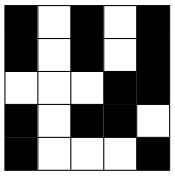
CS 520 Introduction to Artificial Intelligence
Homework 1

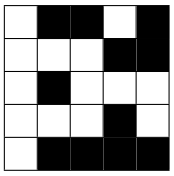
Malike Alikhani (RUID: 167001652)
Ana Echavarria Uribe (RUID: 167007648)

October 12, 2015

Part 0

We designed and algorithm to create a maze like structure based on the algorithm presented in (`\http://www.migapro.com/depth-first-search/`) but with certain modifications. The idea is that when running a depth first search from a given cell select randomly one of its unseen neighbors that are two cells away in any direction (north, south, east or west). Move to that cell but blocking all of the unseen neighbors of the intermediate cell (the cell in between the current cell and the selected neighbor). This creates a maze like structure with no loops and one connected component.
Now, in order to add loops and creating more than one connected component we simply set random blocked cells as unblocked and random unblocked cells as blocked. Examples of 5×5 grids generated using this algorithm are:





Part 1

a. Since A* follows a path that has the lowest expected total cost, it will move to East rather than North to minimize the total expected cost.

$$g(\text{East}) = 1 \text{ and } h(\text{East}) = 2 \rightarrow f(\text{East}) = 1 + 2 = 3$$

$$g(\text{North}) = 1 \text{ and } h(\text{North}) = 4 \rightarrow f(\text{North}) = 1 + 4 = 5$$

b. The agent will not get stuck in an infinite loop because we do not visit a cell that has already been expanded in that iteration which means that every unblocked cell is expanded at most once. In this algorithm, we move that particular state to the closed list.
Whenever the path traversed by the agent is blocked, we recompute a new path. This makes sure we cover all the unblocked cells in the grid that are reachable by the agent. Unless there are infinite number of cells, the agent (A* algorithm) would find the goal state or would discover it is impossible to reach the goal because the there is no new path that gets the agent to the goal.

WE ARE MISSING THE BOUND PROOF

Part 2

The repeated forward A* was executed on 50 grid of size 101×101 using two different rules for selecting the next cell to expand when their $f = g + h$ value is the same: selecting the cell with the smaller g value or with

	Num cells expanded	Num searches	Num moves	Running time (sec)
Smaller g	66,695	97	385	0.1608
Larger g	8,164	95	376	0.0939
Ratio Smaller/Larger	8.1699	1.0185	1.0240	1.7124

Table 1: Average statistics when running on 50 grids and breaking ties in favor of larger and smaller values of g.

the larger value of g. The average number of cells expanded, searches (repetitions of the A* search), number of cells the agent moves and running time are shown in results in table 1.
It can be seen that, on average, the number of cells expanded when selecting the cell with the smaller value of g is more than 8 times higher than the number of cells expanded when selecting the cells with the larger value of g. The running time, nonetheless is not 8 times higher; this could be because the time overhead of starting a search is so big that the ratio in the running time is only 1.7 times higher even if the search is doing 8 times the work. Additionally, we can see that in spite of the fact that the number of cells expanded is much larger, the algorithm still does around the same number of searches and the same number of moves for the agent; this would indicate that the number of cells expanded is larger for the tie breaking in favor of cells with smaller g values because there are more cells being explored in every search rather than because there are more searches being done.
This difference in the number of cells expanded could be due to the fact than selecting a cell with a smaller g rather than a cell with a larger g implies that we are favoring the cell with the largest h value. Since the h value is a lower bound on the number of steps from that cell to the target, the cell that is being expanded is possibly farther away from the target than the other cell is. This could in the long run make us explore all the neighbors of that cell which are farther away and are not getting us to closer to the target.

Part 3

The algorithms for the forward and backward A* were executed on the same 50 grids of size 101×101 and using the same initial cell for the agent and for the target cell. The results in terms of the average number of expanded cells, number of searches, number of moves of the agent and running time, along with the ratio comparison of these values are shown in table 2.

	Num cells expanded	Num searches	Num moves	Running time (sec)
Backward A*	455,896	96	373	0.7797
Forward A*	8,164	95	376	0.0879
Ratio Backward/Forward	55.845	1.0103	0.9923	8.8687

Table 2: Average statistics when running on 50 grids and running backward and forward A*.

Part 4

Consider the coordinates of goal state G be (G_x, G_y). Let cell A be (x_1, y_1). The estimated path cost from cell A to goal G (A-G) is

$$|G_x - x_1| + |G_y - y_1|.$$

Now assume the agent travels to the goal through a cell $B(x_2, y_2)$ which lies outside the path (A-G). The new path cost A-B-G is

$$|G_x - x_2| + |Gy - y2| + |x_2 - x_1| + |y_2 - y_1|$$

Compare the two path costs, A-G and A-B-G.

$$|G_x - x_2| + |x_2 - x_1| \rightarrow |G_x - x_1|$$

(This is always true for any 3 values)

$$|G_y - y_2| + |y_2 - y_1| \rightarrow |G_y - y_1|$$

(This is also always true for any 3 values) Combining the two inequalities prove that estimated path cost

$$(A - B - G) \rightarrow (A - G).$$

So there can't be a shorter path to the goal through B. Therefore the initial path (A-G) estimated by the heuristic is the shortest path. Therefore Manhattan distances are consistent. From the proof, we have

$$(B - G) + (A - B) = (A - B - G) \rightarrow (A - G)$$

by triangle inequality.

Part 5

	Num cells expanded	Num searches	Num moves	Running time (sec)
Adaptive A*	6,062	96	378	0.0886
Repeated A*	8,164	95	376	0.0893
Ration Adaptive A*/Repeated A*	0.7426	1.0078	1.0068	0.9923

Table 3: Average statistics when running on 50 grids and running Repeated A* and adaptive A*.

Part 6

We suggest the following methods for decreasing the memory usage of our implementation.
We can use a method when creating the states which does not create all the states initially. We only create a state if it's going to be explored by the algorithm.
Another possibility is prevent storing f-value and h-values, which can both be calculated easily. The g-values can also be calculated by following tree-parents until reaching current start state. However, since this will increase the computation costs considerably, we do not include this improvement in our calculation.
Another improvement could be instead of storing the counter as search variable, a boolean variable can be used to check if the state is visited in the current iteration.