SENSOR NETWORK SYNTHESIS - A GRAPH THEORY ALGORITHMIC APPROACH

PRESENTER: YU-WEI VINCENT YEH

PROJECT PARTNER: KELLEN CHENG

COURSE: UCLA ECE 134, GRAPH THEORY IN ENGINEERING

INTRODUCTION

The study of wireless sensor networks has drawn great interests within disciplines such as area monitoring, environmental detection, and space defense. Through an optimal placement of sensors, environmental obstructions of the signal could be mitigated while maintaining good connectivity and coverage - the connectivity and coverage that arise from this optimal placement are the critical figures of merits to this type of study. Thus, the goal of this project is to place the minimum number of sensors, where each sensor covers a 9x9 area, on a map of size 300x300, while simultaneously optimizing the network's connectivity and coverage. Afterwards, we'd like a routing algorithm to effectively deliver sensors. Thus, there are two parts to this study:

1. Part one: To create highly generalizable algorithms for sensor deployment and connectivity calculations

2. Part two: To algorithmically find optimal traveling paths for drones to drop sensors at their desired locations

Link to code & specifications:
https://drive.google.com/drive/folders/1IsX8hy8pg_Ibl5P0FF_NvGSchHjmbz5R?usp=sharing

SOME DEFINITIONS:

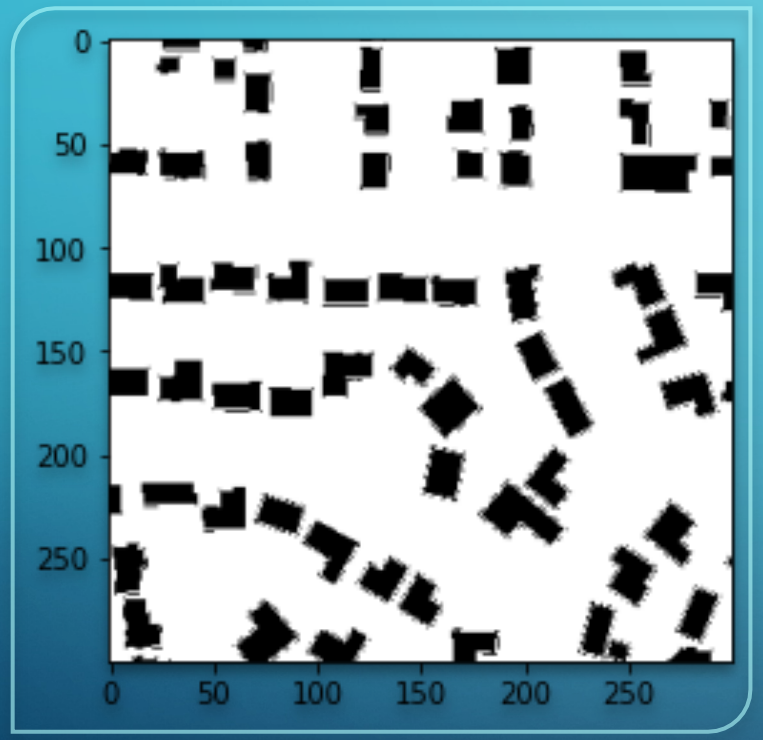
Part one:

- Coverage: Each sensor has a fixed number of blocks in which the signal could reach, and coverage is simply the percentage of the map that was covered by the sensors deployed.
- Connectivity: Generally, if two sensors are located far apart, or had an obstruction in between (walls, buildings, etc.) the connectivity value would be low and vice versa.
- Average/minimum min-cut: The average/worst connectivity between a sensor and the middle coordinate of the map.

Part two:

- Minimum set cover: Assuming we have some buckets (sets) that contain some number of balls in different colors, and a total of 20 colors of balls. A minimum set cover is the minimum number of buckets we could pick that includes all 20 colors.

PART ONE: INTRODUCTION



- In part one, we were given a 300x300 grayscale map (shown on the left) that includes obstructions in black (buildings, walls, etc.), and free spaces where the sensors should be deployed.
- Each sensor covers a 9x9 block, but sensors could communicate with each other no matter how far apart.
- There are tradeoffs between coverage and connectivity. Thus, the goal was to minimize overlaps between sensors, while not placing them too far from one another.

