But it's a dry heat!

(Dated: February 2, 2018)

In this assignment we are exploring different types of search in the office agent HeatMiser. We explore breadth first search as our baseline uninformed search and then a heuristic based A* search. For our heuristic, we have straight line distance information from each office to every other office. The agent has to achieve a temperature average of 72°F ± 1.5 and average humidity of 45% ± 1.75, and has access to the temperature and humidity of each office, as well as the averages and standard deviations.

OptiHeatMiser

In order to code this agent, first we consider its performance measure, environment, actions, and sensors. The matrix bellow applies to both the breadth first search and A* scenario:

P (performance)	E (environment)	A (actuators)	S (Sensors)
Achieve 45% ± 1.75 & 72°F ± 1.5 in as few office visits and power used as possible	12 offices in a graph structure with known distances between each of the offices	Raise/decrease temperature by integer values as needed to achieve specified goal	Temperature and humidity of every office

This goal oriented agent can only raise or lower the temperature or the humidity of each office by whole numbers in order to achieve the desired averages and standard deviations.

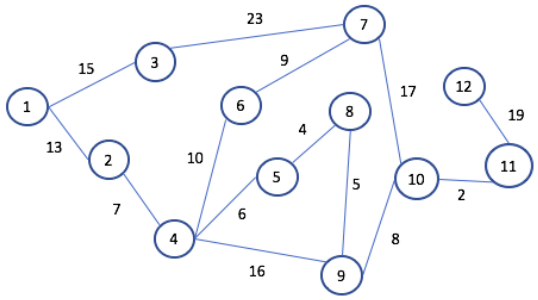

Figure 1. Map of the offices

HeatMiser finds the office that has the biggest difference from the preset goal for either temperature or humidity. We explore a graph structure that represents the office floor layout using Breadth First Search (BFS) and A*. Breadth first search expands each node in order based on the number of jumps from the starting node. In order to avoid any infinite loops we store a list of previously visited nodes. A* uses straight line distance from the current office to any desired office as a heuristic. It performs the following calculation:

$$distance = SLD + PC + ND,$$

where SLD is straight line distance to the goal office, PC is the current path cost, and ND is the distance to the next node. When the search begins, PC is zero since the agent has not yet moved. The agent uses the SLD heuristic and then looks for potential paths. It will then choose the one that has the smallest total sum. If the distance including PC is larger than other options, it will backtrack and take a different path until its cost becomes greater.

Implemented Breadth First Search Analysis

Our implemented BFS is complete, since it will always find a solution for the search problem, as in the worst case scenario, it will go through through all nodes. We know that the node we are looking for exists in the graph. It does not get stuck in infinite loops because we are keeping track of previously visited nodes. BFS is optimal in the sense that it gives us the route with the least number of nodes visited. However, in terms of power, it is not optimal, as edges weights are completely ignored. This algorithm has a time complexity of $O(b^d)$, for branching factor b and solution depth d. The space complexity is also $O(b^d)$.

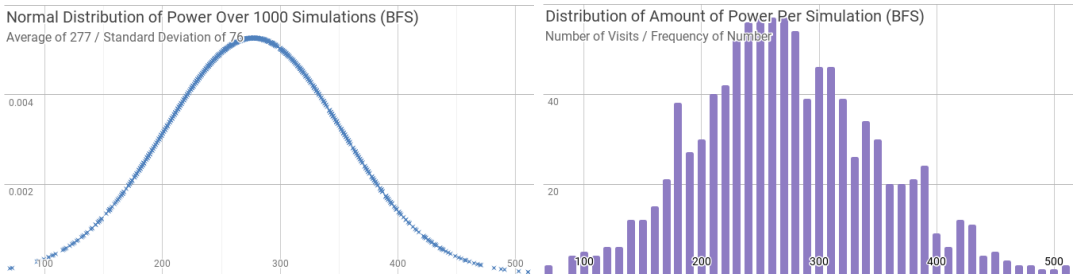

Figure 2. Power distribution in BFS: On the right, the y-axis represents the power consumption and the x-axis represents how many times out of 1000 simulations it took that much power; on the left there is a normal distribution of these results.

