AI Assignment 1

Gates Wang

October 2020

1 Optimizations

The algorithm follows what is given in the project description. There are several differences though. Firstly, the closed list is a hashtable. Hence, it takes O(1) time to determine membership in the closed list.

The open list is a heap and a hash table. We use the hash table to store (location, state) key-pair values. When we need to update the value at a particular location we simply replace it in the hashtable in O(1) time. We do not update the heap except when popping the minimum element. Note that a particular location may appear multiple times in the heap. However, that does not matter because if that location was expanded before then that location is part of the closed list. Therefore, that location will not be expanded multiple times even if it is popped from the heap multiple times because we only expand nodes that are not part of the closed list.

2 Metrics

The four metrics that are calculated are iterations, distance, time, and memory. Iterations is the number of nodes expanded. Distance is the cost of the path that is returned. It is returned as a ratio of the optimal path cost (except for sequential). Time is how long the program took to run. Memory is the maximum size of the heap throughout running the program.

3 Heuristics

The heuristics used were manhattan distance, euclidean distance, diagonal distance, and euclidean distance with straightening.

The admissable heuristic that was used was a quarter of the manhattan distance. This is because the rivers allowed for a quick path to the goal, and there is no faster way to get to the goal.

The heuristics that worked very well were straight and euclidean. Straight is the same as euclidean except it favors paths that go in the same direction as the vector that is formed from start to goal.

Diagonal takes into account moving diagonally, giving it a cost of $\sqrt{2}$ and giving vertical and horizontal movements a cost of 1. Diagonal had better performance than Manhattan because the agent can move diagonally in addition to moving horizontally and vertically.

Figure 1: Heuristics: Mean (top) + Median (bottom)

	iterations	distance	time	memory
heuristic				
admissable	4847.392	1.191518	0.131401	589.128000
diagonal	503.804	3.060307	0.016377	341.272000
straight	440.884	3.271374	0.015611	312.576000
manhattan	2152.590	3.396669	0.059595	378.063333
euclidean	457.360	3.604910	0.014835	321.104000

	iterations	distance	time	memory
heuristic				
admissable	4301.0	1.000000	0.114355	581.0
straight	265.0	1.667685	0.010338	305.0
euclidean	279.0	1.677736	0.010039	314.0
diagonal	341.0	1.731197	0.011003	332.5
manhattan	375.5	1.745402	0.012482	334.0

4 Weighted A^* search

Having a weight of 0 is uniform cost search. Having a weight of 1 is unweighted A^* search. Note that having a weight of one did not guarantee an optimal solution because not all heuristics used were admissible.

The results of the different weights used are shown in Figure 1. It was found that weights greater than 1.5 were finding paths that were too far off from the optimal. So we tested numbers from 1 to 1.5. We have both the mean and the median shown below. The mean is shown first but it is skewed because of tail skewness.

Increasing weight causes iterations, memory, and time to decrease. However, distance increases, leading to a less optimal solution. The best weight to use will depend on the constraints of the problem.

5 Sequential A^* search

It is not entirely clear why but sequential search had significantly more iterations (nodes expanded) than weighted A^* search. It is possible that there are naturally more iterations since multiple searches are being conducted.

The results of sequential search are shown in Figure 2. In general it was found that increasing weight1 for sequential A^* had the same effect as weighted

Figure 2: Weighted search: Mean (top) + Median (bottom)

weight	iterations	distance	time	memory
0.000	10724.600	1.000000	0.315778	679.940
1.000	1877.256	1.772755	0.057640	423.804
1.125	1545.044	2.755551	0.047457	396.808
1.250	1235.764	3.018788	0.038368	369.056
1.375	1089.652	3.455602	0.034046	354.832
1.500	939.912	4.001416	0.028486	337.268

weight	iterations	distance	time	memory
0.000	12289.0	1.000000	0.320819	698.0
1.000	878.0	1.261326	0.026332	401.0
1.125	620.0	1.464294	0.019881	365.5
1.250	348.5	1.651675	0.014006	332.5
1.375	257.5	1.765442	0.010109	320.0
1.500	197.0	1.908241	0.007294	303.5

A^* search. Increasing weight2 for sequential A^* search had the same effect, that is iterations, time, and memory decreased. However, in order for weight2 to have an effect weight2 must be greater than weight1.

