CS170 Project Design

Qitian Liao 3031980823, Zuojun Zheng 3033242006
Conquer and Divide

1. Because we know every vertex has a Guavabot, we can always keep track of their locations and thus we do not have to call *scout* at all. We use Kruskal's algorithm to find an MST T of G. Then consider T as unweighted and run DFS on T starting from H and keep track of the "prev" pointers. Call *remote* on edge (u, v) where u is in the reversed linearization order and v is the vertex referred by u's prev pointer.

2. In this more general case, we know the locations of each Guavabot. Similarly, we do not need to call *scout*. It is best to avoid exploiting new edges and use only the "inevitable" edges. An "inevitable" edge is defined as an edge on which we have to call *remote* no matter where the other bots are located at. Intuitively, we want to herd every bot in the neighborhood to a specific vertex, then send the entire butch of bots to the next vertex and repeat the process until they reach home. We run Prim's Algorithm with a modified priority queue to find an MST T of G. We prioritize the edges with vertices that have bots in it. More specifically, suppose there is a cut S such that G is partitioned into S and V-S. (u, v) and (u', v') are both minimum weight edges that cross the cuts. Then we prioritize (u, v) if more vertices in {u, v} has bots than vertices in {u', v'}. Then consider T as unweighted and run DFS on T starting from H and keep track of the "prev" pointers. Call *remote* on edge (u, v) in the reversed linearization order of u and v is the vertex referred by u's prev pointer.

3. First, let all the k students scout all the vertices. Then, call *remote* on the vertices in the order of majority vote, i.e. call *remote* on the vertex that has been confirmed to be "True" by the most students first, specifically, move the bots from this vertex to its closest neighbor (along the lowest-weight adjacent edge). Meanwhile, we also keep track of the number of bots "n" that have been moved and located so far. Continue the process until n = |L|, by which time we have found all the bots their respective locations.

4. **(a)** Naive algorithm: proceed with the algorithm in part(1) to move all the bots home.
Advantage: Easier implementation
Disadvantage: Significantly longer runtime since we did not take advantage of the possibility of avoiding redundant edges.

(b) Use the algorithm in part(3) to find the locations of all the bots. Then proceed with the algorithm in part(2) to move them home.
Advantage: Better runtime since we try to only use the "inevitable" edges and avoid using the redundant ones. Since we also rank the vertices according to the majority vote, we start with the vertex that is most likely to contain a bot. Thereby, we also reduce the total cost of the *remote* operations as opposed to starting with a random vertex.
Disadvantage: Harder implementation since we have to call *scout* multiple times and rank the vertices according to their majority vote. We probably need to construct a priority queue to substantiate the ranking.
Possible improvements: Use Multiplicative Weights Algorithm or Machine Learning (the one we used in HW12) instead of Majority Vote.

5. The first algorithm works best if the bots are uniformly and universally distributed. This means that each vertex has one bot from the beginning. Then we can directly apply part(1) without having to worry about where the bots are located at.
On the other hand, it does not work so well when the bots are scattered around in the graph. In this case, there is no guarantee that the leaf node contains a bot, so it is unnecessary to start the search in the reversed linearization order, which would result in excessive runtime. Besides, the algorithm does not take full advantage of "inevitable" edges, as the priority queue only recognizes the lowest-weight edges, but we should actually avoid using new edges even if they are a little lower-weighted.

The second algorithm works well if the bots are scattered on the graph. In this case, it is best to locate the vertices first before foolhardily trying to start calling *remote* right away. In other words, it is better to have an educated guess first.

It does not work so well when a large portion of bots lie on the vertices which most students tend to lie about. In this case, these vertices would rank relatively low by the majority vote. But in truth, they should be ranked much higher. In other words, the priority queue we constructed is rendered useless and might even backfire. The algorithm also does not work well when there are bots on most of the vertices, in which case it is better to skip step (4) and assume there are bots on every vertex. It is no longer necessary to find out the exact locations since we would not be severely punished if we assume there are bots on every vertex.