CS 344 Assignment 5

Craig Perkins, Alex Tang, Steve Grzenda

Due May 6, 2014

Problem 1

(a) Assuming the function s satisfies metric requirements, the algorithm for this problem is the same as the k-Clustering approximation algorithm using $s(u_i, u_j)$ as the distance function. The algorithm goes as follows (taken from the textbook):

```
Pick any point u_1 as the first cluster center
for i = 2 to k:
  Let u_i be the point that is farthest from u_1, ... , u_i-1
    based on the function s
return k clusters
```

The k clusters that this function returns will be the k groups that satisfy the optimality requirement mentioned in the problem. Assign the rest of the users to the closest cluster.

(b) Each time you find a cluster center u_i, you need to create a max heap for all other non-cluster vertices $u_1, \ldots, u_{i-1}$ to determine the next cluster center to pick. The first iteration will require calculating one max heap for u_1, the second iteration will require calculating calculating two max heaps for u_1 and u_2, so the k-th iteration will require calculating $k-1$ max heaps for $u_1, \ldots, u_{i-1}$. This is a total of $1 + 2 + \cdots + (k-1) \approx k^2$ calculations of max heaps. Each max heap calculation requires $nlogn$ time for a total running time of $O(k^2 nlogn)$.

(c) The k clusters will be as far away from each other so any two users that are not centers cannot possibly farther away from each other than the

cluster centers. Therefore, this will minimize the similarity of the most similar pair of users that have been assigned to different groups.

Problem 2

A. For $d = 2$, the problem reduces to finding a RUDRATA-PATH in the graph. The MST for $d = 2$ is the same as a RUDRATA-PATH throughout the sensor nodes network, with the two endpoint nodes of the path having degree = 1 and every non-endpoint intermediate node having degree = 2. The RUDRATA-PATH problem is NP-complete so this problem is too.

B. Solving TSP is the same as solving RUDRATA-PATH so it will solve this problem too.

Problem 3

1. The problem is that they need to find the maximum number of edges between two sets of vertices, similar to a bipartite graph. Let these two sets be U and V. Then the cost function is $\sum_{x \in U, y \in V} e(x, y)$.

2. A local search approach would be to start with 1 person in table A and the rest in table B. Move 1 person over to table A and if it improves the cost then keep that person there permanently. Repeat this process until there is no improvement.
   This is a 2-approximation polynomial time solution because the optimal solution would have cost = m. In the worst case (where everyone hates everyone), the graph is a complete graph with number of edges $m = \frac{n(n-1)}{2}$. The search will stop when each table has $\frac{n}{2}$ people with cost = $\frac{m}{2}$ edges between them.

3. A greedy solution would be to find the person who hates the most people. Put that person in table A. Pick the next hateful person and place him in table B. Repeat until everyone is placed at a table.
   This is a 2-approximation polynomial time solution for the same reasons as part 2.

4. A randomized approach would be to queue up all the names and alternately assign to table A and B.

This is a 2-approximation polynomial time solution for the same reasons as part 2.

Problem 4

a. One way to do this would be to use Karger's algorithm which is a randomized algorithm that contracts edges of a graph. Since it is randomized it does not always work; however since it runs in polynomial time we can run it multiple times and take the min of the min cuts. The edges in the min cut between the two cities would be where the soldiers need to be placed.

b. A solution to this problem would be to block all roads in between the cities. Then queue up all the edges. Pop an edge and return it to the graph. Perform 3 DFS's (A to B, A to C, B to C). If a DFS succeeds then that road must be blocked. If it does not succeed then the edge can be returned to the graph. Repeat until the queue is empty.

c. Once again remove all edges. Add in one edge and perform Floyd Warshall. If the number of infinities equals m-1 then stop. Else save the minimum number of infinities that is $\geq m$. Continue adding in edges until there is no more improvement

d.