PART ONE: ALGORITHM DESIGN

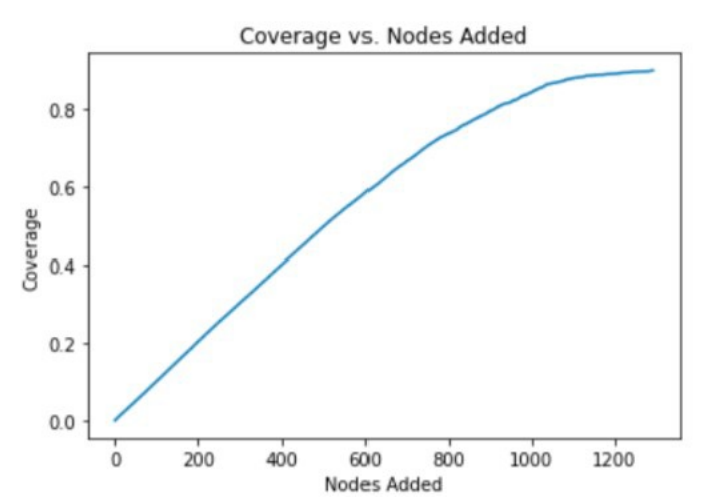
- Some miscellaneous yet somewhat important graph-theory-parameter-calculating functions include: max_span, min_cut, worst_mult_cut, and comm_rate. Please see the attached code for specifics.
- check_obstruction: The check-obstruction function takes two coordinates as inputs, then returns whether there is an obstruction between the two locations. We assume the coordinates represent the middle of some block, so if two sensors lie on the same row or column, checking whether there is an obstructed coordinate would suffice. If the two sensors are on different rows and columns, we fit a line that passes through both coordinates, and depending on the slope, we either traverse through the row or column coordinates. If the slope is less than 45 degrees, we traverse through the columns, and vice versa.

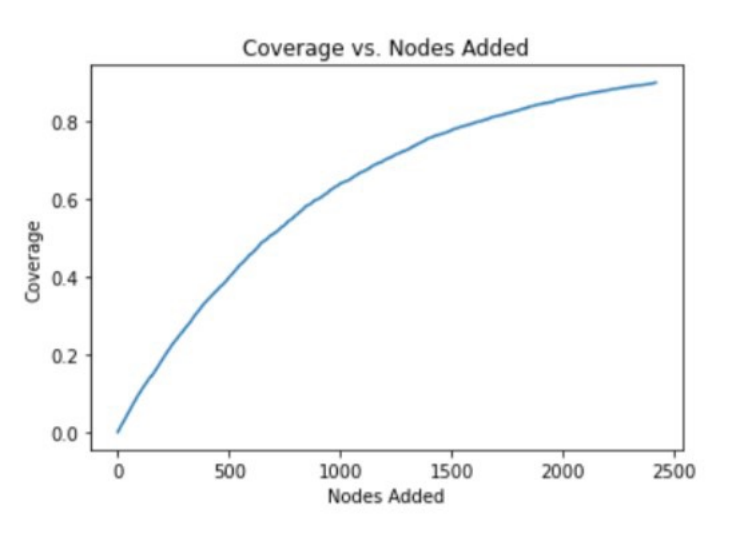
PART ONE: ALGORITHM DESIGN

- add_one_node: The add-one-node function takes three arguments: G, map, and sensor-map. It employs a waterfall thresholding technique to first search for the most optimal placement locations, utilizing a set of three different thresholds. As always, a new node is always tied next to an ”anchor” node, so that the algorithm will not place a subsequent node on the opposite end of the map. Within this anchor, it will search a predefined square of surrounding coordinates, choosing the location that meets the coverage threshold. It will fall to the next threshold if no option is located, until it reaches completely random placement at the fourth threshold. To avoid connectivity issues, all nodes are placed within a range of this anchor node, so that a node is guaranteed to be within a certain distance to another node in the network.

