CS 170 Project Design Document

Xiaochen Han, Bofan Xue, Boyan Zhou

April 17, 2019

1 If there was a Guavabot on every node, what sequence of scouts and remotes would you do to get them all home?

Find the MST of the graph, treat home as the root of the tree. Then run DFS from home on the MST, and remote every node to its parent in post order of DFS untill all bots are returned home, merging naturally in the process. And the cost will be the sum of all edge weights of MST.

2 More generally, if you knew where all the bots were, what sequence of scouts and remotes would you do to get them all home?

(a) No scout is needed. Construct multiple of MST, such that each MST only contains vertices that have bot in it.

(b) Run Dijkstra's algorithm starting from home. This computes the shortest path from home to all of MST's. Store all paths and their length in list D. Sort list D from shortest to longest.

(c) Remove the shortest path in D, which correspond to a closest MST m. Draw a radius $f(D, m)$ circle starting from MST (where $f(\cdot, \cdot)$ is a to be determined function. The naive case would be $f(D, m) = D$), and remote each neighbor MST that is inside this circle to m, via the shortest path between these two MST's. Also remove each of such neighbors from list D. Repeat this process until all L bots are remoted to home.

3 If you didn't care about getting the bots home and just wanted to find their locations as quickly as possible, what sequence of scouts and remotes would you do?

- Scout all the vertices, using all students.
- Sort all vertices based on students' votes, from high to low.

- Then remote each vertex to its nearest neighbor, according to its vote.
- Record the number of bots moved from each remote, stop when we have found all L bots.
- Specifically, record the start and end vertices of each remote to avoid double counting.

4 What ideas do you have for solvers? Please provide at least 2. What are their advantages/disadvantages?

Define $P(i)$ as the proportion of student votes that indicates a bot is present at vertex i.

(a) *Algorithm 1*
Pseudocode:

- Scout all the vertices, using all students.
- Sort all vertices based on $P(i)$, from high to low. Store result in list A.
- Starting with vertices with high $P(i)$, remote it to its nearest neighbor.
- Record the number of bots moved from each remote, stop when we have found all L bots. Specifically, record the start and end vertices of each remote to avoid double counting.
- Run algorithm described in question 2

Advantages: Deterministic model. Potentially optimal remote ordering once we have all locations of all bots, and thus a lower cost.
Disadvantages: Worst case O(V) remote to determine bot locations, depending on student reliability.

(b) *Algorithm 2*
Pseudocode:

- Scout all the vertices, using all students.
- Sort all vertices based on $P(i)$, from high to low. Store result in list A.
- Run Dijkstra's algorithm, starting from home. This computes shortest path from home to each of the vertices.
- For each vertex v in A, remote it to home according to shortest path computed in Dijkstra's algorithm. If v does not contain any bot (which can be realized once the first remote is performed), remove v from A, pick the next closest vertex, and rerun the previous step; otherwise, remove from A all vertices that are in the shortest path from v to home and that are also in A. Repeat until all L bots get remoted to home.

Advantages: no potentionally suboptimal remote to determine bot location.
Disadvantages: probability of remoting empty vertex depends on student reliability. Also depending on graph, each remote all the way back home may not carry additional bot back, which may cause suboptimal remotes.

5 What kinds of inputs do you think your solvers will do well on? Do poorly on?

No matter which solver we adopt, if $|S_i| = |V|/2 \ \forall i$, which means each students is as much unreliable as possible, then both our solvers will suffer from weak performance (The only solver that does not suffer from this is blindly sort all edges in graph, and remote lowest weight edges first). The rest performance issues are specific to graph.

(a) *Algorithm 1*
It will do well in case where bot containing vertices are neighbor to each other, so that they can form a single MST. It will do poorly when each bot containing vertex is far away from other bot containing vertices.

(b) *Algorithm 2*
It will do well in case where all other bots locate along the shortest path between one bot and the home, and that one bot gets remoted first. It will do poorly is we have the same graph setup but instead remote vertices that are closest to home to vertices that are farthest to home.