CS 520: Introduction to Artificial Intelligence Programming Assignment 1

Akshay Arjun Bhatia (174000727)
Pavan Nihal Renduchintala (172002406)

1 Environment Setup

We have created a virtual grid world with 120 rows and 160 columns. This environment is similar to a real time computer game and different cells in the grid have different costs of traversal attached to them. A virtual agent is made to operate in this environment The agent is provided with a start position and an end position and its task is to find the optimum path for traversal using variations of the A star algorithm.

The cost of transitioning between two regular unblocked cells is 1 if the agent moves horizontally or vertically and 2 if the agent moves diagonally. Then, we decide the placement of harder to traverse cells. To do so, we select eight coordinates randomly (xrand, yrand). For each coordinate pair (xrand, yrand), we consider the 31x31 region centered at this coordinate pair. For every cell inside this region, we choose with probability 50% to mark it as a hard to traverse cell. The cost of transitioning into such hard to traverse cells is double the cost of moving over regular unblocked cells, i.e.,

- **moving horizontally or vertically between two hard to traverse cells has a cost of 2;**
- **moving diagonally between two hard to traverse cells has a cost of sqrt(8);**
- **moving horizontally or vertically between a regular unblocked cell and a hard to traverse cell (in either direction) has a cost of 1.5;**
- **moving diagonally between a regular unblocked cell and a hard to traverse cell (in either direction) has a cost of (sqrt(2)+ sqrt(8))/ 2 ;**

The next step is to select four paths on the map that allow the agent to move faster along them (i.e., the rivers or highways). We allow only 4-way connectivity for these paths, i.e., these highways are allowed to propagate only horizontally or vertically. For each one of these paths, we start with a random cell at the boundary of the grid world. Then, move in a random horizontal or vertical direction for 20 cells but away from the boundary and mark this sequence of cells as containing a highway. To continue, with 60% probability we select to move in the same direction and 20% probability we select to move in a perpendicular direction (turn the highway left or turn the highway right). Again we mark 20 cells as a highway along the selected direction. If we hit a cell that is already a highway in this process, we reject the path and start the process again. We continue marking cells in this manner, until we hit the boundary again. In terms of defining costs, if we are starting from a cell that contains a highway and we are moving horizontally or vertically into a cell that also contains a highway, the cost of this motion is four times less than it would be otherwise (i.e., 0.25 if both cells are regular, 0.5 if both cells are hard to traverse and 0.325 if we are moving between a regular unblocked cell and a hard to traverse cell). The set of blocked cells over the entire map are selected

randomly as 20% of the total number of cells (i.e., 3,840 cells) on the map. Agents are not allowed to move into these blocked cells. Paths cannot pass through blocked cells or move between two adjacent blocked cells but can pass along the border of blocked and unblocked cells. For simplicity, we assume that paths can also pass through vertices where two blocked cells diagonally touch one another.

Finally, we place the start vertex sstart and the goal vertex sgoal. We select the start vertex randomly among unblocked cells (standard or hard to traverse) on the grid in one of the following regions:

- top 20 rows or bottom 20 rows
- left-most 20 columns or right-most 20 columns

Similarly, we select the goal vertex randomly among unblocked cells in the above regions. If the distance between the start and the goal is less than 100, then we select a new goal. We assume that the grid is surrounded by blocked cells. We also assume a static environment (i.e., blocked cells remain blocked, unblocked cells remain unblocked, etc.). The agents are occupying and transitioning between the centers of cells. The objective of path planning is to find as fast as possible a short path from sstart to sgoal in terms of accumulated cost. We will be searching unidirectionally from the start vertex to the goal vertex.

