15-887: Assignment #2

Due on Wednesday, October 19, 2016

Jordan Ford

Contents

Problem 1

(a)

To run this planner, please use the following command: **python world.py -a prob1.txt**

This planner performs a forward djikstra search in x and y and uses the result as an informed heuristic for a single backward A* search in x, y, and t.

The 2D djikstra search expands all nodes in the graph - one million nodes for test case one. On my machine, it takes about 8 seconds to complete this phase of the search. The distances calculated by the djikstra search are then used as the heuristic for a backward A* search which searches in both position and time. The search prunes nodes for which the number of timesteps required to reach the node is less than the manhattan distance from the start to that node.

Test Case	Time [sec]	Path Cost	States Expanded
0	0.000301	7	20
1	25.700	81915	1,632,110

Table 1: Optimal A* Planner Performance Metrics

(b)

To run this planner, please use the following command: **python world.py -b prob1.txt**

One of the advantages of the algorithm I chose for part (a) is its ease of extension to part (b). By weighting the backward A* search, it is easy to trade optimality for speed, and it comes with the typical weighted A* performance guarantees. For the results presented in Table 2, the A* heuristic was weighted by a factor of 100, resulting in a path that is 15% higher cost but which required only 34% of the time to compute.

Test Case	Time [sec]	Path Cost	States Expanded
0	0.000311	7	22
1	8.788	94947	1,003,175

Table 2: 100x Weighted A* Suboptimal Planner Performance Metrics

The weighting reduces the time required for the A* phase of the search from about 17 seconds to around 0.7 seconds, but does nothing to speed up the djikstra phase, which still requires 8 seconds. In order to further speed up the search, it would likely be useful to use A* to generate the heuristic instead of djikstra. The g-values from a forward 2-D A* could be used to augment a manhattan distance heuristic for use in the 3-D A* search. From experience with this problem, I estimate this technique would reduce the number of states expanded by the heuristic generating search in test case one by half, bringing the total number of states expanded to around 500k.

Problem 2

2.1

Suppose you have two consistent heuristic functions: h_1 and h_2. Prove that h(s) = max($h_1(s)$, $h_2(s)$) for all states s in the graph is also a consistent heuristic function.

A heuristic is consistent if

$$h(n) \leq c(n, n') + h(n')$$

for every node n and its child node n'.

Proof:

$$\begin{aligned} h(n) &= max(h_1(n), h_2(n)) \\ &\leq max(c(n, n') + h_1(n'), c(n, n') + h_2(n')) \\ &\leq c(n, n') + max(h_1(n'), h_2(n')) \\ &\leq c(n, n') + h(n') \end{aligned}$$

Suppose you have two consistent heuristic functions: h_1 and h_2. Prove that h(s) = min($h_1(s)$, $h_2(s)$) for all states s in the graph is also a consistent heuristic function.

A heuristic is consistent if

$$h(n) \leq c(n, n') + h(n')$$

for every node n and its child node n'.

Proof:

$$\begin{aligned} h(n) &= min(h_1(n), h_2(n)) \\ &\leq min(c(n, n') + h_1(n'), c(n, n') + h_2(n')) \\ &\leq c(n, n') + min(h_1(n'), h_2(n')) \\ &\leq c(n, n') + h(n') \end{aligned}$$

2.2

d. Monotonically non-increasing sequence

2.3

f. None of the above

2.4

e. None of the above