PART ONE: RESULTS

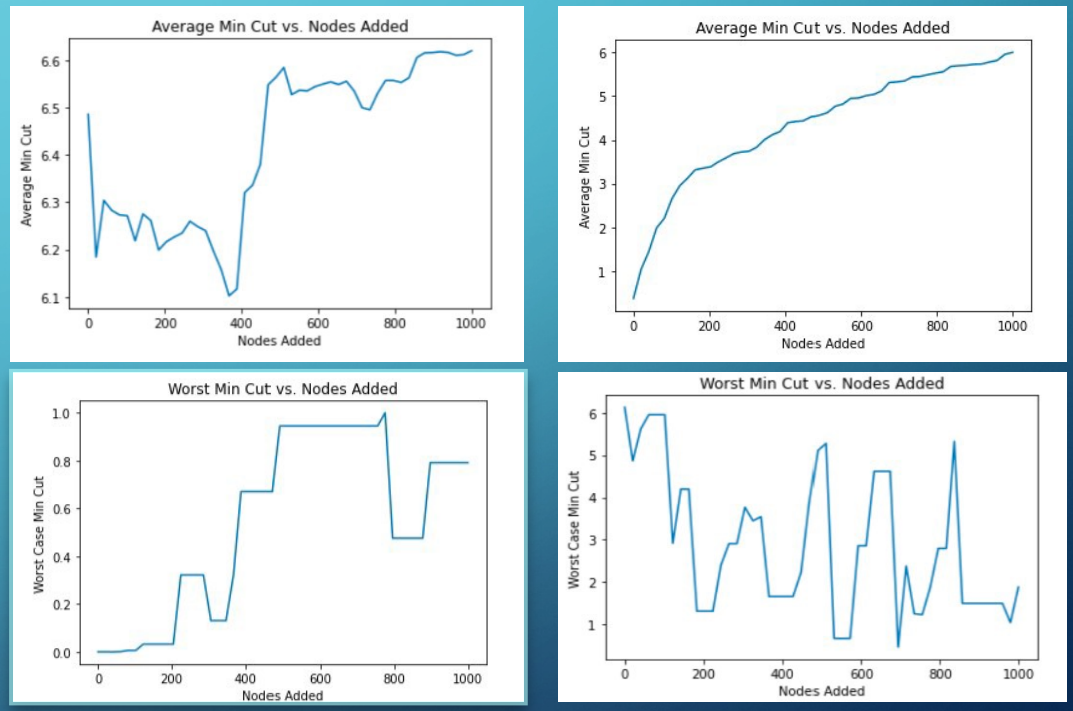
- The waterfall sensor placement algorithm that we developed not only achieved a coverage that outperformed random placement methods by 35-40%, but also possessed a far superior connectivity.
- As shown in the following plots, Our algorithm required 1293 nodes to reach 90% coverage, while the random placement required 2419 nodes to reach 90% coverage. Below are the figures required for 90% for our algorithm (left) and random placement (right):



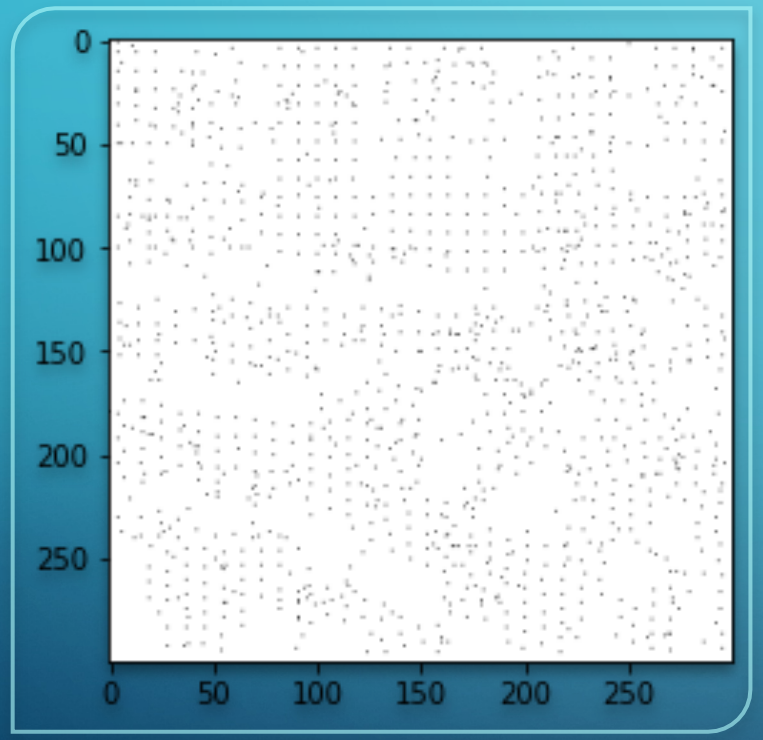


PART ONE: RESULTS

- The following are the subplot figures illustrating the worst-case minimum cut performance for our algorithm (lower right) and random placement (lower left), and average minimum cut performance for our algorithm (upper left) and random placement (upper right)



