Contents

1. Lecture 5

When a problem is (NP-hard), we might be happy with just an approximation.
The approximation should then at least be quick (polynomial time) and it would be good to know that we are never too far away from the optimal value.

Definition

$$\alpha_A = \max_I \frac{A(I)}{OPT(I)}$$

This so-called approximation ratio of algorithm A gives the maximum deviation than can be observed between the value of a solution found by A and the optimum value.

We will learn how to compute approximation ratios algorithms **without being able to actually compute the optimum**.

Under the assumption that $N \neq NP$:

1. There are NPC problems for which no efficient algorithm exists with a finite approximation ratio
2. There are NPC problems where the approximation ratio is finite but increases with the size of the instance
3. There are NPC problems for which finite approximation ratios are possible (but not arbitrarily small)
4. There are NPC problems for which you can specify a value for the desired performance ratio, and an approximation algorithm can be found that yields it.

1.0.1 Knapsack Problem 0-1

Given a set of n items with:

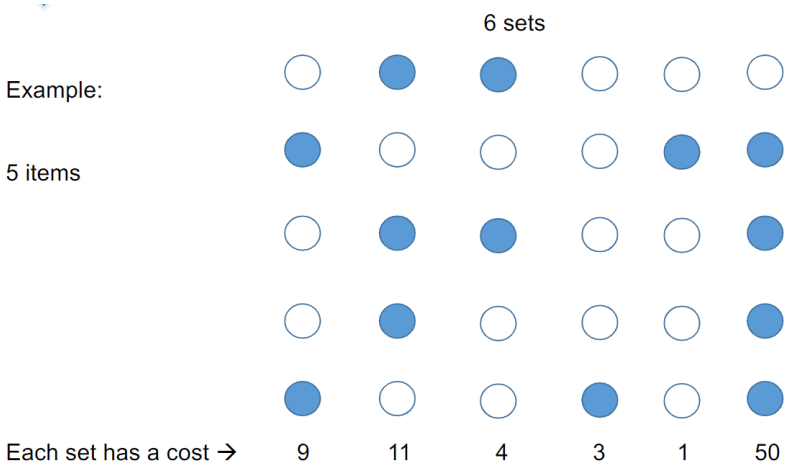
- a profit p_j
- a weight w_j

And given also a knapsack having capacity c able to carry each individual item.
Goal: choose a subset of items that fits into the knapsack with maximum total profit. In mathematical term, we want:

$$\max\left(\sum_{j=1}^{n} p_j x_j\right) \text{ such that } \sum_{j=1}^{n} w_j x_j \leq c \text{ with } x_j \in \{0,1\}$$

1.0.2 Set Covering Problem

Given m items and a collection of n sets each containing some of those items. There is a cost c_j for using set j.
Choose sets with minimum total cost which together include (cover) all items



In mathematical term, we want:

$$\min\left(\sum_{j=1}^{n} c_j x_j\right) \text{ such that } \sum_{j=1}^{n} a_{ij} x_j \geq 1 \text{ with } x_j \in \{0,1\} \text{ and } 1 \leq i \leq m$$

Or

$$\min c'x \text{ such that } Ax \geq 1 \text{ with } x \in \{0,1\}^n$$

1.0.3 Traveling Salesman Problem

Given n locations and the cost c_{ij} of traveling from each location i to each location j with $1 \leq i, j \leq n$ find a sequence of all locations that minimizes the total travel cost including the return to the departure location.

We will find our inspiration (plus a valuable analysis instrument) for a greedy algorithm for the binary Knapsack by first looking at a slightly different version of the problem, the fractional Knapsack problem:

$$\max\left(\sum_{j=1}^{n} p_j x_j\right) \text{ such that } \sum_{j=1}^{n} w_j x_j \leq c \text{ with } 0 \leq x_j \leq 1$$

Turn out that fractional Knapsack is much easier than $0-1$ Knapsack. The following is optimal:

1. Order the variables in non-increasing order of $\frac{p_i}{w_j}$
2. Let k be the first variable for which $\sum_{j=1}^{k} w_j > c$ with the sum taken in the order found in the previous step
3. Set $x_j = 1$ for all $j < k$, $x_k = \frac{c - \sum_{j=1}^{k-1} w_j}{w_k}$ and $x_i = 0$ for all other i.

The ratio p_j/w_j indicates the efficiency of selecting item j.Let's use this in the 0-1 setting as well: We start with the item of highest efficiency and add items in non-increasing efficiencies for as long as items fit. Call this algorithm GreedyKP01.

Show that for any given large positive number L you can create a simple KP01 instance with just 2 items where GreedyKP01 finds a solution with a value L times smaller than the optimum value.
That is, fill the following matrix and give a value for c (capacity of the knapsack).

L is a large number, and for example:

Item	Profit	weight
1	1	0.5
2	L	L

And $c = L$

Clearly the optimum is $(0,1)$ (take the second item)
with profit L, yet greedy selects $(1,0)$ with profit 1.

Improved greedy for KP01.

1. Let x be the greedy solution
2. Let y be the solution consisting only of the first item that does not fit in the knapsack when the solution is constructed by greedy
3. Take the best of the two solutions

.
Let's work this out for the following example: and knapsack capacity = 7

Theorem The optimum value of a KP01 instance can never be more than twice the value of the solution found by the algorithm above.
In other words: it has an approximation ratio of 0.5