Optimization Methods in Finance **(EPFL, Fall 2010)**

Lecture 10: The knapsack problem

24.11.2010

Lecturer: Prof. Friedrich Eisenbrand *Scribe: Anu Harjula*

The knapsack problem

The *Knapsack problem* is a problem of how to choose items to maximize their value under a constraint of maximal weight. Let's assume that we can choose from items $1,\ldots,n$ with weights $a_1,a_2,\ldots,a_n$ and profits $p_1,p_2,\ldots,p_n$. The capacity of the knapsack $K \in \mathbb{N}$ is also given. The task is now to select a subset of the items so that its total weight does not exceed K and its profit is maximized among those subsets.

The integer program formulation of the knapsack problem is the following.

$$\forall i = 1,\ldots,n \qquad \text{we have a variable } x_i \in \{0,1\}$$

$$x_i = \begin{cases} 1 & \text{"pick the item"} \\ 0 & \text{"don't"} \end{cases}$$

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

$$\text{s.t.} \sum_{i=1}^{n} a_i x_i \leq K$$

$$x_i \in \{0,1\} \quad \forall i = 1,\ldots n$$

We can see that by replacing the last constraint with $0 \leq x_i \leq 1$ we would get a linear program.

Example 1 (A capital budgeting problem)**.** We want to invest \$19'000 to different unsplittable opportunities and we aim, of course, to maximize our profit without exceeding the budget. The fact that the investments cannot be split makes this a knapsack problem. Suppose that we know the investments needed and their net present values. These are presented in Table 1.

Table 1: Investments and net present values of the opportunities

opportunity	investment	net present value
1	6'700	8'000
2	10'000	11'000
3	5'500	6'000
4	3'800	4'000

The mathematical formulation of this knapsack problem is (the numbers are thousands)

$$\begin{aligned} \max\ & 8x_1 + 11x_2 + 6x_3 + 4x_4 \\ \text{s.t.}\ & 6.7x_1 + 10x_2 + 5.5x_3 + 3.8x_4 \leq 19 \\ & x_1, x_2, x_3, x_4 \in \{0,1\}. \end{aligned}$$

In other words, we maximize the profit under the budget constraint. Here again, $x_1 = 1$ means that you choose investment 1.

Dynamic program for knapsack problem

Now we will construct a dynamic program for solving the knapsack problem. Let's introduce variable $W_i(p)$ which denotes the smallest possible weight of a subset of the items $1,\ldots,i$ that has a total profit p. These values are the shortest path distances in a graph with a starting node s. The graph is constructed such that each node has two values i and p and from node (i,p) there exist two arcs to nodes $(i+1,p)$ and $(i+1,p+p_{i+1})$ as illustrated in Figure 1. The weights of the arcs are 0 (meaning that we don't choose this item) and a_{i+1} respectively.

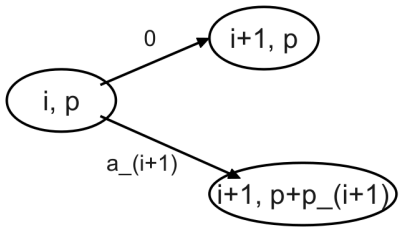

Figure 1: Part of the graph

The total number of nodes in this graph is $\#(nodes) \leq n \cdot (n \cdot p_{max}) = n^2 \cdot p_{max}$, and the number of arcs $\#(arcs) \leq 2n^2 \cdot p_{max}$, where p_{max} denotes the largest profit of p_i $i = 1,\ldots,n$. The graph is directed and acyclyc (no cycles) and the starting node is $s = (0,0)$.

Theorem 2. *The knapsack problem can be solved in time $O(n^2 \cdot p_{max})$.*

Proof. We start by computing the shortest path labels (weights) from s to all other nodes. A node is feasible if the weight of the path leading to it doesn't exceed the capacity K of the knapsack. The shortest path to a feasible node with the largest second component p (profit) represents the optimal solution. □

Let's consider the following example to impliment the algorithm for the knapsack problem.