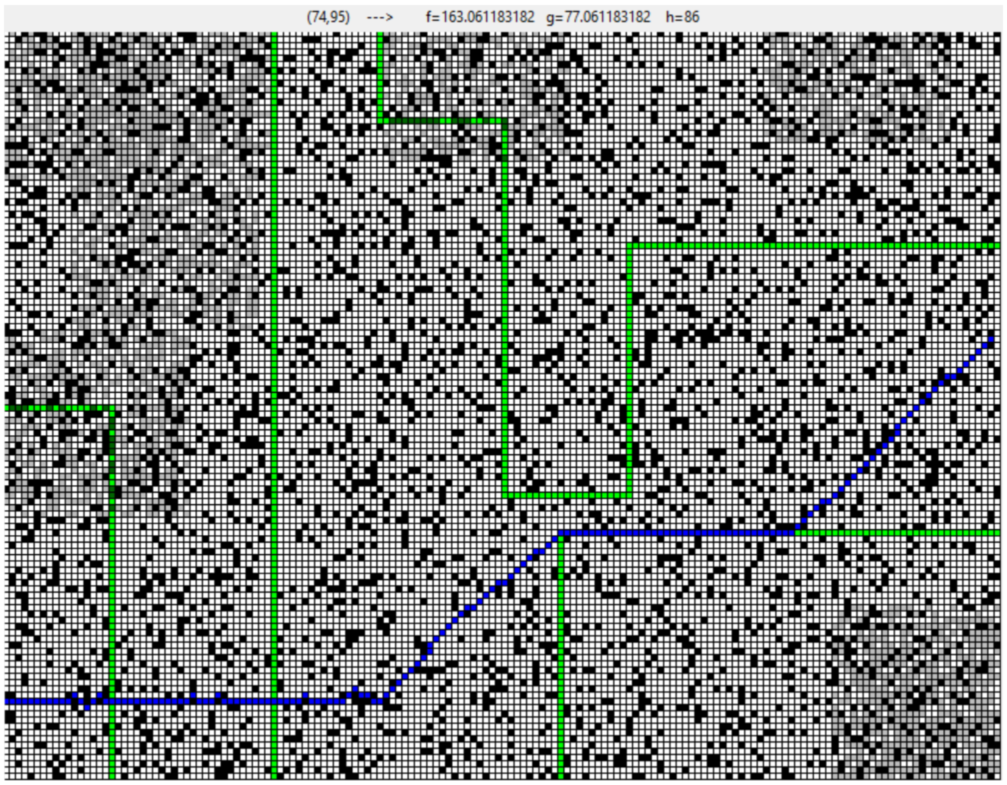

Figure 1: Sample of Visualization Software.

2 Visualization:

We display the different types of cells in the grid using different colors. The color code used is as follows:

- **White for normal cells**
- **Grey for hard to traverse cells**
- **Green for highways/rivers**
- **Dark green for hard to traverse cells on a highway**
- **Black for blocked cells**
- **Blue for cells in the generated path**

The interface is clickable and when a cell is clicked, it provides the co-ordinates of the cell along with its f, g and h values.

3 Technologies Used

Implementation: Python 2.7
Visualization: Pythons Tkinter module

4 Experimental Evaluation

We performed an experimental evaluation on three algorithms viz. Uniform Cost Search, A star algorithm and Weighted A star algorithm. In order to perform these evaluations, we have considered 5 different grid environments with 10 different start and end nodes for each. We compare the solutions provided by these algorithms in terms of time for execution, path cost, length of the path and number of nodes expanded. The heuristics considered are Chebyshev Distance, Manhattan Distance, Euclidean Distance, Canberra Distance and (Manhattan/4) Distance.

Table 1: Average Time for execution(seconds)

Algorithm	**Chebyshev Distance**	**Manhattan Distance**	**Euclidean Distance**	**Canberra Distance**	**Manhattan/4 (self made admissible heuristic)**
Uniform cost Search	162.97	162.97	162.97	162.97	162.97
A star	64.19	19.16	43.67	164.08	172.95
Weighted A star (w=1.5)	10.04	4.24	4.95	162.057	143.24

Table 2: Average Path Cost

Algorithm	**Chebyshev Distance**	**Manhattan Distance**	**Euclidean Distance**	**Canberra Distance**	**Manhattan/4 (self made admissible heuristic)**
Uniform cost Search	163.03	163.03	163.03	163.03	163.03
A star	165.0	173.49	165.01	163.03	167.17
Weighted A star (w=1.5)	188.95	176.81	167.49	163.03	167.33

Table 3: Average Path Length

Algorithm	**Chebyshev Distance**	**Manhattan Distance**	**Euclidean Distance**	**Canberra Distance**	**Manhattan/4 (self made admissible heuristic)**
Uniform cost Search	160	160	160	160	160
A star	154	160	154	160	154
Weighted A star (w=1.5)	157	160	154	160	155

Table 4: Number of nodes Expanded

Algorithm	**Chebyshev Distance**	**Manhattan Distance**	**Euclidean Distance**	**Canberra Distance**	**Manhattan/4 (self made admissible heuristic)**
Uniform cost Search	15743	15743	15743	15743	15743
A star	4988	1001	3168	15743	15395
Weighted A star (w=1.5)	490	168	158	15743	9538

5 Deductions

5.1 Uniform Cost Search:

The uniform cost search returns an optimal cost but this algorithm is not feasible because the execution time is very high and the number of nodes expanded is way greater than the number of nodes in the path.

5.2 A star Algorithm:

As compared to uniform cost search, the execution time of this algorithm is less as the number of nodes expanded are less.

