COMP523 Tutorial 4 - Solutions

Coordinator: Aris Filos-Ratsikas Demonstrator: Michail Theofilatos

December 27, 2019

Problem 1

Consider the *fractional knapsack problem*, in which there is a set of n *infinitely divisible* items with values v_i, for $i = 1, \ldots, n$ and weights w_i, for $i = 1, \ldots, n$, and there is a total weight constraint W. The goal is to find fractions $(x_1, \ldots, x_n)$ of each item, with $0 \leq x_i \leq 1$ such that $\sum_{i=1}^n x_i \cdot v_i$ is maximised, subject to the total weight constraint $\sum_{i=1}^n x_i \cdot w_i \leq W$.

Design an optimal polynomial time greedy algorithm for the fractional knapsack problem and argue about its correctness.

Solution

This is Dantzig's greedy algorithm for solving the fractional knapsack problem. The algorithm works as follows:

- First, sort the items in terms on non-increasing ratio v_i/w_i (this is sometimes called the "bang-per-buck").
- Start putting items in the knapsack in that order, until you encounter an item that can not fit in the knapsack anymore.
- Put as large a fraction of that item in the knapsack, until you reach the total weight constraint W.

We claim that this algorithm solves the fractional knapsack problem optimally. Suppose without loss of generality that the items are sorted in terms of non-increasing v_i/w_i and that no two items have the same such ratio (therefore the order is actually decreasing). Let $o_1, o_2, \ldots, o_n$ be the fractions of items that are put in the knapsack in the optimal solution in that order, and let $g_1, \ldots, g_n$ be the fractions of items selected by the greedy algorithm (in both cases, some fractions might be 0). By definition of the optimal solution, we have that $\sum_{i=1}^n o_i \cdot v_i \geq \sum_{i=1}^n g_i \cdot v_i$.

If $o_i = g_i$ for every index, then the two solutions are the same and we are done. Let j be the first index for which the two solutions differ. By definition of the greedy algorithm, it must hold that $g_j > o_j$, as item j has the largest ratio v_j/w_j over all remaining items (not already in the knapsack), and the greedy algorithm selects as large a fraction of it as possible. By the definition of the optimal, there must exist another index $\ell > j$ such that $o_\ell > g_\ell$. Now, construct a new solution $o' = \{o'_1, o'_2, \ldots, o'_n\}$ such that

- $o'_k = o_k$ for all $k \neq j, \ell$,
- $o'_j = o_j + \varepsilon$,
- $o'_\ell = o_\ell - \varepsilon \cdot \frac{w_j}{w_\ell}$.

Note that $\sum_{i=1}^n o_i \cdot w_i = \sum_{i=1}^n o'_i \cdot w_i$ and therefore o'_i is a feasible solution. Furthermore, we have that

$$\sum_{i=1}^n o'_i \cdot v_i = \sum_{i=1}^n o_i \cdot v_i + \varepsilon v_j + \varepsilon \cdot \frac{w_j}{w_\ell} \cdot v_\ell \geq \sum_{i=1}^n o_i \cdot v_i,$$

where the last inequality holds since $v_j/w_j > v_\ell/w_\ell$. This means that o is a feasible solution with largest total value than o, contradicting the optimality of o.

Problem 2

Solved Exercise 3 from Kleinberg and Tardos - Algorithm Design, Chapter 4, page 187.

Suppose you are given a connected graph G, wieh edge costs that you may assume are all distinct. G has n vertices and m edges. A particular edge e of G is specified. Give an algorithm with running time $O(m+n)$ to decide whether e is contained in a minimum spanning tree of G.

Solution

See Kleinberg and Tardos - Algorithms Design, Chapter 4, page 187.

Problem 3

A contiguous subsequence of length k a sequence S is a subsequence which consists of k consecutive elements of S. For instance, if S is $1, 2, 3, -11, 10, 6, -10, 11, -5$, then $3, -11, 10$ is a contiguous subsequence of S of length 3. Give an algorithm based on dynamic programming that, given a sequence S of n numbers as input, runs in linear time and outputs the contiguous subsequence of S of maximum sum. Assume that a subsequence of length 0 has sum 0. For the example above, the answer of the algorithm would be $10, 6. -10, 11$ with a sum of 17.

Solution

Let $a_1 a_2 \ldots a_n$ be the sequence S. We will use dynamic programming to design an algorithm that solves the contiguous subsequence problem. Let $M(j)$ be the optimal solution (the length of the subsequence of maximum sum) *ending at position* j. By definition, we have that $M(0) = 0$. We have the following relation:

$$M[j+1] = \max\{M[j] + a_{j+1}, 0\},$$

with $M[1] = \max\{a_1, 0\}$. To find the contiguous subsequence S^* of maximum sum, we operate as follows. First, we find the element $i*$ for which $M[i^*]$ is maximised. This can be done in polynomial time, by computing the partial sums and storing them in an array (similarly to the approach in the weighted interval scheduling problem). S^* will end at i^*. The beginning of S^* will be the largest $j \leq i^*$ for which $M[j-1] = 0$, as extending the subsequence to start before j will only decrease the sum. If there is no such j, then S^* starts at the beginning of S.