Scenario 1 Algorithm

First we create the vertices which represent the sensor locations taken from the input ('loc' array). Once we have the list of vertices, we create a complete graph containing all vertices on the map, with the weights representing the distance between a vertex pair (u,v), using an adjacency matrix. This graph object is then translated to a second graph object which has a class method function to create the Minimum Spanning Tree (MST) of the graph using Kruskal's Algorithm. Once we have the MST, we create a tree dictionary, which represents an adjacency list of the MST. We then do a Depth-First-Search, starting at the sink vertex (the center sensor location), of the adjacency list which creates the path of the drone.

To take in account the fact that the big drone can drop sensors for all locations in a 5x5 square, we first begin traversing through the path we previously calculated. If the current vertex is 'unvisited', we append it to the final path. We then check the locations of all other vertices on the map to see if any are within the 5x5 square centered around the current vertex we're visiting. We mark any that are as 'visited' so we know that they should not be included in the final path. We continue this process through the entire list of vertices and then finally append the sink vertex to form a Hamiltonian cycle so the drone visits all locations and returns to the center.

Scenario 2 Algorithm

The algorithm for scenario 2 is similar to scenario 1. Given m given drones, we split up the map into m horizontal sections. We create m lists of vertices with each list containing all vertices within boundaries of that section. With m lists of vertices, we then create m complete graphs with the weights representing distances between a vertex pair (u,v). From these m complete graphs, we then create m MST's. We do a Depth-First-Search traversal for each MST to create the drone's path for each of the m sections.

To take in account each of the m drones fuel and sensor capacity, we adjust their path previously calculated. First we make its start position the sink vertex. We begin iterating through the path previously calculated. If the drone has enough fuel to visit the current sensor and return home as well as visit the next two sensors and return home, along with enough sensors, we append the current vertex to the path, subtract that distance from the fuel and decrement the number of sensors it has. This is to ensure that the drone has enough fuel to return home at the next vertex. If the drone doesn't meet these conditions, it flies back to the sink, refills on fuel and sensors, then flies back to the next unvisited vertex in the path. It continues this until all sensors in its section are visited, then flies back to the sink vertex.

Problem 1 Results

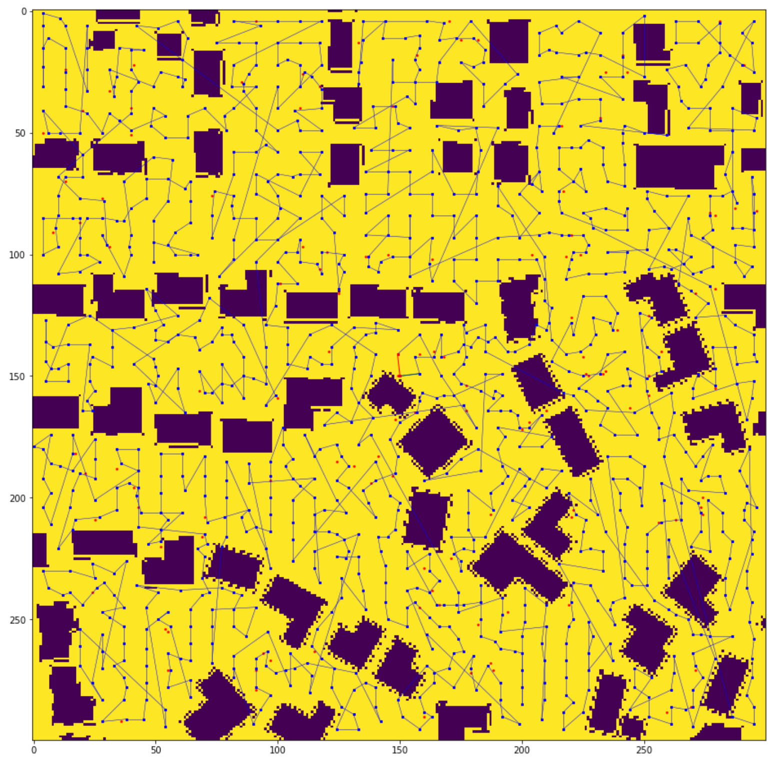

Figure 1: Trajectory of drone for scenario 1. Red points denote sensors that were visited through the 5x5 proximity of the drone. *Total distance: 11,821.4 units*