Example 3.

$$\begin{aligned} \max\ & 2x_1 + 2x_2 + x_3 \\ s.t.\ & 2x_1 + x_2 + 2x_3 \leq 4 \\ & x_1, x_2, x_3 \in \{0,1\}. \end{aligned}$$

To solve this knapsack problem we construct the graph presented in Figure 2.

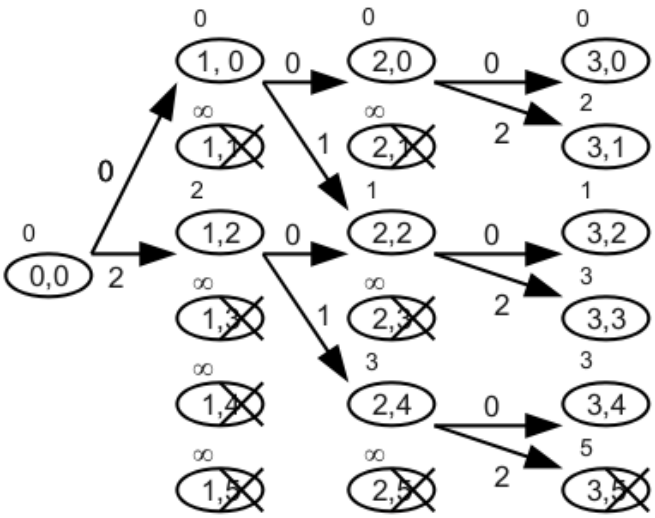

Figure 2: Solving the knapsack problem

First we draw all the nodes that we might need. Then, starting from the node $(0,0)$, we add arcs to those nodes that are reachable. In this case we can either choose item 1 (arc to node (1,2) with weight 2) or NOT choose it (arc to node (1,0) with weight 0). Choosing item 1 leads us to node (1,2) because item 1 has a profit of $p_1 = 2$. Applying this rule for the other items, we obtain the arcs illustrated in figure 2. The number next to an arc denotes the weight of the item chosen (or 0 if we did not choose it).

Next we add the labels denoting the weight of the shortest path from the starting node to the node in question. These labels are marked with the smaller numbers above the nodes. For the unreachable nodes we give label value ∞. Now we can delete the nodes that have a label value larger than the capacity of the knapsack (4 in this case). This way we have only the possible nodes to work with.

From these feasible nodes we now search the one with the largest second component (the profit) and this is the optimal solution to the knapsack problem. Here the largest profit is 4 and we reach it at nodes (2,4) and (3,4), which actually mean the same solution. From the shortest path leading to these nodes we can conclude that we must choose items 1 and 2 ($\overline{x}$=(1,1,0)).

The knapsack problem cannot be efficiently solved in a complexity theoretic sence, meaning in polynomial time, unless we have equivalence between problem classes $P = NP$. Thus, it would be useful to find a reasonable approximation.

Approximation of the knapsack problem

For a given $\varepsilon > 0$, our goal is to, find a feasible solution $\overline{x}$ of the knapsack problem

$$\begin{aligned} &\max\ p^T x \\ &s.t.\ a^T x \leq K \qquad\qquad \text{(I)} \\ &\quad i = 1,\ldots,n:\ x_i \in \{0,1\} \end{aligned}$$

so that $(1+\varepsilon)p^T\overline{x} \geq p^T x$ for every feasible x. We also want the algorithm to be polynomial in $\frac{1}{\varepsilon}$ and n.

This is possible to achieve as follows. Suppose that

$$2^{\ell-1} < p_{\max} \leq 2^{\ell} \quad \text{for some } \ell \in \mathbb{N}.$$

And let's rewrite p as a sum of two parts

$$p = 2^k p_1 + p_2 \quad \text{with } p_1, p_2 \in \mathbb{N}_0^n \quad \text{s.th. } \|p_2\|_\infty \leq 2^k - 1.$$

The solution to the knapsack problem

$$\begin{aligned} &\max\ p_1^T x \\ &\text{s.t.}\quad a^T x \leq K \qquad\qquad \text{(II)} \\ &\quad i = 1,\ldots,n:\ x_i \in \{0,1\}. \end{aligned}$$

can be found in time $O(2^{\ell-k}\cdot n^2)$.