PART TWO: INTRODUCTION



- The goal for part two was to determine the trajectory of drones to deploy sensors in the locations shown on the left.
- The sensor allocation generated by our algorithm from part one was adopted as the basis of part two for the entire class.
- We aimed to minimize the total distance travelled by the drones to achieve minimum deployment time, under two different scenarios: 1. One giant drone with no fuel constraint 2. Multiple drones deliver simultaneously with fuel constrains

PART TWO: ALGORITHM DESIGN

- Some miscellaneous yet somewhat important graph-theory-parameter-calculating functions include: num_sensors_eliminated, min_set_cover, edges_path, and path_length_calculation. Please see the attached code for specifics.
- square_cycles_size_n: The goal of this function is to output a dictionary that maps the list of sensor locations covered to every possible three-sided square on the grid, that would later be used in the minimum set cover algorithm. This algorithm takes in three parameters: n, the side length of the arbitrary squares, grid, the map that the sensor is on, and all loc, the location of all sensors. The side length n acts as a hyper-parameter for fine tuning in order to achieve the shortest cycle length, and it was determined that n=6 produces an optimal result. The reason to use only three sides of squares is because to create a cycle, an entry and exit point is required for the line segments. However, if we want to traverse a full square with 4 sides, determining where to enter and where to exit would create unnecessary complexities. Furthermore, using consecutive sides not only allowed us to string the segments into a path more easily, the computational cost associated with the ”stringing” is also extremely low.

PART TWO: ALGORITHM DESIGN

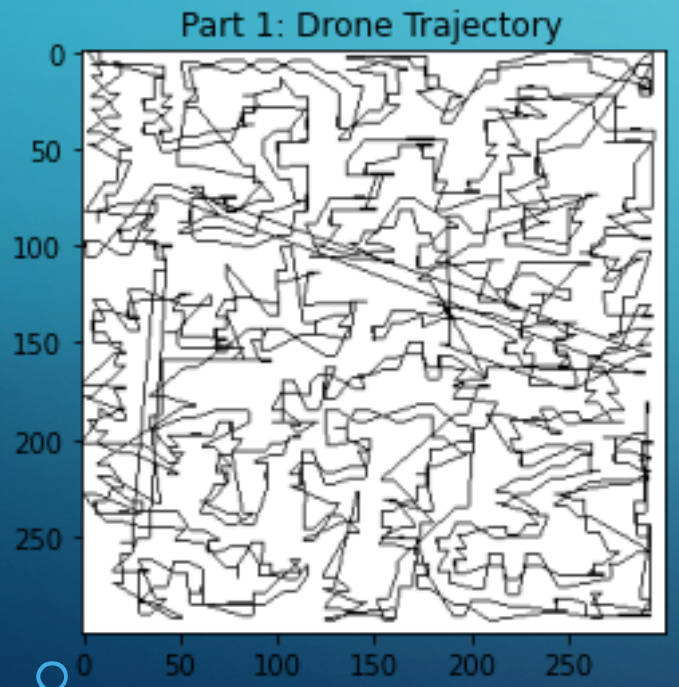
- edge_connecting_algorithm: This algorithm employs a forward-back and back-forward stacking, reverse augmented binary search approach to construct a cycle based on the inputs: the center location in which the drone starts and ends, and the edge set determined by the minimum set cover algorithm that we want to string together. The algorithm first goes through every edge in the set cover to combine edges that connect with each other into sets, as they should be in our original construct of the ”semi-squares.” Then, the algorithm produces a cycle that starts with center loc and ends with center loc by repeatedly finding the 1st and 2nd closest set of edges to the latest set that we added to the path array, such that we build the path concurrently in the forward and backward directionthat contains all sets of edges that the drone traverses.

