Problem Set 1 – Combinatorial Optimization 2012 – EPFL
Solutions

Note that these are fairly brief solutions/explanations; in your handed-in problems you should be a bit more careful and precise.

Review of Linear Programming

1. *For the primal and dual linear program*

$$\max\{c^T x : Ax \leq b,\ x \geq 0\}, \quad \min\{b^T y : A^T y \geq c,\ y \geq 0\},$$

complete the following table by writing possible or impossible, with justification.

↓ primal dual →	finite optimum	unbounded	infeasible
finite optimum	*possible*	*impossible*	*impossible*
unbounded	*impossible*	*impossible*	*possible*
infeasible	*impossible*	*possible*	*possible*

2. *Given an "oracle" algorithm that can tell you if any system of linear inequalities $Mx \leq m$ has a feasible solution $x \in \mathbb{R}^n$, give an algorithm that solves*

$$\max\{c^T x : Ax \leq b,\ x \geq 0\}.$$

There was a mistake in the question. What was meant was that the oracle *gives* a feasible solution, rather than just saying whether one exists. In that case one could have asked for a feasible solution to the system of inequalities

$$\{Ax \leq b,\ -A^T y \geq -c,\ -x \leq 0,\ -y \leq 0,\ b^T y \leq c^T x\}.$$

By the duality theorem, a feasible solution (x, y) to these inequalities would give a maximum solution x for the linear program.
But for the stated question, the best one seems to be able to do is to approximate an optimum, as follows.
Let $z \in \mathbb{R}_+$ denote a number which is equal to the maximum absolute value of a subdeterminant of the matrix $(\mathbf{A}\ \mathbf{b})$. First, we can easily test if the linear program is bounded by asking the oracle if the following systems $Ax \leq b,\ -c^T x \leq -z - 1,\ -x \leq 0$ has a solution.
In case the linear program is bounded the algorithm performs a binary search such that in each step we ask the oracle whether the following system is feasible. $Ax \leq b,\ -c^T x \leq \alpha,\ -x \leq 0$, where α is defined as follows. We associate with every step an interval containing the solution of our linear program, which is initially $(-z, z)$. Then α is the midpoint of this interval. Thus, initially $\alpha = (-z + z)/2 = 0$. At each step, if the oracle says "yes", we associate the upper half of the current interval with the next step. Otherwise, we associate the lower half of the current interval with the next step. We run the binary search until the interval associated with the current step is sufficiently small.

3. *Give the dual of the following linear program:*

$$\begin{array}{lll} \text{maximize} & x_1 + x_2 + x_3 & \\ \text{subject to} & x_1 + 2x_2 - 3x_3 & \leq 1, \\ & x_1 - x_2 + 2x_3 & \geq 2, \\ & -x_2 + x_3 & = 1, \\ & x_1 \geq 0,\ x_2 \leq 0. & \end{array}$$

Dual:

$$\begin{array}{rll} \text{minimize} & y_1 + 2y_2 + y_3 & \\ \text{subject to} & y_1 + y_2 & \geq 1, \\ & 2y_1 - y_2 - y_3 & \leq 1, \\ & -3y_1 + 2y_2 + y_3 & = 1, \\ & y_1 \geq 0, y_2 \leq 0. & \end{array}$$

4. *Given a set of blue points and red points in $\mathbb{R}^2$, write down a linear program for finding a line that separates the blue points from the red points, if possible.*

   Let $R = \{\mathbf{r_1}, \ldots, \mathbf{r_k}\}$ denote the set of red points. Let $B = \{\mathbf{b_1}, \ldots, \mathbf{b_l}\}$ denote the set of red points.

   A linear program for finding a required line $x_1x + x_2y = z$, such that $\mathbf{x} = (x_1, x_2)$, can look as follows. By $< . >$ we denote the scalar product.

$$\begin{array}{rll} \text{maximize} & d & \\ \text{subject to} & < \mathbf{r_i}, \mathbf{x} > -z & \geq d, \ i = 1 \ldots k, \\ & < \mathbf{b_i}, \mathbf{x} > -z & \leq -d, \ i = 1 \ldots l. \end{array}$$

5. *Write down an integral program for the Independent Set Problem: Given an undirected graph, find a maximum subset of vertices that have no edges between them. Give an example of a graph for which the relaxation has a different optimum value from the integer program.*
   Give the dual of the relaxation of the integer program, and an example for which this dual has a different optimum value from its integer version.

   Let $G = (V, E)$ denote a graph.
   Integral program for IS:

$$\begin{array}{rll} \text{maximize} & \sum_{v \in V} x_v & \\ \text{subject to} & x_v + x_u & \leq 1, \ \forall uv \in E \\ & x_v & \geq 0, \ x_v \in \mathbb{Z}, \ \forall v \in V. \end{array}$$

   The optimal value of the relaxation for any cycle of length $2k + 1$ is $k + 1/2$ while the largest independent set is of size k.

