Ryan Gahart

Final Project Report: A* Search Algorithm

For my final project, I decided to create a pathfinding application. In my application, it takes in a text file which has the letter ‘O’ for a tile that can be traversed, an ‘X’ for a tile that cannot be traversed, an ‘S’ for the starting point, and an ‘E’ for the ending point. The application then uses the A* search algorithm to run through the nodes and find the shortest path.

The algorithm solves many pathfinding problems. It can also work with weighted graphs. Possible applications include pathfinding for videogames AI, solving the traveling salesman problem, and finding the best route to take on GPS.

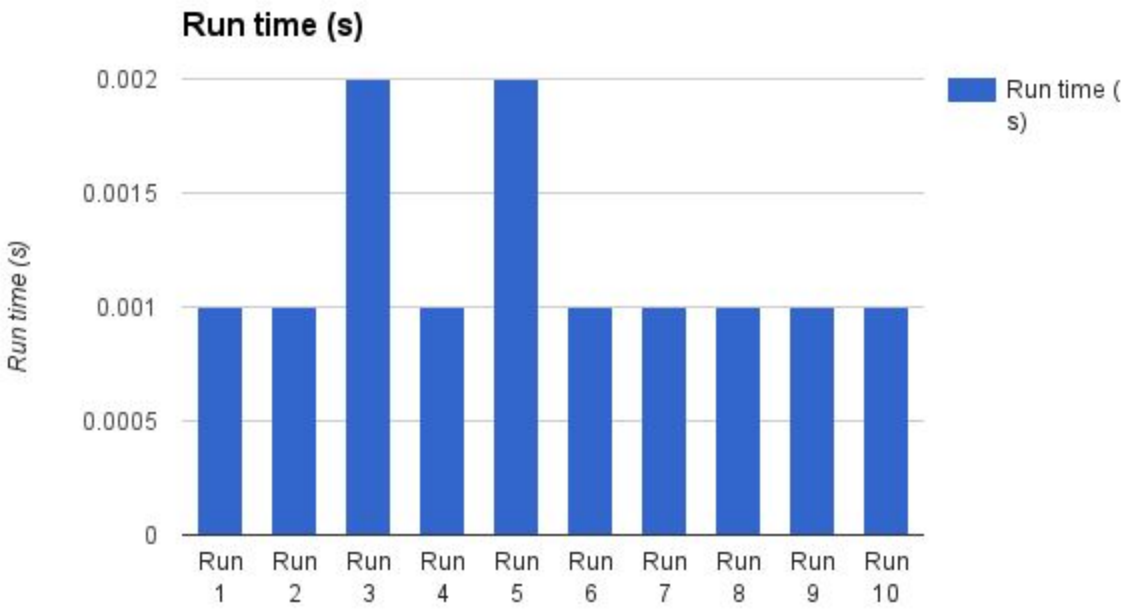
A* is similar to Dijkstra's algorithm and the Best-First Search algorithm. It is similar to Dijkstra’s in that in it will visit each node and record its weight up to that point and then choose the best route. It is also similar to a BFS in that it will expand the nodes that are most probable to reach the goal with the least cost or in this case least distance. A* combines the two so that it starts at the node that is most probable to reach the goal, and then expands to all its neighbors and gives them a weight to be used on the next loop to find which node to expand.

I ultimately chose to work with the A* algorithm because it was a compromise between Djikstra’s and BFS. It took the best of those two algorithms and combined them to create an algorithm that was just as accurate as Djikstra’s but with the speed of a BFS.