Generally speaking having two weights allows for better precision in terms of balancing run time and optimality.

6 Problem

6.1 Notation

Let C(u,v) denote the cost of the optimal path from u to v.
Let h_i denote the $i - th$ heuristic. h_0 is an admissible heuristic
Let Q_i denote the priority queue for h_i.
Let $g_i(s)$, $h_i(s)$, and $f_i(s)$ denote the f, g, and h values of a particular state.

6.2 Part 1

There is always a node that is on the optimal path in Q_0. The first node that is in Q_0 is the start node, which is on the optimal path. Whenever the optimal node is expanded, the next node on the optimal path is added to the open list because all of it's neighbors that are not in the closed list are added. The next node on the optimal path cannot be in the closed list because we used an admissible heuristic. If it were in the closed list, then there would be another

Figure 3: Sequential search: Mean (top) + Median (bottom)

weight	weight2	iterations	distance	time	memory
1.5	1.5	3956.14	22.731184	0.104911	562.04
	2.0	3842.54	22.788463	0.106346	556.22
1.0	1.5	6043.00	23.059735	0.169174	620.90
2.0	1.5	3056.82	25.940800	0.082816	535.64
	2.0	3056.82	25.940800	0.083222	535.64
1.0	2.0	4805.22	27.415224	0.160952	547.38
	2.5	3472.04	28.314792	0.137500	485.18
1.5	2.5	2846.32	31.692550	0.083294	479.90
2.0	2.5	2382.86	39.323753	0.071713	477.70

weight	weight2	iterations	distance	time	memory
1.0	1.5	5076.0	21.480518	0.141316	625.5
1.5	1.5	3716.5	22.561260	0.103637	557.5
	2.0	3636.5	22.561260	0.105638	551.5
2.0	1.5	3038.5	22.914943	0.081785	527.0
	2.0	3038.5	22.914943	0.079992	527.0
1.0	2.0	3375.5	23.607896	0.137258	579.0
	2.5	2178.0	23.607896	0.102444	427.0
1.5	2.5	1839.0	24.378927	0.058757	510.5
2.0	2.5	2262.0	29.339950	0.061725	490.0

path, that is shorter than our optimal path. Therefore there is always a node in Q_0 that is on the optimal path

6.3 Part 2

We assume that for all u in Q_0

$$f_0(s) \leq f_0(u) = g_0(u) + w_1 * h_0(u) \tag{1}$$

We are given that (1) implies

$$g_0(u) \leq w_1 * \text{C(start, u)} \tag{2}$$

Since h_0 is admissible

$$h_0(u) \leq \text{C(u, goal)} \tag{3}$$

Substituting (2) and (3) into (1) yields.

$$f_0(s) \leq w_1 * \text{C(start, u)} + w_1 * \text{C(u, goal)} \tag{4}$$

$$= w_1 * \text{C(start, goal)} \tag{5}$$

Note that the equality is true because we know that there is a u that is on the optimal path.

6.4 Part 3

There are two ways for termination.
Case 1:

$$f_i(s) \leq w_2 * f_0(s) \text{ and } g_i(s) \leq f_i(s) \tag{6}$$

We combine the two inequalities to derive

$$g_i(s) \leq f_i(s) \leq w_2 * f_0(s) \leq w_2 * w_1 * \text{ C(start, goal)} \tag{7}$$

Case 2:

$$f_i(s) > w_2 * f_0(s) \text{ and } g_0(s) \leq f_0(s) \tag{8}$$

Using only the second inequality yields

$$g_0(s) \leq w_1 * \text{C(start, goal)} \leq w_2 * w_1 * \text{C(start, goal)} \tag{9}$$