   Dual of its relaxation:

$$\begin{array}{rll} \text{minimize} & \sum_{e \in E} x_e & \\ \text{subject to} & \sum_{e \ni v} x_e & \geq 1, \ \forall v \in V \\ & x_e & \geq 0, \ \forall e \in E. \end{array}$$

   The optimal value of the dual of the relaxation for any cycle of length $2k + 1$ is $k + 1/2$ while the smallest edge cover is of size $k + 1$.

6. *Find an algorithm that, given a directed graph and two disjoint vertex sets S, T, returns a dipath between a vertex of S and a vertex of T, if one exists.*

   For each $v \notin S$ such that there is an edge uv with $u \in S$, add one such edge. Keep repeating this until you reach a vertex t of T (if you don't, there is no dipath). Now trace a path back from t to a vertex s, which is possible since every vertex you've added has only one entering edge.

7. *Find an algorithm that, given an undirected graph, determines if the graph is bipartite. If it is, it should give a bipartition, and if it isn't, it should give proof of that.*

   Since we can run the algorithm on each connected component separately we suppose that G is connected. Let $v \in V$ denote an arbitrary vertex. Color v with *red*. Let *blue* be the color which is opposite to *red* and vice-versa.

   Perform a breadth-first search from v such that each visited vertex receives a color which is opposite to the color of its immediate predecessor in the search. Our algorithm traverses all the edges so one vertex can have more than one immediate predecessor. If we visit for the first time during our search a vertex u that was already colored by the same color as its current predecessor z, we know that there exists a walk W in our graph starting at v passing (in this order) through the vertex z, the edge zu and finally returning to v. Colors of the vertices on the initial part of W between v and z alternate between *red* and *blue*, u has the same color as z and colors of the vertices on the remaining part of v alternate between *red* and *blue*. Thus, W is a closed walk of an odd length. It follows that there exists a cycle $C \subseteq W$ of an odd length in G which means that G is not bipartite.

   If we do not visit during our search a vertex u that was already colored by the same color as its actual predecessor z, in the end we obtain a proper two coloring of G, which shows that G is bipartite.

8. *Find an algorithm that, given an undirected connected graph, returns an Euler circuit or proof that none exists. An Euler circuit is a circuit that uses each edge of the graph exactly once.*

   It is well-known that an Euler circuit in our graph exists iff all the vertices have an even degree and our graph is connected. Thus, we first check the parity of degrees of the vertices in G. If all the vertices have an even degree, similarly as in the previous example, we perform a graph search algorithm starting from an arbitrary vertex $v \in V$. However, this time we do a depth-first search.

   Since the degrees of all vertices in V are even whenever we visit a vertex $u \in V$, $u \neq v$, there exists at least one untraversed edge incident to u. Thus, we can prolong our Euler tour by one edge or close our tour T by visiting v. If we visited v and all the edges incident to v were already traversed, we start traversing T, until we visit a vertex u that is incident to an untraversed edge. If no such vertex u is found T is Euler circuit. Otherwise, we start a new depth-first search from u and in the end we obtain a new tour $T' = uT'u$. We concatenate T and T' thereby obtaining a larger closed tour $T := vTuT'uTv$.

   Then we start traversing T from u until we encounter a vertex with an adjacent untraversed edge and repeat the argument from the previous paragraph with u playing the role of v.

9. (Harder) *Give a 2-approximation algorithm that finds a maximum acyclic subgraph in a digraph. A digraph is acyclic if it has no directed cycle (a connected subgraph with the in-degree and out-degree of every vertex equal to 1). Note that this is not the same as the underlying graph being acyclic.*

   Solution 1: Divide the set of nodes into two nonempty sets, A and B. Consider the set of edges from A to B and the set of edges from B to A. Throw out the edges from the smaller set and keep the edges from the larger set (break ties arbitrarily). Recurse on A and B individually.

   The resulting graph is acyclic because every cycle will be broken the first time the cycle's nodes are split among A and B. The total set of edges we throw out is no bigger than the total set of edges we keep, so we've thrown out at most half of the edges.

   Solution 2: Start with $S = V(G)$. Repeat this: Pick a $v \in S$ and remove it from S . Look at the set of edges from v to S, and the set of edges from S to v; throw away the smaller of these sets. The resulting graph has at least half the number of edges, and cannot have a cycle: inside any subgraph, the first-picked of its vertices cannot have both an incoming and an outgoing edge (from/to later-picked vertices), unlike any vertex in a cycle.