Now we can compare the optimal solutions of (I) and (II). Let $\hat{x}$ and $\overline{x}$ be the optimal solutions of (I) and (II) respectively. For the ratio of these two objective functions we obtain the following.

$$\begin{aligned} \frac{p^T\hat{x}}{p^T\overline{x}} &= \frac{p^T(\hat{x}-\overline{x}) + p^T\overline{x}}{p^T\overline{x}} \\ &= 1 + \frac{p^T(\hat{x}-\overline{x})}{p^T\overline{x}} \\ &\leq 1 + \frac{p^T(\hat{x}-\overline{x})}{(p_1)_{\max}\cdot 2^k} \quad (*) \\ &\leq 1 + \frac{p^T(\hat{x}-\overline{x})}{2^{\ell-1}} \\ &= 1 + \frac{2^k\cdot p_1^T(\hat{x}-\overline{x}) + p_2^T(\hat{x}-\overline{x})}{2^{\ell-1}} \\ &\leq 1 + \frac{n\cdot 2^k}{2^{\ell-1}} \end{aligned}$$

In $(*)$, use $p_1^T \overline{x} \geq (p_1)_{\max}$ and

$$p^T \overline{x} = (2^k p_1 + p_2)^T \overline{x} \geq 2^k p_1^T \overline{x} \geq 2^k (p_1)_{\max}.$$

We want to know when this ratio is less or equal to $1+\varepsilon$, and thus we get

$$\frac{n \cdot 2^k}{2^{\ell-1}} \leq \varepsilon$$

which can also be stated as

$$\frac{2^{\ell-k-1}}{n} \geq \frac{1}{\varepsilon}.$$

We have now shown that the running time of (II) is polynomial in $\frac{1}{\varepsilon}$ and n. This is actually a very good approximation.

Integer programming & Branch and bound

An integer program (IP) is a problem of the form

$$\begin{aligned} &\max c^T x \\ &s.t.\ Ax \leq b \\ &\quad x \text{ is integer} \end{aligned}$$

Example 4 (Combinatorial auctions)**.** An auctioneer is selling items $M = \{1, ..., m\}$. A *bid* is a pair $B_j = (s_j, p_j)$, where $s_j \subseteq M$ is a subset of the items and $p_j \in \mathbb{R}$ is the price. The question needed to be answered is: How should the auctioneer determine the winners and the loosers of the bidding in order to maximize his revenue?

Let's consider an example case of the combinatorial auctions.

$$\begin{aligned} & M = \{1,2,3,4\} \\ \text{Bids:} \quad & B_1 = (\{1\},6), \qquad B_2 = (\{2\},3) \\ & B_3 = (\{3,4\},12), \quad B_4 = (\{1,3\},12) \\ & B_5 = (\{2,4\},8), \quad B_6 = (\{1,3,4\},16) \end{aligned}$$

We now define a variable x_i as follows.

$$x_i = \begin{cases} 0, & \text{if bid } i \text{ is not served} \\ 1, & \text{if bid } i \text{ is served} \end{cases}$$

Bid i being served means that the bidder gets the items.

Now we can write an integer program to maximize the revenue of the auctioneer. The constraints describe the fact that every item can be sold to only one bidder.

$$\begin{aligned} \max\ & 6x_1 + 3x_2 + 12x_3 + 12x_4 + 8x_5 + 16x_6 \\ s.t.\ & x_1 + x_4 + x_6 \leq 1 \\ & x_2 + x_5 \leq 1 \\ & x_3 + x_4 + x_6 \leq 1 \\ & x_3 + x_5 + x_6 \leq 1 \\ & x_i \in \{0,1\},\ i = 1,...,6 \end{aligned}$$

Next time we will discuss how this can be solved.