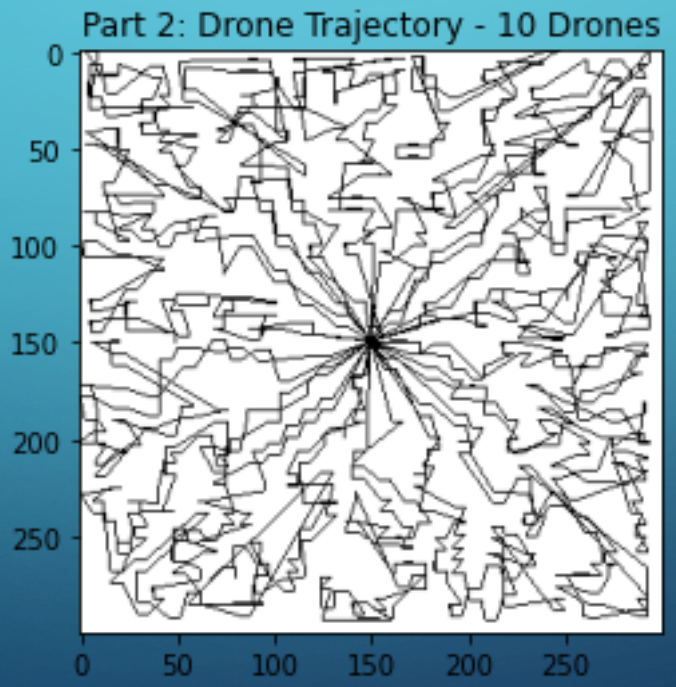
PART TWO: ALGORITHM DESIGN

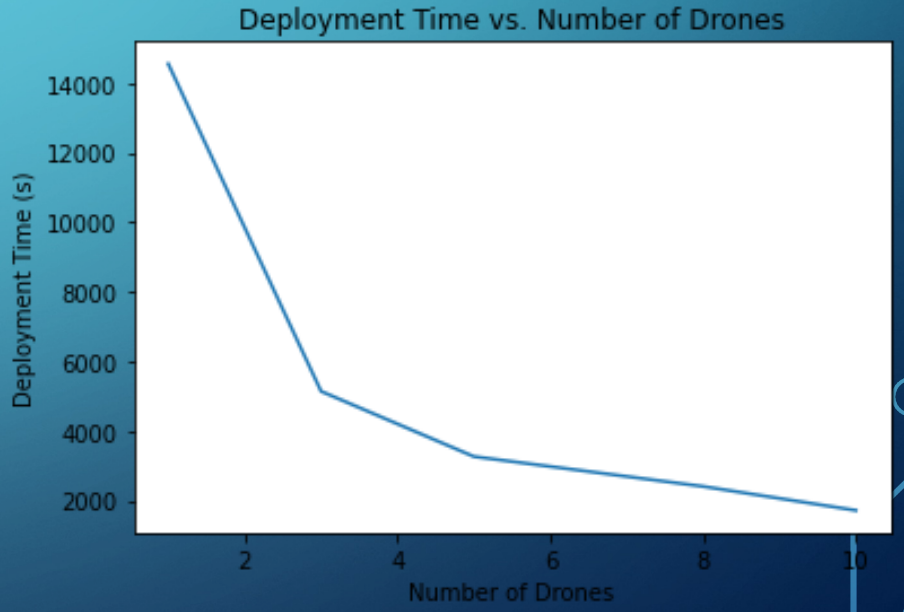
- partition_algorithm: In the case of a multi-drone delivery system, a partition of the plane is required. Since the drones eventually have to complete a cycle by coming back to the sink, we propose a polar-coordinate-based radial partitioning algorithm centered around the sink, such that each slice of the plane would have similar number of tasks. The algorithm begins with converting the coordinate of each minimum set edge into polar coordinates, stored in a dictionary with edges being the key, and the angles, which take values from 0 to 2pi, being the values. Then, the edges are sorted based on the angles, then partitioned into equal parts based on the number of sections we would like to partition. The end result is a list of dictionaries that contains the edges in each slice of the plane.

PART TWO: RESULTS

- The following figures from left to right are: 1. The one-drone trajectory without fuel constraint 2. A partition of drone trajectory under fuel constraints 3. Deployment time vs number of drones with fuel constraint







CONCLUSION

- By creating a state-of-the-art waterfall threshold algorithm in part one, a novel polar-coordinate-based radial partitioning and path construction algorithm in part two, and multiple graph-theory based algorithms, we were able to minimize the tradeoffs between coverage and connectivity, while effectively deploy the sensors. Unlike the allocation-specific path finding algorithms others have deploying, that may work for one distribution but fail in many other cases, our multifaceted algorithm was developed without prior assumptions on any specific distribution.

- For the source code and specifications, please refer to: https://drive.google.com/drive/folders/1IsX8hy8pg_IbI5P0FF_NvGSchHjmbz5R?usp=sharing