CS170–Fall 2012 — Solutions to Homework 11

Ben Augarten, section 106, `cs170-bo`

November 25, 2012

1. (a) Polynomial time. If removing an subset of L doesn't disconnect the graph, then we can reduce the problem to MST. Remove all of L, find the MST and then add each vertex in L back to the MST. Now each node in L will be a leaf node in the new MST.

   (b) We can reduce Hamiltonian path to Spanning Tree with leaves equal to set L. If there is a Hamiltonian path between v and u then there is a spanning tree with v as the root and u as the only leaf, i.e. $L = \{u\}$. So we input the Graph and $L = \{u\}$ and the spanning tree that we get back represents the path from u to v. If the spanning tree algorithm returns no answer, then there is no Hamiltonian path. Therefore, this problem is NP-complete.

   (c) We can again reduce Hamiltonian path to Spanning Tree with leaves included in set L. We input to this Spanning algorithm the graph G and $L = \{u\}$, the vertex that we want to reach. If there is no Hamiltonian path to u then there won't be any leaves (since u won't be a leaf and no other leaves are allowed), which means that the spanning tree algorithm will fail. If there is a Hamiltonian path to u then u will be the only leaf node in the spanning tree again and we can reconstruct the path by following the spanning tree form v.

   (d) Spanning tree has at most k leaves. We can reduce Rudrata path to spanning tree with at most 1 leaf node (I guess 2 if you count the root). Because Rudrata is NP-complete, so too is Spanning tree with at most k leaves.

   (e) Spanning tree has k or more leaf vertices: just run BFS on G starting from every vertex. The resulting spanning tree with the most leaf nodes is the most leaf nodes any spanning tree of G can have because the tree branches as much as possible, resulting in the most leaf nodes. Of course you can stop as soon as you find a spanning tree with k or more leaf nodes. This runs in polynomial time.

   (f) I think exactly k leaves is NP-complete, but I can't think of a problem that reduces to it.

2. We can solve the Clique problem by reducing it to Maximum Common Subgraph – Input to Max Common Subgraph the graph and the complete graph with k nodes. Binary search on k to find the maximum Clique. Therefore Maximum Common Subgraph must be NP-complete.

   The check is polynomial time – make sure that $V_1 - V_1'$ and $V_2 - V_2'$ are at least b vertices $O(V)$, and check that all of the new edges are in the original graphs $O(|E|)$ so its in NP.

3. (a) Given a solution to Feedback Arc Set, we want to check if the solution is valid in polynomial time. We have a graph G and a set of arcs E'. First make sure that $|E'| < b$. If not, its not a valid solution. Now remove the edges in E' from G, $O(b)$. Now run cycle detection. If there is a cycle, then the solution is not valid, else its a valid solution. We can detect cycles in polynomial time, so checking if the solution is valid is polynomial.

   (b) Vertex Cover: find the minimum set of vertices that cover every edge.
   For every edge in the original graph, we create a cycle in the new graph that we input to FAS. If the edge is from (v_i, v_j), the cycle is from $w_i \to w_i' \to w_j \to w_j' \to w_i$. So, we need to remove one of these edges in FAS, any one of these edges will remove the cycle. So, for every vertex in vertex cover that requires a vertex to be covered, there is a required edge to be removed in FAS. But, how do we deal with one vertex covering multiple edges? Well, in FAS, if v_i covered $v_j, v_k, and v_l$, we can remove edge $w_i \to w_i'$ and remove all of the cycles, thus every vertex in the cover corresponds to exactly one edge removed in FAS. Furthermore, for every vertex v_i not in the cover in the original problem, we don't need to remove an edge in FAS because for every vertex that v_i is connected to, there will be an edge removed in that vertex to prevent the cycle – there must be in order to satisfy vertex cover. Therefore, the size of the feedback arc set must at least the size of the vertex cover and cannot be any larger than the vertex cover, i.e. it must be equal.

   (c) If we are given that G' contains a feedback arc set of size b, then show that G contains a vertex cover set of size at most b. We want a mapping of feedback arcs to vertices in the cover. This can happen if each arc in the feedback set corresponds to a single vertex, i.e. its between a (w_j, w_j'). But, if some of the arcs are between (w_j', w_i) then they don't directly relate to a single vertex that should be in the cover. So, we have to modify the arc set that we get – luckily, for each arc (w_j', w_i) that is in the feedback arc set, we can just change the arc to be (w_j, w_j') which keeps the cycle removed. Now each (w_i, w_i') in the feedback arc set corresponds to a vertex in a cover, so there must exist a vertex cover of at most this size (no reason why this has to be the optimal vertex cover).