CS 3510 - Honors Algorithms
Homework 8 - Solutions

1. (Dominating Set)

   Clearly Dominating Set is in *NP*. Given a dominating set, one can verify in polynomial time if that is a dominating set. This can be done by taking each vertex and checking if it is either in the given set or one of its edges travel into the set.

   To show that is *NP*-complete, first of all notice that a dominating set has to include all isolated vertices (those which have no edges from them). So let us assume that our graph does not have any isolated vertices. We will show that Dominating Set is *NP*-complete using a reduction from Vertex Cover. Given a graph G, we will construct a graph G' as follows. G' has all edges and vertices of G. Also, for every edge $\{u, v\} \in G$, we add intermediate node on a parallel path in G'. Keeping $\{u, v\}$ intact in G', we add vertex w and edges $\{u, w\}$ and $\{w, v\}$ in G'. Now we will show that G has a vertex cover of size k **if and only if** G' has a dominating set of the same size.

   If S is a vertex cover in G, we will show that S is a dominating set for G'. S is a vertex cover, this means that every edge in G has atleast one of its end points in S. Consider $v \in G'$. If v is an original node in G, then either $v \in S$ or there must be some edge connecting v to some other vertex u. Since S is a vertex cover, is $v \notin S$, then u must be in S, and hence there is an adjacent vertex of v in S. So v is covered by some element in S. However, if w is an additional node in G', then w has two adjacent vertices $u, v \in G$ and using the above argument at least one of them is in S. So the additional nodes are also covered by S. So if G has a vertex cover, then G' has a dominating set of at most the same size (in fact the same set itself would do).

   If G' has a dominating set D of size k, then look at all the additional vertices $w \in D$. Notice that w must be connected to exactly 2 vertices $u, v \in G$. Now see that we can safely replace w by one of u or v. w in D will help us dominate only $u, v, w \in G'$. But these three edges form a 3-cycle, and we can as well pick u or v and still dominate all the vertices that w used to dominate. So we can eliminate all the additional vertices as above. Since all the additional vertices correspond to one of the edges in G, and since all of the additional vertices are covered by the modified D, this means that all the edges in G are covered by the set. So if G' has a dominating set of size k, then G has a vertex cover of size atmost k.

So we have proved both sides of equivalence. A dominating set of size k exists in G' if and only if a vertex cover of size k exists in G. Since we know that vertex cover is an *NP*-complete problem, Dominating Set is also *NP*-complete.

2. (Steiner Minimum Tree)

   Consider the Minimum Steiner Tree T of the graph G and terminal nodes V'. Now think of the tour of the vertices in the tree T as follows. We start from some leaf in T, and do a DFS. Include all the DFS edges in the tour. When we hit a dead end, include the edge in the reverse direction. This way, we cover every edge in the tree T exactly twice. So the cost of this tour is $2w(T)$. Note that this tour will contain all the Steiner nodes as well.

   Now obtain a Hamiltonian cycle of the vertices in V' as follows. For every Steiner node, we will skip that node and jump directly from the preceding vertex to the next vertex. Since the edge weights satisfy triangular inequality, the cost can only decrease from the tour. Now we have a Hamiltonian cycle through all the terminal nodes. If we remove any edge in the cycle, we get a spanning tree T_0 for the set of terminal nodes V'. Now we have the following string of inequalities.

$$w(\text{MST } T') \leq w(T_0) < w(\text{H-cycle}) \leq 2w(T)$$

3. (Generalizations of Max-Flow)

   (a) We can solve this problem by taking the original graph G, and adding a new source node s and new sink t. We create an infinity capacity edge from the new source node s to each of the original source nodes in G, and an infinity capacity edge from each sink node to the new sink t. We can run the usual max flow algorithm, and it is easy to see that sum of the flows from the sources to the sinks in the original graph is maximized if and only if the the flow is maximized from s to t in the new graph.

(b) This can be solved by linear programming. We start from the linear programming formulation of the max flow problem, and also add the additional constraints, which are the lower bound requirements. For any edge (u, v) with lower bound $l_{u,v}$, we add the constraint $f(u, v) \geq l_{u,v}$. The linear program will now find a max flow subject to the lower bound constraints as well.

(c) Here also we can use linear programming. We need to replace the original flow conservation constraints to say that for every $v \in V \backslash \{s, t\}$, $(1 - \epsilon_v) \sum_{u:(u,v)\in E} f(u, v) - \sum_{w:(v,w)\in E} f(v, w) = 0$ (recollect that in the original case the constraints were that the flow was balanced). We can leave the other constraints and objective the same, then we will have a flow that maximizes the flow out of s while taking the loss into account.

(d) We can solve this using two linear programs. We can find the max flow by using the standard max flow linear program. Assume that the maximum flow possible is M. Then we write another program, where the objective is to minimize the cost of the flow $\sum_{(u,v)\in E} \text{cost}(u, v) * f(u, v)$, and an added constraint being that the flow must be equal to the maximum value $\sum_{v:(s,v)\in E} f(s, v) = M$. The usual flow conservation constraint, capacity constraint and the non negativity constraints are there. Solving this LP will get us a maximum flow with minimum cost.

4. (Max Flow Verification)

We can use the following fact. If there is no $s \rightarrow t$ path in the residual graph, then the flow is optimal. Else we can improve by pushing the minimum flow in that path, so it is not optimal.

Assume that the given flow is a valid flow, i.e., the flow at each edge is at most the capacities. Else we can check that at each edge in $O(|E|)$ time.

We can construct the residual graph, this can be done in time $O(|E|)$ (for every edge with capacity c and flow f, add a back edge of capacity f, and reduce the capacity of the forward edge to $c - f$). Then we can use DFS from s to check if there is a path to t. DFS running time is $O(|E| + |V|)$, linear in the number of edges and vertices. If there is a path from s to t, then the given flow is not max flow. If there is no path, then it is indeed the max flow. All the steps are linear time, so the verification is in linear time.