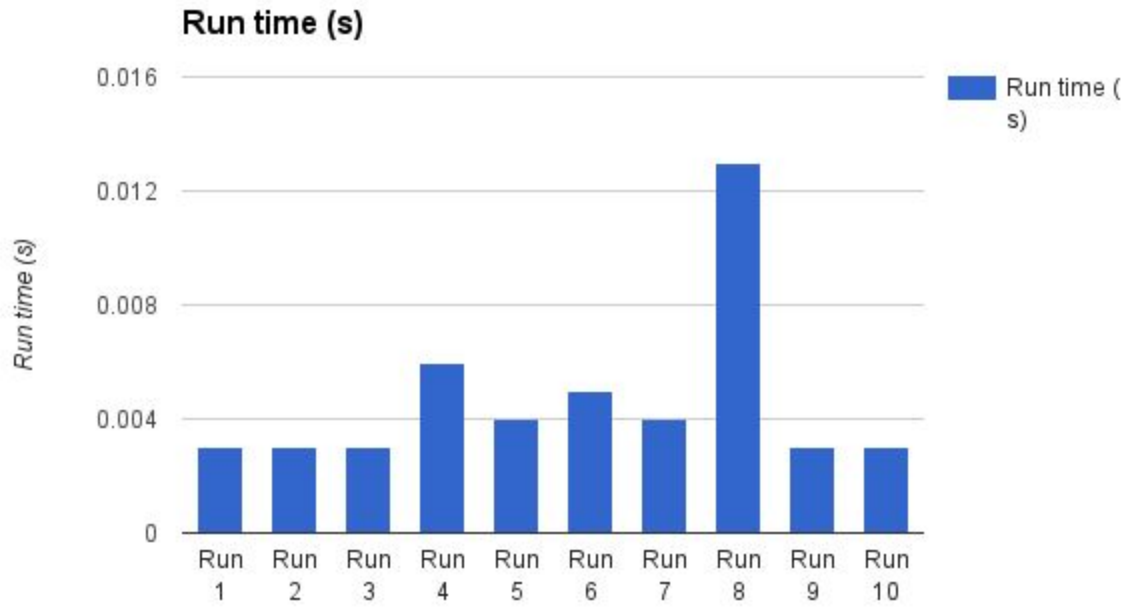
Validity: My implementation of the A* algorithm can be seen as complete in that given a start node and and end node with an actual path between the two, it will always find the shortest path. This is because it will look at every node that fits with the given heuristic. It is optimal because in my version of the application the grid must be square, and when finding the neighbors of each node, I only look at all eight, unless the node is on an edge or corner of the grid. It is also optimal in that the heuristic value is the distance from the current node to the end node, which can never be more than the current cost plus the heuristic. It will only look one node away from the current node, which is generally all that is necessary.

Times: The time complexity of the A* algorithm depends on what the heuristic value of h(n) is. It also depends on the branching factor of the algorithm. For my implementation it only looks at one node away from the current node, so has an effective branching factor of 1. So the theoretical worst case time is O(m*n).

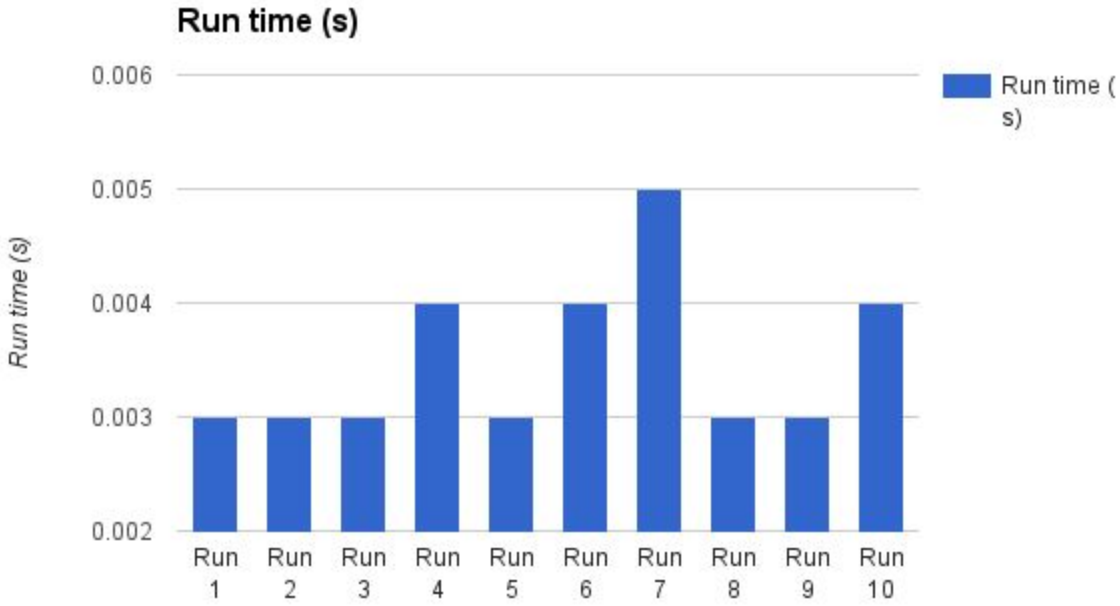
For test.txt:



For test2.txt



For test3.txt:



In test.txt, the grid was only 5x5 which can explain the smaller run time. In test2.txt and test3.txt, they are both 10x10, but with different blocking layouts. In test3.txt, I was expecting a longer runtime because I knew that more nodes would have to be traversed, but that didn't seem to matter. Both test2.txt and test3.txt had similar run times, because of the size of the grid.

References:

https://en.wikipedia.org/wiki/A*_search_algorithm
http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html
http://web.mit.edu/eranki/www/tutorials/search/