We found that on average, BFS took 26 visits with standard deviation of 6.5 and a power consumption of 277 with standard deviation of 76.

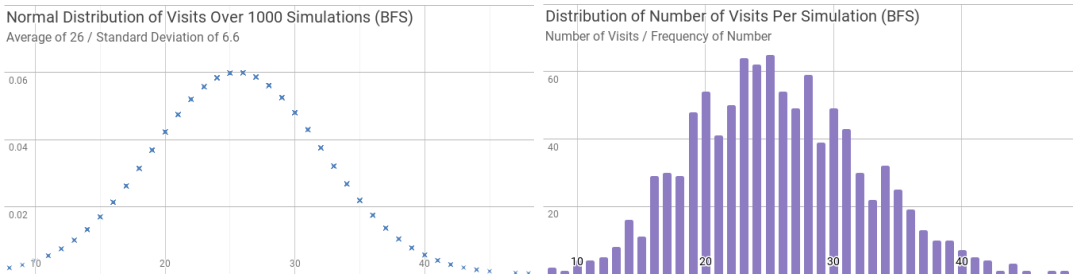

Figure 3. Visits distribution in BFS: On the right, the y-axis represents the number of visits and the x-axis represents how many times out of 1000 simulations it took that many visits; on the left there is a normal distribution of these results.

Implemented A* Analysis

Our implemented A* is complete as it will always find a solution for the search problem, since in the worst case scenario, it will go through through all nodes, as we know the node we are looking for exists within the graph. It does not get stuck in infinite loops because we are keeping track of previously visited nodes. It is not optimal because sometimes it picks a path that costs more than the best available (such as from office 4 to 9). However, it produces a path that is more efficient than an uninformed BFS. This algorithm has a time complexity of $O(b^d)$, for a branching factor b and solution depth d. The space complexity is also $O(b^d)$.

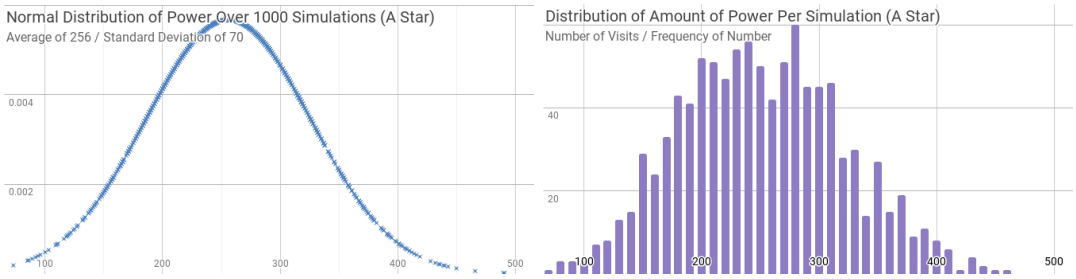

Figure 4. Power distribution in A*: On the right, the y-axis represents the power consumption and the x-axis represents how many times out of 1000 simulations it took that much power; on the left there is a normal distribution of these results.

We found that on average, A* took 27 visits with standard deviation of 7.5 and a power consumption of 256 with standard deviation of 70.

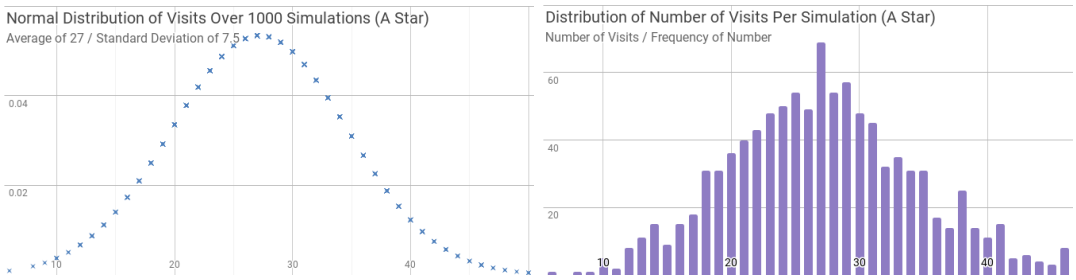

Figure 5. Visits distribution in A*: On the right, the y-axis represents the number of visits and the x-axis represents how many times out of 1000 simulations it took that many visits; on the left there is a normal distribution of these results.

