Project 1 Part 1

Joshua Westbrook & Arifur Rahman

1 The maps

The map files are available at `https://github.com/Joshua-Westbrook/AIproject1maps`

They are also in the folder that opens after unzipping the .zip file attached and importing it into Eclipse as an existing project.

2 The algorithm

The algorimths can be found in in the searches package. The UCS algorithm is fully implemented in SearchAlgo.java but uses UCS.java so it has a specific name. A* and Weighted A* overwrite the h(n) being zero in UCS with them using either the heuristic from DistanceHeuristic.java or the heuristic times the weight. UCS and A* always return the cheapest path while Weighted A* may or may not depending on the weight and the specific map with its start and goal point. In some situations such as map5j even a weight of 3.0 still has Weighted A* return the right path while other maps such as map5b return a longer path with a weight of 2.

3 Optimizations

One trade-off made in the algorithm is the use of a 2d array to store every node once it is added to the fringe. This allows for much quicker look up times of a node already in the fringe when checking if the new path found to that node is more optimal faster but also increases the memory requirements of the program because it will always require space to store all the nodes on the map. Due to the nodes not requiring much memory to store this is a good trade-off. In other situations the memory cost of allocating the 2d array might be worse than the time cost of iterating through the fringe, which is Java's priority queue, to retrieve the previously found hcost of reaching that node.

4 Heuristics

The Heuristic we finally used was a modified Euclidean distance . We then multiply the Euclidean distance between the currNode and goal by .25 since in a best case scenario the entire path to the goal is an highway/river with no hard to travel in which case

all movements is the distance divided by 4 which is the same thing as times 0.25 The formula for the heuristic is $0.25 * \sqrt{(currNode.x - end.x)^2 + (currNode.y - end.y)^2}$.

Other attempted heuristics were the following.

1. Euclidean distance from currNode to goal. Not admissable because it is not optimistic due to not accounting for rivers in its travel cost.

2. A Heuristic that attempts to avoid cells which are hard to travel through. This ended up being useful for some maps such as map1a and map1b where it did find the optimal path but it was not optimal on the majority of the maps.

3. Chebyshev distance from current node to the goal.

4. Manhattan Distance from current node to the goal.

5 benchmarks

Table 1: Results

Search method	Nodes	Time in MS	Cost
UCS	12274.83673	69.04081633	113.3290976
Modified Euclidean Distance A*	9533.163265	5.87755102	113.3290976
Modified Euclidean Distance 2 weight A*	6628.510204	13.3877551	113.9998702
Modified Euclidean Distance 3 weight A*	3700.897959	16.89795918	117.2203349
Euclidean Distance A*	1564.836735	6.795918367	122.383466
Euclidean Distance 2 weight A*	140.3265306	0.142857143	170.2938775
Euclidean Distance 3 weight A*	136.5714286	0.163265306	172.7860726
Hard Cells Avoidance A*	12177.65306	12.44897959	117.0232475
Hard Cells Avoidance 2 weight A*	11984.71429	25.69387755	118.5417895
Hard Cells Avoidance 3 weight A*	11779.40816	16.51020408	120.0317971
Chebyshev A*	9444.204082	9.530612245	120.310548
Chebyshev 2 weight A*	7708.816327	6.102040816	144.1297948
Chebyshev 3 weight A*	7261	6.714285714	153.1129255
Manhattan Distance A*	639.1632653	0.571428571	144.8362468
Manhattan Distance 2 weight A*	147.3673469	0.081632653	187.3616096
Manhattan Distance 3 weight A*	143.6734694	0.142857143	189.7389947

The excel sheet that was submitted with this contains the cost for eah map with each of its individual start and end points. Alternatively they can be obtained by ruinning our program on whichever map you wish to check.

6 Final Explanation

The results clearly show that when using an admissable heuristic while UCS and A* find the same path A* is able to do it with much less nodes expanding and a much quicker speed. Adding a weight increases the speed and decreases the nodes expanded

even more but also results in a slightly higher cost found on average. This makes it clear that using A* when you have an admissable heuristic is always better than using UCS and UCS should only be used in situations where an admissable heuristic isn't available. It also shows that weighted A* can be used to increase the speed of the algorithm and reduce the nodes expanded but that will also reduce the ability to get an optimal path. Many maps, such as 5a and 1j, see a pure increase in performance when using a weight of 2 which says that it is often worth using at least a small weight to increase your performance because the path found is very close to optimal at least on the maps we have. This is also shown by the average cost of A* with a weight of 2 only being 0.76 higher than the average optimal path from UCS/A* when using an admissable heuristic which is less than the cost of moving between two normal nodes. Meanwhile the average cost for A* with a weight of 3 is around 4 higher than the optimal path which makes it very clear that using larger weights will almost always cause a noticable increase in the cost of the found path. Although the inadmissable heuristics definitely did not come close to finding optimal paths there were clearly occassions where they would be useful. The unmodified Euclidean distance looked at under 150 nodes on average when weighted and was also noticable faster on the system which was used for benchmarking. This much faster and much less nodes look at is also shown by the Manhattan Distance which looks at much less nodes compared to UCS and an admissable A* even when not weighted. The shown results match what is expected. A* requires an admissable heuristic to a path of the same cost as UCS while always decreasing the number of nodes looked at compared to UCS. Adding a weight to A* always decreases the number of nodes looked at and the increase is bigger with bigger weights but this also icnreasese the cost of the found path. The one oddity is that the Manhattan Distance with a weight of 2 had an average time of roughly 0.08 milliseconds while the Manhattan Distance with a weight 3 had a time of 0.14 MS. However that is easily explainable by the machine using less resources for some runs compared to others and the difference being from how many runs took 0 milliseconds versus 1 millisecond according to Java. This suggests that it is likely due to Java's own rounding combined with not being able to be certain the amount of resources dedicated was the cause rather than any issue with the algorithm since using Manhattan distance and a weight of 3 did expand less nodes but also find a higher cost path compared to using a weight of 2.