system uses algorithms similar to the shortest-path algorithm explained in Chap. 32.) The objective might be to find a short route between a starting point and a destination in a traffic network. The shortest route, however, is not always the quickest one, as it might directly lead through the city rather than taking a relatively short detour along the highway on which one can travel much faster. Here to every route could be attached two values, distance and time. The interesting solutions are the Pareto-optimal ones with respect to these two criteria among which the driver should make his choice. An entry point into the field of Pareto optimization can be found in Wikipedia:
   http://en.wikipedia.org/wiki/Pareto-efficiency