ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institute for Operations Research
ETH Zurich HG G21-22
Prof. Dr. Rico Zenklusen and Assistants
Contact: math.opt@ifor.math.ethz.ch

IFOR
Institute for Operations Research
Department of Mathematics

Fall 2019

Mathematical Optimization – Solutions to problem set 6

https://moodle-app2.let.ethz.ch/course/view.php?id=4844

Problem 1: Complementary slackness

(a) The dual of the given linear program is the following.

$$
\begin{array}{rrrrrrrrrrrrl}
\min & - & y_1 & + & 5y_2 & + & y_3 & + & 5y_4 & & & & \\
 & & y_1 & + & 7y_2 & & & & & + & y_5 & \geq & 0 \\
 & & & & y_2 & + & y_3 & + & y_4 & & & \geq & 4 \\
 & & & & y_2 & - & y_3 & - & y_4 & + & y_5 & \geq & 2 \\
 & - & y_1 & & & & & + & y_4 & - & 4y_5 & \geq & -1 \\
 & & & & y_2 & & & + & y_4 & + & 5y_5 & \geq & 3 \\
 & & 3y_1 & + & y_2 & - & y_3 & + & y_4 & & & \geq & -7 \\
 & & & & & & & & & & y_i & \geq & 0 \quad \forall i \in \{1, \ldots, 5\}
\end{array}
$$

(b) We first focus on the first point, i.e.,

$$x^* = \begin{pmatrix} 0 & 2 & 2 & 4 & 0 & 1 \end{pmatrix}^\top .$$

Note that it is feasible for the primal linear program, and the first, second, and fourth constraint are satisfied with equality, while the third and fifth constraint are not. Thus, a dual solution $y^* = \begin{pmatrix} y_1^* & y_2^* & y_3^* & y_4^* & y_5^* \end{pmatrix}^\top$ that satisfies the complementary slackness conditions must have $y_3^* = y_5^* = 0$. Moreover, it must satisfy the second, third, fourth, and sixth dual constraint with equality because x_2^*, x_3^*, x_4^*, and x_6^* are strictly positive, i.e.,

$$
\begin{array}{rrrrrrrrrrr}
 & & y_2^* & + & y_3^* & + & y_4^* & & & = & 4 \\
 & & y_2^* & - & y_3^* & - & y_4^* & + & y_5^* & = & 2 \\
- & y_1^* & & & & + & y_4^* & - & 4y_5^* & = & -1 \\
 & 3y_1^* & + & y_2^* & - & y_3^* & + & y_4^* & & = & -7
\end{array} .
$$

Plugging in $y_3^* = y_5^* = 0$, the above system reduces to

$$
\begin{array}{rrrrrrrr}
 & & & y_2^* & + & y_4^* & = & 4 \\
 & & & y_2^* & - & y_4^* & = & 2 \\
- & y_1^* & & & + & y_4^* & = & -1 \\
 & 3y_1^* & + & y_2^* & + & y_4^* & = & -7
\end{array} .
$$

Note that the first three equations have the unique solution $y^* = \begin{pmatrix} 2 & 3 & 0 & 1 & 0 \end{pmatrix}^\top$, but this solution does not satisfy the fourth equation. We conclude that there is no dual feasible solution y^* that satisfies complementary slackness conditions with x^*. Consequently, x^* cannot be a primal optimal solution.

Let us now focus on the second given point, namely

$$x^* = \begin{pmatrix} 0 & 3 & 2 & 1 & 0 & 0 \end{pmatrix}^\top .$$

This point x^* is primal feasible, and it satisfies the first, second, and third constraint in the primal linear program with equality, while the fourth and fifth constraint both have slack. Thus, a dual solution $y^* = \begin{pmatrix} y_1^* & y_2^* & y_3^* & y_4^* & y_5^* \end{pmatrix}^\top$ that satisfies the complementary slackness conditions must have $y_4^* = y_5^* = 0$. Moreover, it must satisfy the second, third, and fourth dual constraint with equality because x_2^*, x_3^*, and x_4^* are strictly positive, i.e.,

$$
\begin{array}{rrrrrrrrrrr}
 & & y_2 & + & y_3 & + & y_4 & & & = & 4 \\
 & & y_2 & - & y_3 & - & y_4 & + & y_5 & = & 2 \\
- & y_1 & & & & + & y_4 & - & 4y_5 & = & -1
\end{array} .
$$

Plugging in $y_4^* = y_5^* = 0$, we see that this system has the unique solution $y^* = \begin{pmatrix} 1 & 3 & 1 & 0 & 0 \end{pmatrix}^\top$. Moreover, we can check that y^* is a dual feasible solution. The complementary slackness theorem thus implies that x^* is optimal for the given primal linear program, and y^* is optimal for the corresponding dual linear program.

