(1) Prove or give a counter-example for the following claim: If an undirected, connected graph G has all unique integer edge weights, then the Minimum-Spanning Tree of G must be unique as well.

Proceed with contradiction. Let's assume there is more than one cheapest tree. Tree1 and Tree2 are different trees but are the same total weight. To be different they need to have at least one different edge: call that one different edge Edge1 and it is ocntained in Tree1 but not Tree2. That corresponding edge in Tree2 will be called Edge2. Edge1 is less than Edge2 in this scenario byt 1. Since all edges are coming from the same graph, including Edge1 in TreeB would create a cycle. Now, if we have a cycle, we could just replace Edge2 with Edge1 in B, no longer have a cycle and we have a better/more optimal MST. By the definition of MST, it is the minimum weight, so clearly TreeB was not an MST to begin with and we have a contradiction.

(2) Layover Optimization

We would use a modified form of Dijkstra's here. The initial setup is the same, up until the while-loop.

If the adjacent node being examined is unseen, and it it's start city is the desired destination (B), then update it's layover time to be the layover time of the current node + the layover time from the current node to this adjacent node./ INsert it into the priority queue and add it to a l ist of valid flights, then break the loop because we've reached the desired destination (B).

If the adjacent node is unseen and it's start city is not B, update the layover time normally, add to priority queue and add to list of valid flights.

If the adjacent node has been seen and the new layover time is better/ shorter than the old one recorded, update it's layover time using decreaseKey().

At the end of the while loop, return list of valid flights.

(3) Robot Path Crossing Algorithm

First, we create a separate graph that defines the possible positions that the robots can be in at any given time. Each node in the new graph would define 2 nodes in the initial graph. These 2 nodes originally show the position of the 2 robots at any given moment. With this, we can take possible collisions and handle them accordingly. We then use graph traversal to grab the optimal path. We would start graph traversal at (s1,s2), finding all nodes level by level and then finall reaching the desired (d1,d2) node. Along this path, we add the non-collision, compatible edges to the final path then return that path.

Worst Case Runtime: The total worst case runtime would be Theta(node runtime + edge runtime). Each robot goes thorugth one edge using n/r-1 nodes and be bound by r(n/r-1)

Robot Numbers Growing: As the number of robots grows, the algorithm would get faster. There are less available nodes for traversal options because there are more nodes occupied with robots. The runtime then changes to (nodes/robots)^robots.