Problem 2 Results

Number of Drones	Deployment Ratios
1	15073.42
3	1653.74
5	622.79
8	238.68
10	164.31

Figure 2: The optimal deployment ratio is 164.31 which is using 10 drones.

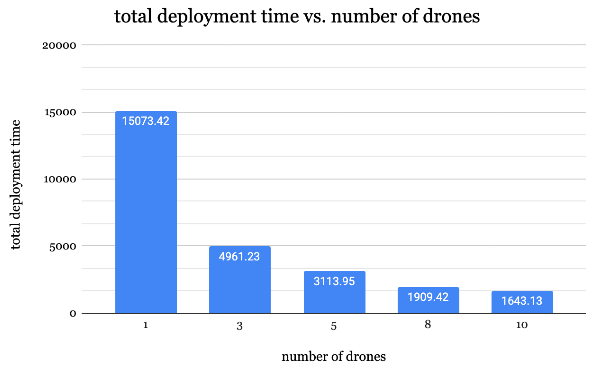

Figure 3: Total Deployment Time vs. Number of Drones Deployed

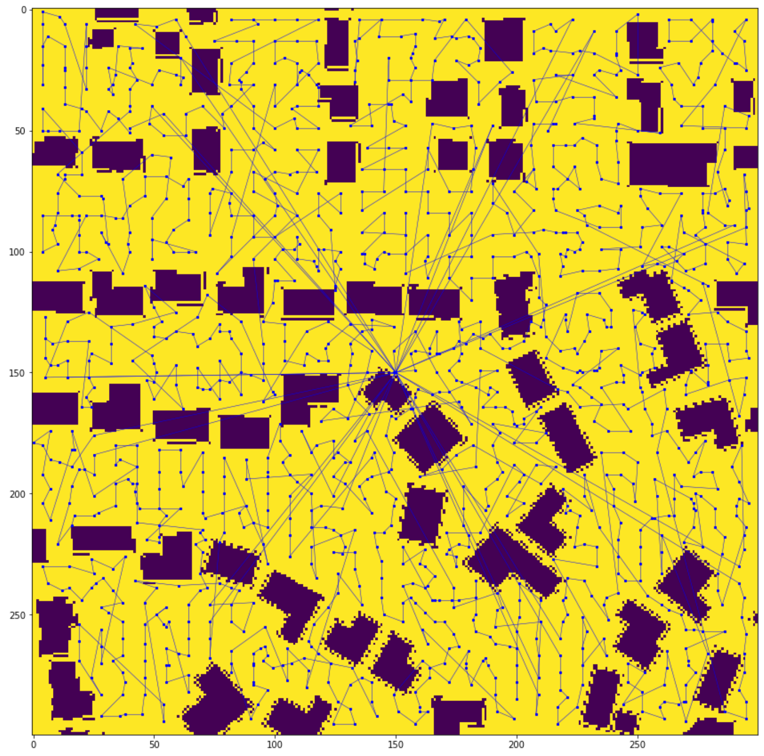

Figure 4: Trajectory for m=1 drones

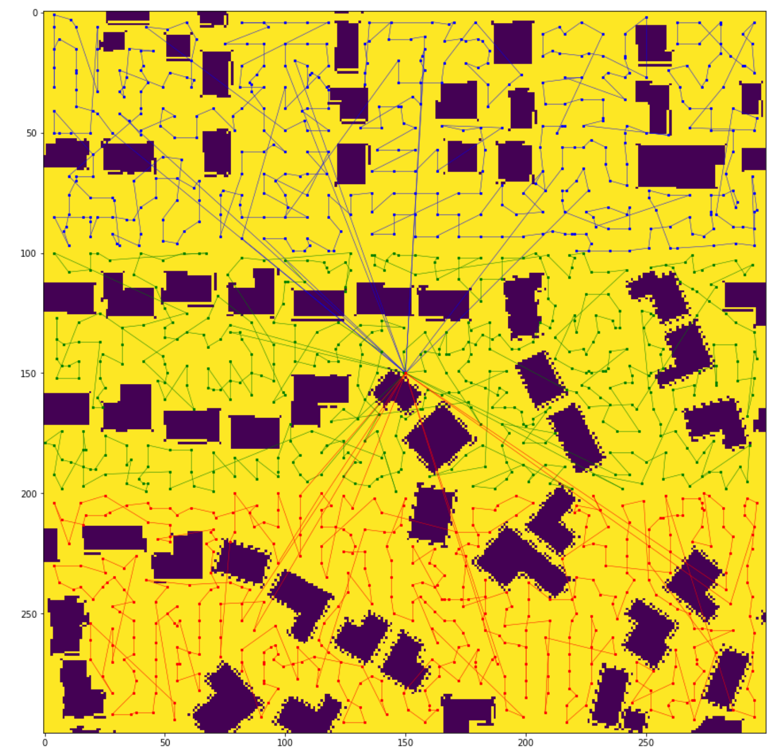

Figure 5: Trajectory for m=3 drones

