Solutions to assignment 12

Exercise 1
Formulate the maximum multicommodity flow problem as an LP: Let $G = (V, E)$ be a directed graph with edge capacities $u : E \to \mathbb{R}^+$, a set of commodities $K = \{1, 2, \ldots, k\}$, where commodity $i \in K$ is given by a source vertex $s_i \in V$ and a sink vertex $t_i \in V$, a “multicommodity flow” $f$ consists of $s_i - t_i$ flows $f_i$ for $i \in K$. The flow $f$ satisfies the capacity constraints “$f \leq u$”, if $f(e) = \sum_{i=1}^{k} f_i(e) \leq u(e)$, for every $e \in A$. The value of a multicommodity flow $f$ is the $k$–tuple $(|f_1|, |f_2|, \ldots, |f_k|)$; its total value is $|f| = \sum_{i=1}^{k} |f_i|$.

If in addition, a set of demands $(d_1, \ldots, d_k) \in \mathbb{R}^k_+$ is given for each commodity $i \in K$, a multicommodity flow $f$ satisfies the demands if $|f_i| = d_i$ for all $i \in K$.

The maximum multicommodity flow problem asks for a multicommodity flow $f$ maximizing $|f|$.

Solution
We denote by $x_i(e)$ the amount of flow of commodity $i$ going through the edge $e$. Then the maximum multicommodity flow problem can be expressed as an LP problem as follows:

$$\max \sum_{i=1}^{k} \left( \sum_{e \in \delta^+(s_i)} x_i(e) - \sum_{e \in \delta^-(s_i)} x_i(e) \right)$$

subject to:

- capacity constraints: $\sum_{i=1}^{k} x_i(e) \leq u(e)$ for all $e \in A$
- flow conservation constraints:
  $$\sum_{e \in \delta^+(v)} x_i(e) - \sum_{e \in \delta^-(v)} x_i(e) = 0 \quad \text{for all } i \in K \text{ and for all } v \in V \setminus \{s_i, t_i\}$$
  $$x_i(e) \geq 0 \quad \text{for all } i \in K, e \in A$$

In this case the number of variables is: $k|A| = km$
number of constraints: $m + k(n - 2) + km$.

An alternative LP formulation can be obtained by using ”path” variables. For $i \in K$ let $P_i$ denote the set of all $s_i - t_i$-paths and let $P := \bigcup_{i=1}^{k} P_i$. Let $x_p$ be the amount of flow on the path $p \in P$

$$\max \sum_{p \in P} x_p$$

subject to:

- $\sum_{p \in P, e \in p} x_p \leq u(e)$ for all $e \in A$
- $x_p \geq 0$ for all $p \in P$
This LP formulation has an exponential number of variables.

Exercise 2
Linear time fractional knapsacks.

Explain how to solve the fractional knapsack problem (the linear relaxation of the knapsack problem, i.e. items can be selected fractionally) in linear expected time. Hint: use a similar idea as in the well known quicksort-like median selection algorithm.

Solution
Let \( n \) be the number of items and \( p_i, w_i \) be the profit and the weight of item \( i \) respectively. It is known, that one can obtain an optimal solution for the fractional knapsack problem, by first taking the ratio profit per weight \( p_i / w_i \), for every \( 1 \leq i \leq n \), then sort these ratios in decreasing order and finally add items in the knapsack greedily starting with the one that has the largest ratio. Add the items one after the other in their entirety, as long as there is enough space left in the knapsack. When there is not enough space left to fit one more item, we take only the fraction of this item that fits in and we terminate.

This procedure requires time \( O(n \log n) \) because of the sorting step. In this exercise we show how to avoid the sorting step by using the idea of the quicksort-like median selection algorithm so that we achieve linear expected time.

Let \( R = \{ \frac{p_1}{w_1}, \ldots, \frac{p_n}{w_n} \} \) be the profit/weight ratios. Consider the following procedure:

**Input:** A set \( R \) of \( n \) ratios, knapsack capacity \( W \).

**Output:** A set of items fitting in the knapsack maximizing the total profit.

1. Choose an element \( r \) uniformly at random from \( R \)
2. Determine:
   \[
   \begin{align*}
   & R_1 = \{p_i / w_i \mid p_i / w_i > r, \text{ for } 1 \leq i \leq n\}, \quad W_1 = \sum_{i \in R_1} w_i \\
   & R_2 = \{p_i / w_i \mid p_i / w_i = r, \text{ for } 1 \leq i \leq n\}, \quad W_2 = \sum_{i \in R_2} w_i \\
   & R_3 = \{p_i / w_i \mid p_i / w_i < r, \text{ for } 1 \leq i \leq n\}, \quad W_3 = \sum_{i \in R_3} w_i 
   \end{align*}
   \]
3. if \( W_1 > W \)
   then recurse on \( R_1 \) and return the computed solution.
   else
   while (there is space in knapsack and \( R_2 \) not empty)
     add items from \( R_2 \)
     if (knapsack gets full)
       return the items in \( R_1 \) and the items just added from \( R_2 \).
     else
       reduce knapsack capacity by \( W_1 + W_2 \),
       recurse on \( R_3 \) and return the items in \( R_1 \cup R_2 \)
       and the items returned from the recursive call.