Problem 2: Asymptotic growth and Landau notation

(a) Let $f(n) = \log(n^2)$ and $g(n) = \log(n)$. Then $f = \Theta(g)$.

True ☑ False ☐

Short justification: We have $f(n) = \log(n^2) = 2 \cdot \log(n) = 2 \cdot g(n)$ for all $n \geq 1$, hence $f = O(g)$ and $g = O(f)$, so $f = \Theta(g)$.

(b) Let $f(n) = n + \sqrt{n}$ and $g(n) = n \cdot \log(n)$. Then $f = O(g)$.

True ☑ False ☐

Short justification: We have $f(n) = n + \sqrt{n} \leq 2n \leq 2n \log n = 2 \cdot g(n)$ for all $n > e$ (assuming that log is the natural logarithm).

(c) Let $f(n) = 2^n$ and $g(n) = n!$. Then $f = \Theta(g)$.

True ☐ False ☑

Short justification: We have $g(n) = n! = 1 \cdot 2 \cdot \ldots \cdot n \geq 2^{n-2} \cdot n = \frac{n}{4} \cdot 2^n = \frac{n}{4} f(n)$. Thus, g cannot be bounded from above by a constant multiple of f, and hence $g = O(f)$ does not hold, implying that $f = \Theta(g)$ does not hold, either.

(d) Let $f(n) = \sqrt{n}^n$ and $g(n) = n^{\sqrt{n}}$. Then $f = \Theta(g)$.

True ☐ False ☑

Short justification: If $f = O(g)$, there exist $c > 0$ and M with $\sqrt{n}^n \leq c \cdot n^{\sqrt{n}}$ for all $n \geq M$. Equivalently (taking logs and dividing by $\sqrt{n} \log n$), $\frac{1}{2}\sqrt{n} \leq \frac{\log c}{\sqrt{n} \log n} + 1$. The latter is wrong for large n, as $\frac{1}{2}\sqrt{n} \to \infty$ while $\frac{\log c}{\sqrt{n} \log n} + 1 \to 1$ for $n \to \infty$. Thus $f \neq O(g)$, hence $f \neq \Theta(g)$.

(e) Let $f(n) = 3^n$ and $g(n) = 2^{n + \log n}$. Then $f = \Omega(g)$.

True ☑ False ☐

Short justification: Note that $\frac{n}{2} > \log n$ (assuming that log is the natural logarithm). Using this, we get $f(n) = 3^n = 9^{\frac{n}{2}} > 8^{\frac{n}{2}} = 2^{\frac{3}{2}n} > 2^{n + \log n} = g(n)$, thus $f = \Omega(g)$.

(f) Let $f, g \colon \mathbb{Z}_{\geq 1} \to \mathbb{Z}_{\geq 1}$. Then, at least one of the relations $f = O(g)$, $g = O(f)$, or $f = \Theta(g)$ holds.

True ☐ False ☑

Short justification: Let $f(n) = \begin{cases} 1 & \text{if } n \text{ is even} \\ n & \text{if } n \text{ is odd} \end{cases}$, and $g(n) = \begin{cases} n & \text{if } n \text{ is even} \\ 1 & \text{if } n \text{ is odd} \end{cases}$. There are no $c > 0$ and M with $f(n) \leq c \cdot g(n)$ for all odd $n \geq M$, hence $f \neq O(g)$, and thus also $f \neq \Theta(g)$. Moreover, there are no $c > 0$ and M with $g(n) \leq c \cdot f(n)$ for all even $n \geq M$, hence $g \neq O(f)$.

Problem 3: Running time of a sorting algorithm

(a) The array A changes as follows:

$$\begin{aligned} [4,3,1,2] &\xrightarrow{i=1,\,j=1} [3,4,1,2] \xrightarrow{i=2,\,j=2} [3,1,4,2] \\ &\xrightarrow{i=2,\,j=1} [1,3,4,2] \xrightarrow{i=3,\,j=3} [1,3,2,4] \xrightarrow{i=3,\,j=2} [1,2,3,4] \ . \end{aligned}$$

(b) The `for` loop is entered precisely $(n-1)$ times, with the `while` loop being repeated at most i times. All operations inside the loops take constant time every at every execution of the loops, so there exists a constant $c > 0$ such that the number of elementary operations that the algorithm does is bounded by

$$c \cdot \sum_{i=1}^{n-1} i = c \cdot \frac{n(n-1)}{2} = O(n^2) \ .$$

Thus, the worst-case running time of the algorithm is in $O(n^2)$.

(c) Consider the input $A = [n, n-1, \ldots, 2, 1]$ consisting of the numbers from 1 to n sorted in decreasing order. For this input, the `while` loop is run precisely i times, with more than one elementary operation inside the loop at each time. Thus, the total number of elementary operations is more than

$$\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} = \Omega(n^2) \ ,$$

proving that the running time on the given input is in $\Omega(n^2)$.