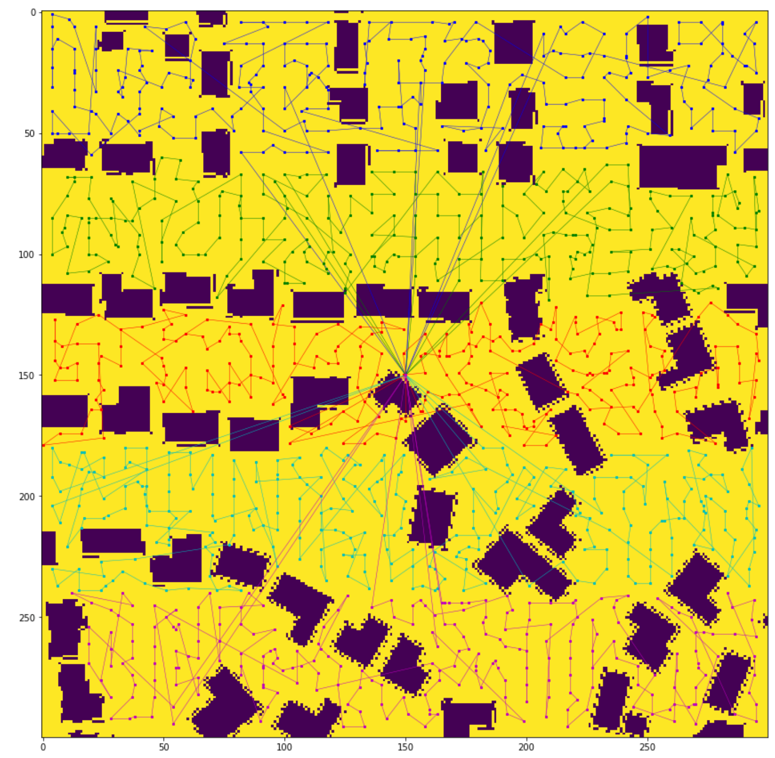

Figure 6: Trajectory for m=5 drones

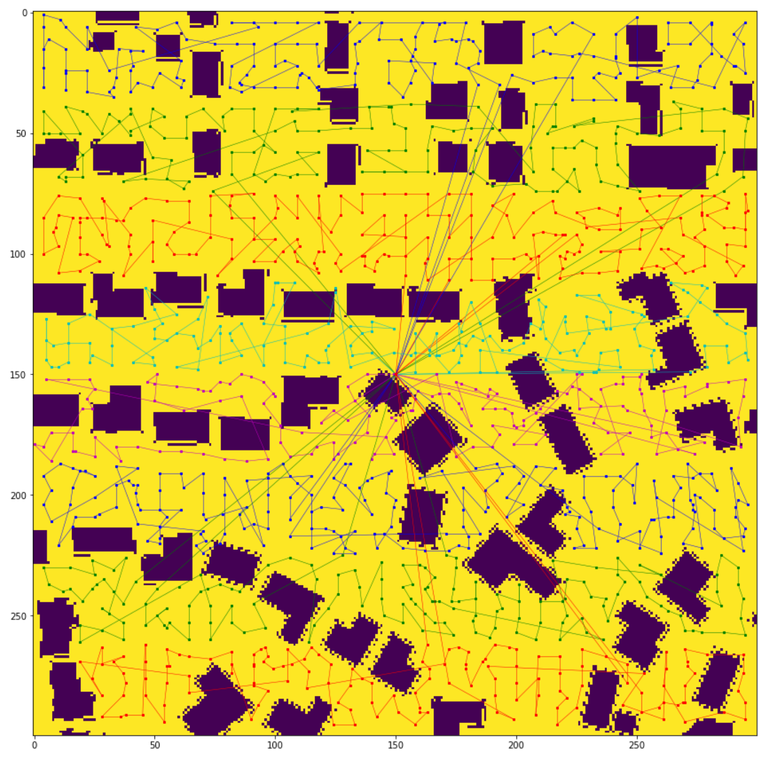

Figure 7: Trajectory for m=8 drones

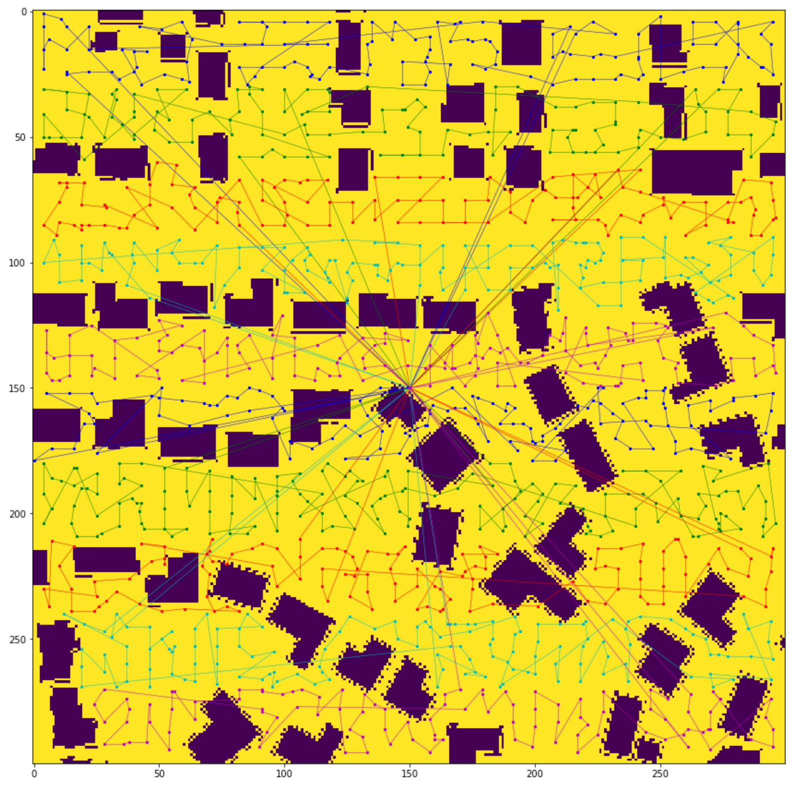

Figure 8: Trajectory for m=10 drones