cs473: Algorithms Assigned: *Wed., Nov. 18, 2020*

Problem Set #9

Forbes/Chekuri Due: *Thu., Dec. 3, 2020 (5:00pm)*

For problems that ask for a linear-programming formulation of some problem, a full-credit solution requires the following components:

- A list of variables, along with a brief in-words description of what each variable represents.
- A linear objective function (expressed either as minimization or maximization, whichever is more appropriate), along with a brief in-words description of its meaning.
- A sequence of linear inequalities (expressed using $\leq$, $=$, or $\geq$, whichever is more appropriate), along with a brief in-words description of what each constraint represents.
- A proof that your linear programming formulation is correct, meaning that the optimal solution to the original problem can always be obtained from the optimal solution to the linear program. This may be very short.

It is ***not*** necessary to express the linear program in canonical form, or even in matrix form. Clarity is much more important than formality.

For problems that ask to prove that a given problem X is NP-hard, a full-credit solution requires the following components:

- Specify a known NP-hard problem Y, taken from the problems listed in Erickson's notes.
- Describe a polynomial-time algorithm for Y, using a black-box polynomial-time algorithm for X as a subroutine. Most NP-hardness reductions have the following form: given an arbitrary instance of Y, describe how to transform it into an instance of X, pass this instance to a black-box algorithm for X, and finally, describe how to transform the output of the black-box subroutine to the final output solving the original instance of Y. A diagram can be helpful.
- Prove that your reduction is correct. As usual, correctness proofs for NP-hardness reductions usually have two components, representing that the answer is true/false, or representing that the answer is too-large/too-small.

All problems are of equal value.

1. There are number of clustering problems that are well-studied in practice. Consider a collection of n points P and a function $d : P \times P \to \mathbb{R}_+$ that defines a distance $d(p, q)$ between every pair of points; often this distance function is assumed to be metric. Given an integer k, center-based clustering consists of finding a subset $S \subseteq P$ of k points[1] to minimize some objective. Once S is chosen (called centers) then this defines a clustering of P into k-clusters where each point p is assigned to its nearest center according to d; that is each point p is assigned to $\text{argmin}_{v \in S} d(p, v)$.

   - The k-median problem is to find k centers S to minimize $\sum_{p \in P} d(p, S)$ where $d(p, S) = \min_{v \in S} d(p, v)$. Write an integer linear programming formulation for k-median by using two types of variables. One set of variables $y_p, p \in V$, is to indicate whether p is chosen as a center. The other set of variables $x_{p,q}$ is to indicate whether p is assigned to q. Argue that even if only the y variables are constrained to be integers, the formulation yields an optimum solution.
   - Write the dual of the LP relaxation.
   - **Optional**. The k-means problem is the same as k-median except that the objective is minimize $\sum_{p \in P} d(p, S)^2$. How would your formulation change?
   - **Optional**. k-median and k-means are special cases of minimizing ℓ_p norms of distances to the chosen centers. The k-center problem is to minimize $\max_p d(p, S)$ which is the ℓ_∞ version. The k-center objective cannot be directly written in the objective. Instead, given R, write a formulation to check if there is a feasible set of k centers such that the maximum distance of any point to the centers is at most R.

2. Flows are typically defined as a function on the edges, but the path-based definition is useful and necessary in various applications. Let $G = (V, E)$ be a directed graph with non-negative capacities $c : E \to \mathbb{R}_{\geq 0}$. Given distinct nodes $s, t \in V$ let $P_{s,t}$ denote the set of all simple paths between s and t. For an integer $k \geq 1$ let $P^{(k)}_{s,t}$ be the set of all simple paths from s to t that contain at most k edges.

   (a) Given a directed graph $G = (V, E)$ with non-negative edge lengths $\ell(e), e \in E$, $s, t \in V$ and integer $k > 0$ describe an algorithm that finds the shortest s-t path among all paths that contain at most k edges. *Hint:* You have already seen an algorithm for this before.

   (b) We wish to compute the maximum s-t flow but we want to allow flow on paths with at most k edges; you can imagine useful applications for such a constrained flow. We call this max s-t k-flow problem. However the standard maximum flow algorithms cannot handle this restriction. Here we point out LP based approach for this. Write the maximum s-t k-flow problem as a linear programming problem with one variable for each path $p \in P^{(k)}_{s,t}$. Note that the primal can have an exponential (in $|V|$) number of variables.

   (c) Write the dual of the LP in the preceding part.

   (d) Suppose we are given an assignment of values to the variables in the dual. Show that, even though there are possibly exponentially-many constraints, one can use the algorithm in the first part to efficiently check that the values satisfy all the constraints of the dual. This is sufficient for the Ellipsoid method to be able to solve the LP efficiently.

[1] If the points come an ambient space then S is not necessarily constrained to be from P but we will not consider that here.

Since the problem is somewhat long you can skip formal proofs but briefly explain/justify your answers.

3. Given an undirected graph $G = (V, E)$ it is easy to compute a minimum spanning tree. The Steiner tree problem is a generalization that appears in a number of networking applications. The input is a graph $G = (V, E)$ and a subset $S \subset V$ called terminals. Note that even though we only want to connect the terminals we may need non-terminals in the tree (such non-terminals are called Steiner nodes). The goal is to find a tree in G that contains/connects all the terminals; such a tree is called a Steiner tree for S. In the minimum Steiner tree problem the goal is to find a Steiner tree with fewest edges (one can also consider weights on edges). The decision version of minimum Steiner tree problem is the following: given graph $G = (V, E)$, terminal set $S \subset V$ and integer k, is there a Steiner tree for S in G with at most k edges; we refer to this as the Steiner tree problem. Prove that the Steiner tree problem is NP-Complete via a reduction from the Set Cover problem. *Hint:* Create a bipartite graph from a set system.

4. **Optional problems:** See pset 9 and pset 10 from Fall 2019.