University of Waterloo
CS 466 — Advanced Algorithm
Spring 2013
Problem Set 8
Siwei Yang - 20258568

1. [10 marks:] Given a set P of n points in the plane, a minimum Steiner tree is a tree that connects the points of P and has minimum total Euclidean (i.e. L_2) length. For example, for 3 points p_1, p_2, p_3 forming an acute triangle, the minimum Steiner tree has leaves p_1, p_2, p_3 plus a node of degree 3 in the middle of the triangle at the Fermat point where the 3 edges form angles of 120°(Look at Wikipedia for examples).

   Prove that there is a polynomial time 2-approximation algorithm for the minimum Steiner tree problem in the plane. In particular, show that the minimum spanning tree of the points(using Euclidean distances as edge weights in the complete graph) is a 2-approximation.

   Given a set P of n points, let $T^* = (V, E)$ be the tree of minimum total Euclidean length where $P = V \cap P$ and $P' = V \setminus P$.

   Clearly, P' is the set of points added to form the minimum Steiner tree T^*. Now, create a multi-graph:

   $$G = (V, E \cup E') \; where \; E' = \{(i,j) \mid (i,j) \in E, \{i,j\} \cap P' \neq \emptyset\} \quad (1)$$

   Now, define a few building blocks for transforming G:

   - Consider an arbitrary node $v \in P'$, between v and any of its neighbours nodes, there are two edges from construction of G. And, if v is removed from G, each node belongs to a different component(T^* is a tree, and G doesn't have any edge that's not in $T*$), thus v has two edges to every component of $G \setminus v$ which also means v has two edges to every neighbouring component of $G \setminus P'$. Thus, we have a condition, ϕ, holds true: **For arbitrary node $v \in P'$, between v and any of its neighbours nodes that belongs to P', there are two edges; between v and any of neighbouring component of $G \setminus P'$, there are two edges**; G is connectted.

- if there exists node $v \in P'$ such that v is neighbouring with at least two components of $G \setminus P'$ and ϕ holds:

  Let two of those compoents be C_1, C_2, and v is connectted to C_1 with $(v, v_{1,1})$ and $(v, v_{1,2})$, C_2 with $(v, v_{2,1})$ and $(v, v_{2,2})$.

  Replace $(v, v_{1,2})$ and $(v, v_{2,1})$ with $(v_{1,2}, v_{2,1})$ to produce G'. By triangle inequality, we know the weight of graph is not increasing. And, now v has two edges to the newly formed component of $G' \setminus P'$. Thus, ϕ holds in G'.

- if there exists node $v \in P'$ such that v is not neighbouring with any components of $G \setminus P'$ and one node $v' \in P'$ and ϕ holds

  Remove v to produce G'. We know the weight of graph is not increasing. And, ϕ holds in G'.

- if there exists node $v \in P'$ such that v is neighbouring with only one components of $G \setminus P'$ and none of the nodes in P' and ϕ holds

  Remove v to produce G'. We know the weight of graph is not increasing. And, ϕ holds in G'.

- if there exists node $v \in P'$ such that v is neighbouring with only one components of $G \setminus P'$ and one node $v' \in P'$ and ϕ holds

  Let the component be C, and v is connectted to C with (v, v_1) and (v, v_2).

  Replace (v, v_1) and (v, v') with (v_1, v'), replace (v, v_2) and (v, v') with (v_2, v'), and remove v to produce G'. By triangle inequality, we know the weight of graph is not increasing. And, now v' has two edges to component C of $G' \setminus P'$. Thus, ϕ holds in G'.

So, G starts off with condition ϕ, and by iteratively applying any of the four transformation steps until no step can be applied, we end up with a graph G^*. The following can be concluded:

- Obviously, for any node $v \in P'$ that remains in G^*, it is neighbouring with at most one component from $G^* \setminus P'$.
- This further implies it is neighbouring with at least two other nodes in P', otherwise v would have been removed.
- Therefore, if there is a node $v \in P'$ in G^*, then we can recursively do depth first search on nodes from P' until a cycle is found.

- But, **this is a contradiction as this would imply a cycle in G, thus a cycle in T^*.**

Thus, we conclude there is no node from P' that is in G^*. And also, not nodes in P is removed during iteration. Therefore, G^* is a connectted graph spanning over all nodes of P. This implies a tree, T, that covers all P is a subgraph of G^*. Let the complete graph of P nodes be G_P, then we have:

$$2*weight(T^*) \geq weight(G) \geq weight(G^* \; geqweight(T) \geq weight(MST(G_P)) \tag{2}$$

Thus, we prove that the minimum spanning tree of a complete graph of P nodes is a 2-approximate solution for the minimum Steiner tree.

2. [10 marks] Given a directed graph, G, represented by its adjacency matrix. (Lets assume there are no loops i.e. no edges (i,i).)

   - [6 marks]Give an efficient algorithm to determine what pairs of nodes have directed paths of length exactly $n-1$. Give the runtime of your method and justify this runtime.

     Let M be the adjacency matrix.

     - Calculate series M^{2^i} up to $i = \lceil \ln(n-1) \rceil$. By using Dynamic Programming, we only need to spend time calculate $M^{2^{\lceil \ln(n-1) \rceil}}$ which takes roughly $O(\lceil \ln(n-1) \rceil)$ time assuming size of M is constant.
     - Then find $I \subset \{1, 2, \ldots, \lceil \ln(n-1) \rceil\}$ such that $\sum_{i \in I} 2^i = n-1$.
     - Calculate $M^* = \prod_{i \in I} M^{2^i}$ which takes roughly $O(\lceil \ln(n-1) \rceil)$ time as well.

     Inspect the entries of M^*, whenever $M^*_{i,j} > 0$, there is a path of length $n-1$ from node i to node j.

   - [4 marks]We know the Hamiltonian cycle problem is NP-Complete. The Hamiltonian path problem, of having a path go though each node exactly once, is also NP-hard. Explain this apparent anomaly, given that you have already given an efficient algorithm to find paths of length n-1.

     The path found using the algorithm given doesn't take into account a node can only be visit once. If the checking were to be

incorporated into the algorithm, then each entries of the matrix have to keep track of all the paths associated with it. And, during the multiplication step, the paths from two entries have to be cross checked to filter out the bad paths. These extra overhead will drive up the runtime to reflect it's NP-hardness.