*The analysis is still to be done.*

Exercise 3
Formulate the following set covering problem as an ILP: Given a set \( M = \{1, \ldots, m\} \), \( n \) subsets \( M_i \subseteq M \) for \( 1 \leq i \leq n \) and a cost \( c_i \) for set \( M_i \). Assume \( \bigcup_{i=1}^{n} M_i = \{1, \ldots, m\} \). Select \( F \subseteq \{1, \ldots, n\} \) such that \( \bigcup_{i \in F} M_i = \{1, \ldots, m\} \) and \( \sum_{i \in F} c_i \) is minimized.
Solution
We construct a $m \times n$-matrix $E$ with
\[
E_{i,j} = \begin{cases} 
1, & \text{if } i \in M_j; \\
0, & \text{else}.
\end{cases}
\]

Then the problem can be given as
\[
\min (c_1, c_2, \ldots, c_n) \times x
\]
\[
x \in \{0, 1\}^n
\]
\[
Ex \geq \begin{pmatrix} 
1 \\
1 \\
\vdots \\
1 
\end{pmatrix}
\]

The vector $x$ has a 1 at position $i$, if $M_i$ is to be selected – and a 0 else. The product of the cost vector $(c_1, c_2, \ldots, c_n)$ and $x$ mirrors the total cost of the selected sets. The product $Ex$ yields a vector, which contains for each integer $i = 1, \ldots, m$ the number of those selected sets, which contain $i$. The $\geq$ is to be understood as a line-by-line comparison: Each element should be contained in at least one selected set.

Exercise 4
Making Change.

Suppose you have to program a vending machine that should give exact change using a minimum number of coins.

1. Develop an optimal greedy algorithm that works in the Euro zone with coins worth 1, 2, 5, 10, 20, 50, 100, and 200 cents and in the dollar zone with coins worth 1, 5, 10, 25, 50, and 100 cents.

2. Show that this algorithm would not be optimal if there were a 4 cent coin.

3. Develop a dynamic programming algorithm that gives optimal change for any currency system.

Solution

1. Generally, we have $n$ coins with values $v_1, v_2, \ldots, v_n$. For this subexercise, we assume w.l.o.g. that $1 = v_1 < v_2 < \ldots < v_n$.

   Function $\text{return greedyChange}(v_1, v_2, \ldots, v_n : \text{coin values}; c : \text{change}) : \text{multiset of used coins}$

   \[
   R = \emptyset
   \]
   \[
   \text{for } i := n \text{ downto 1 do}
   \]
   \[
   \text{while } v_i \leq c \text{ do}
   \]
   \[
   c := c - v_i
   \]
   \[
   R := R \cup \{i\}
   \]
   \[
   \text{return } R
   \]

2. Counter-example: change $c = 8$. The above greedy algorithm returns three resp. four coins with the values 5, 2 and 1 (Euro zone) resp. 5, 1, 1 and 1 (dollar zone). However, the optimal solution consists of two coins 4 and 4.
3. A recurrence that can be used to solve this problem by Dynamic Programming is

\[
m(i, w) = \begin{cases} 
\infty, & \text{if } i < 1 \lor w < 0; \\
0, & \text{if } i \geq 1 \land w = 0; \\
\min\{m(i - 1, w), m(i, w - v_i) + 1\}, & \text{otherwise.}
\end{cases}
\]

where the 2-dimensional table \(m\) contains at position \((i, w)\) the minimum number of coins with indices \(\leq i\) needed to represent the value \(w\). This table can be filled in an appropriate order and, finally, the entry \((n, c)\) contains the optimal number of coins. In order to be able to determine which coins have to be used to get the optimal number, we store for each table entry our decision.

It is possible to modify the recurrence so that we have only an 1-dimensional table:

\[
m(w) = \begin{cases} 
\infty, & \text{if } w < 0; \\
0, & \text{if } w = 0; \\
\min_{1 \leq i \leq n} m(w - v_i) + 1, & \text{otherwise.}
\end{cases}
\]

This reduces the memory usage.