(d) If the input array is $A = [1, 2, \ldots, n-1, n]$, i.e., if the array is sorted already, then the while loop is never actually entered, because the condition $A[j-1] > A[j]$ will never be satisfied. Thus, in every one of the $(n-1)$ iterations of the for loop, there is only a constant number of elementary operations, so there exists $c > 0$ such that the total number of elementary operations is bounded by $c \cdot (n-1)$. Thus, the running time on a sorted instance is $O(n)$.

Problem 4: Quering edge existence in incidence lists

Denote by L_x the list of edges that are incident to vertex x. For working with incidence list here, we assume access to the following functions:

- `is_empty`(L): returns `true` if L is an empty list, `false` if not.
- `start`(L): Returns the first element of a non-empty list L, i.e., an edge.
- `not_last`(e, L): Returns `true` if the edge e is not the last element of L, `false` if it is.
- `next`(e, L): Returns the edge succeeding e in the list L, given that e is not last in the list.

Using these functions, we can write the algorithm as follows.

Algorithm 1

Input: L_u and L_v for $u, v \in V$.
Output: `true` if $\{u, v\} \in E$, `false` else.

1 **if** (`is_empty`(L_u) **or** `is_empty`(L_v)) **then**
2   **return** `false`
3 $a =$ `start`(L_u), $b =$ `start`(L_v)
4 **if** ($v \in a$ **or** $u \in b$) **then**
5   **return** `true`
6 **while** (`not_last`(a, L_u) **and** `not_last`(b, L_v)) **do**
7     $a =$ `next`(a, L_u), $b =$ `next`(b, L_v)
8   **if** ($v \in a$ **or** $u \in b$) **then**
9   **return** `true`
10 **return** `false`

Let us first show that the algorithm terminates in time $O(\min\{\deg(u), \deg(v)\})$. To this end, let $w \in \{u, v\}$ be such that $\deg(w) = \min\{\deg(u), \deg(v)\}$. We show that the algorithm's running time is bounded by $c \cdot \deg(w)$ for some constant c. If $\deg(w) = 0$, the algorithm terminates in line 2, and thus after constantly many elementary operations. If $\deg(w) > 0$, then the first edge in L_w is assigned to one of a or b. If the `while` loop is entered, it is checked whether a or b is the last element in L_u or L_v; if not, a and b are updated to represent their successors. Thus, the `while` loop terminates at the latest once a or b are the last element of L_w, i.e., after at most $\deg(w)$ many iterations of the `while` loop. The number of elementary operations in each loop is constant, hence the total number of operations is in $O(\deg(w))$.

To prove correctness, let $e = \{u, v\}$ be the edge that is queried. Let us first consider the case that $e \notin E$. If $\deg(u) = 0$ or $\deg(v) = 0$, then L_u or L_v is empty, and the algorithm correctly returns `false` in line 2. If $\deg(u), \deg(v) > 0$, then the conditions in line 4 and in line 8 will never be satisfied (as $e \notin E$). As the algorithm terminates after finitely many steps (we saw this above), the `while` loop is left after finitely many steps, thus reaching line 10 and returning `false`, as desired.

If $e \in E$, we know that $e \in L_u$ and $e \in L_v$. If e is the first edge in L_u or L_v, the algorithm returns `true` in line 5. Else, neither a nor b defined in line 3 can be the last edge in L_u and L_v, respectively, hence the `while` loop is started, and it is only left once e is found, or once all edges in one of the lists L_u or L_v were checked. Thus, there is a point where in line 8, $a = e$ or $b = e$ (as $e \in L_u \cap L_v$), hence `true` is returned, as desired.

Problem 5: Meeting at a central point

For $i \in \{1, 2, 3\}$ and every vertex $v \in V$, let $d_i(v)$ denote the minimum time it takes the caterpillar starting at v_i to get to v. To calculate all values $d_i(v)$, we can run BFS with starting vertex v_i, for each $i \in \{1, 2, 3\}$.

For every vertex $v \in V$, the earliest time for a meeting of the three caterpillars at v is $h(v) := \max\{d_1(v), d_2(v), d_3(v)\}$. Thus, a minimizer of $\min_{v \in V} h(v)$ is what we are looking for.

Note that the three calls to BFS take time $O(|V| + |E|)$, determining $h\colon V \to \mathbb{Z}$ the results of the BFS calls takes time $O(|V|)$, and finding $v \in V$ minimizing $h(v)$ takes time $O(|V|)$. Hence the algorithm described above has running time $O(|V| + |E|)$.

Remark: We could also run BFS starting from every node $v \in V$ to determine $h(v)$—this would give a running time of $\Omega(|V|^2 + |V| \cdot |E|)$.