Comparison

The graphs above show that BFS consistently visits one to two less offices than A*. However, A* consistently uses 20 less power. This makes sense, because the point of A* is that will pick a path with more nodes than BFS, but with the goal of minimizing path cost. We consider this to be small, but significant, particularly when it comes to scaling to a larger building. At the current scale, the differences are less obvious, as can be seen in the graphs below.

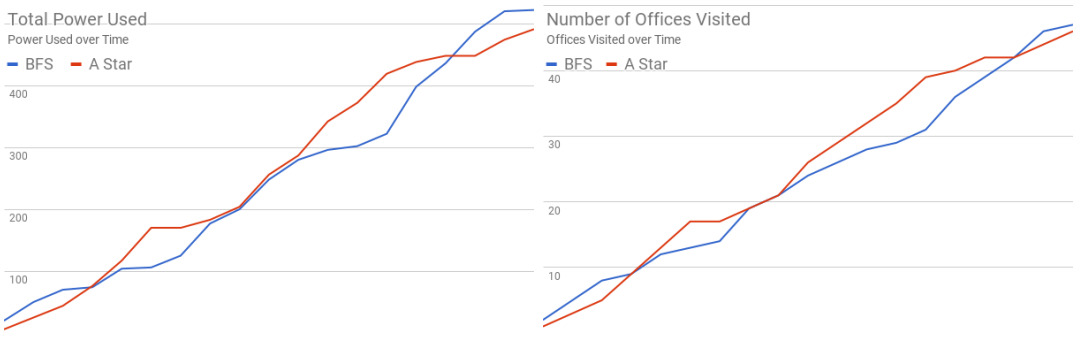

Figure 6. On the left: Total power comparison between BFS and A*, where the x-axis represents time and the y-axis power. On the right: Visits over time comparison between BFS and A*, where the x-axis represents time and the y-axis visits.

Improvements and future design

For a future design of HeatMiser, there are two possible changes we could make to its structure. First is to selectively remove certain edges (e.g. the edge between office 4 and 9). While there might physical exist a path there, other paths may exist that are cheaper than the original edge (e.g. 4-5-8-9 is cheaper than 4 to 9). We also believe a different heuristic could greatly improve the performance. For instance, the current heuristic says that office 12 is closer to most others than 11 is. However, 12 only has one access point – through 11. This is why we think the current heuristic should have information about accessibility. One could also imagine a heuristic based on existing precomputed efficient paths through the graph (like the equivalent of taking the freeway versus surface streets).

Furthermore, we would like to simultaneously change both the temperature and humidity of any office we pass through. Right now, we are only going to the office with the highest temperature or humidity difference and changing one of them, then finding the next biggest difference. Instead, we could change the humidity and temperature simultaneously of every office we visit along the path. Then, in the worst case, we would need to traverse every node once and are limited by the length of the Hamiltonian path of the graph (granted, that is still NP-complete).

Finally, if allowed to change the office values by fractions rather than just integers, we could more precisely adjust floor-wide averages quickly. A slightly modified PEAS matrix with that change is as follows (all the other proposed changes wouldn't affect this):

P (performance)	E (environment)	A (actuators)	S (Sensors)
Achieve $45\% \pm 1.75$ & $72°F \pm 1.5$ in as few office visits and power used as possible	12 offices in a graph structure with known distances between each of the offices	Raise/decrease temperature by *float* values as needed to achieve specified goal	Temperature and humidity of every office