5.3 Weighted A star:

This algorithm is faster as compared to the previous two algorithms as it heavily relies on the heuristic function and the number of nodes expanded are lesser.

5.4 Heuristics:

1)Chebyshev Distance
2)Manhattan Distance
3)Euclidean Distance
4)Canberra Distance
5)Manhattan Distance/4

5.5 Admissible/Consistent Heuristic:

We have chosen the Manhattan Distance/4 (heuristic 5) as the admissible/consistent heuristic because: The minimum cost is obtained by traversing on the highways and the highway traversal reduces the cost by a factor of 4. Hence a Manhattan distance divided by a factor of 4 never overestimates the actual cost to the goal.

6 Phase 2:

6.1 Solution for (g):

The observed statistics for the two algorithms are as follows:

6.1.1 Sequential A star

Average Running Time : 96.728 secs
Average Path Cost: 198.5929
Average Path Length: 175
Average number of nodes expanded: 11526

6.1.2 Integrated A star

Average Running Time : 93.728 secs
Average Path Cost: 197.63
Average Path Length: 170
Average number of nodes expanded: 9521

6.2 Solution for (h)

The algorithms are efficient because they consider a combination of admissible and inadmissible heuristics. This helps in optimizing both the path cost and the distance to the goal.In sequential each state can expand up to n+1 times,where as the in Integrated each state can expand at most twice.Hence the running time and the number of nodes of latter is lesser than the former. The path cost of Integrated is in the same bounds as sequential.

6.3 Solution for (i)

We prove by contradiction:
Let us assume $key(s,0) = g_0(s) + h_0(s) > w_1 * g * (s_{goal})$
Let us consider a least cost path from s_start **to** s_goal **as** P. **From this path, we pick the first state** s_i **that has not yet been expanded by the anchor search, but is a part of** $OPEN_0$
$g_0(s_i) \leq g_0(s_{i-1}) + c(s_{i-1}, s_i)$
$\leq w_1 * g * (s_{i-1}) + c(s_{i-1}, s_i)$
$\leq w_1 * (g * (s_{i-1}) + c(s_{i-1}, s_i))$
As $s_{i-1}, s_i\ \in$ **optimal path,**
$= w_1 * g * (s_i)$
Thus,we have $g_0(s_i) \leq\ w_1 * g * (s_i)$. **Using this we obtain,**
$key(s_i, 0)$
$= g_0(si) + w1 * h_0(si)$
$\leq w_1 * g * (s_i) + w_1 * h_0(s_i)$
$\leq w_1 * g * (s_i) + w_1 * c * (s_i, s_{goal})$

h_0 **is consistent, thus admissible**
$= w_1 * g * (s_{goal})$
Now , as $s_i \in \ OPEN_0$ **and** $key(s_i, 0) \leq w_1 * g * (s_{goal}) < key(s, 0)$**, we have a contradiction that** $key(s, 0) \leq \ key(u, 0) \forall u \ \in \ OPEN_0$

If anchor search terminates at line 32, we have
$key(s_{goal}, 0) \leq key(u, 0), \forall u \in OPEN_0$
It is given that $g_0(s) \leq w_1 * c^*(s)$
Therefore,
$g_0(s_{goal}) \leq w_1 * g^*(s_goal)$
$\leq w_1 * w_2 * g^*(s_goal)$ **As** $w_2 \geq 1.0$
Similarly, if an inadmissible search terminates at line 24, we have
$g_i(s_goal) \leq w_2 * OPEN_0.Minkey()$
$\leq w_2 * w_1 * g^*(S_goal)$
From the earlier proof .

6.3.1 Solution for (j)

A state can only be expanded if it is selected as the top state of $OPEN_i$**. After being expanded, the next call in the algorithm is to Expand() which removes the state from** $OPEN_i$**(line34) The algorithm checks if a state has already been expanded in the inadmissible searches and if so, it is not inserted again in** $OPEN_i$**.(line 14) If a state is expanded in the anchor search, again it is removed from all** $OPEN_i$**(line 4).Hence, the state can only be expanded again in either inadmissible searches or in anchor search if it is reinserted in any of the** $OPEN_i$**(lines 13,17). But, the control will never reach those lines as the check at line 12 will never be true. Therefore (ii) is true. A state s can be expanded at most once in the inadmissible searches and at most once in the anchor search.Thus, it cannot be expanded more than twice. Hence (i) is true.A state s that has been expanded in an inadmissible search can only be expanded again if the state is reinserted in** $OPEN_0$**. A state can only be inserted in any** $OPEN_i$ **if its g value is less than its earlier g value. Thus, a state whose g value has not been lowered cannot be reinserted ever. Hence (iii) is true.**