Exercise: A Dynamic Programming Algorithm for the Knapsack Problem

Introduction to Optimization lecture
at Ecole Centrale Paris / ESSEC Business School

Dimo Brockhoff
firstname.lastname@inria.fr

November 2, 2015

Abstract

In the lecture, we have seen the general concept of *dynamic programming* and it is the purpose of this exercise to apply it to the knapsack problem. We are going to not only formally define the algorithm but also implement it (like in the previous exercise preferably in python).

1 Part I: Dynamic Programming for the Knapsack Problem

We want to develop and implement an exact algorithm for the knapsack problem based on the idea of dynamic programming. Before you actually implement the algorithm, answer the following questions about the dynamic programming formulation of the problem first:

a) What are potential subproblems here? Tip: potential constraints to restrict the overall problem to subproblems are naturally based on the problem's parameters, i.e., the number or types of the items, the profits, and the weights.

b) How do you construct solutions to larger subproblems from already computed solutions of smaller subproblems? Write down the Bellman equation.

c) How do you solve the smallest problems (initialization)?

Finally, implement the algorithm for the knapsack problem and test it. To this end, follow the tasks below.

d) Implement a first, naive, recursive approach to solving the knapsack problem using the Bellman equation from part b).

e) Implement the same functionality in terms of a dynamic programming algorithm which solves the subproblems of the Bellman equation in a bottom-up approach from the smallest subproblems to the largest while storing the optimal function values of the subproblems in a matrix/an array and not solving a subproblem more than once.

f) Test your implementations of the recursive and the dynamic programming algorithm on a few example instances which you can find via the lecture's web page at `researchers.lille.inria.fr/~brockhof/introoptimization/knapsackinstances/`.

g) To this end, compare the output of the two algorithms with the exact algorithm from the previous exercise on the provided knapsack instances (the smallest instances will suffice). In particular check whether both the brute-force and the dynamic programming approach result in the same optima (and in particular the same optimal values). Why is the latter more important to test?

h) Finally also compare the times, the algorithms need to solve the provided instances. When looking at the influence of the problem dimension (i.e. the number of items), can you make predictions about larger, yet un-tested instances?