Kshitij Shah - ks1223
Sai Susmitha Batchu – sb1543

CS 520: INTRODUCTION TO ARTIFICIAL INTELLIGENCE

Assignment 1

Heuristic Search using Information from Many Heuristics

A. CREATING AN INTERFACE

We've created an interface using JavaFX that enables us to create, read, visualize and solve grid maps of dimension 160 columns and 120 rows. Every map contains different terrain types, namely, blocked cells, regular unblocked cells, hard to traverse cells, and highways with unblocked and hard to traverse cells. First, we initialize all the cells in the map to regular unblocked cells, then we randomly select eight coordinates and consider the 31x31 region centered around these coordinates in which every cell is hard to traverse with the probability 50%. In the next step, we select four non-intersecting paths that allow the agent to move faster along them (i.e., the "rivers" or highways). Then, we randomly select 20% of the total number of cells (which are not on highways) to mark as blocked. Finally, we randomly select the start and goal nodes. Maps will be stored in text files which contain the following information:

- The first line provides the coordinates of start node.
- The second line provides the coordinates of goal node.
- The next eight lines provide the coordinates of the centers of the hard to traverse regions.
- Then, we have 120 rows with 160 characters each that indicate the type of terrain:
  - '0' indicates a blocked cell
  - '1' indicates a regular unblocked cell
  - '2' indicates a hard to traverse cell
  - 'a_x' indicates a regular unblocked cell on highway x
  - 'b_x' indicates a hard to traverse cell on highway x

Figure 1 shows a text file containing a sample map.

We created five different maps with ten different start-goal pairs in each. After creating the maps, we run family of A* algorithms (uniform-cost, A* and weighted A*) to compute a path from the start node to goal node. These maps are visualized by using a color coding system to represent the various terrains:

- The start cell is represented by an aqua square.
- The goal cell is represented by a green square.
- Regular cells are represented by white squares.
- Blocked cells are represented by black squares.
- Hard to traverse cells are represented by grey squares.
- Regular cells on highway are represented by blue squares.

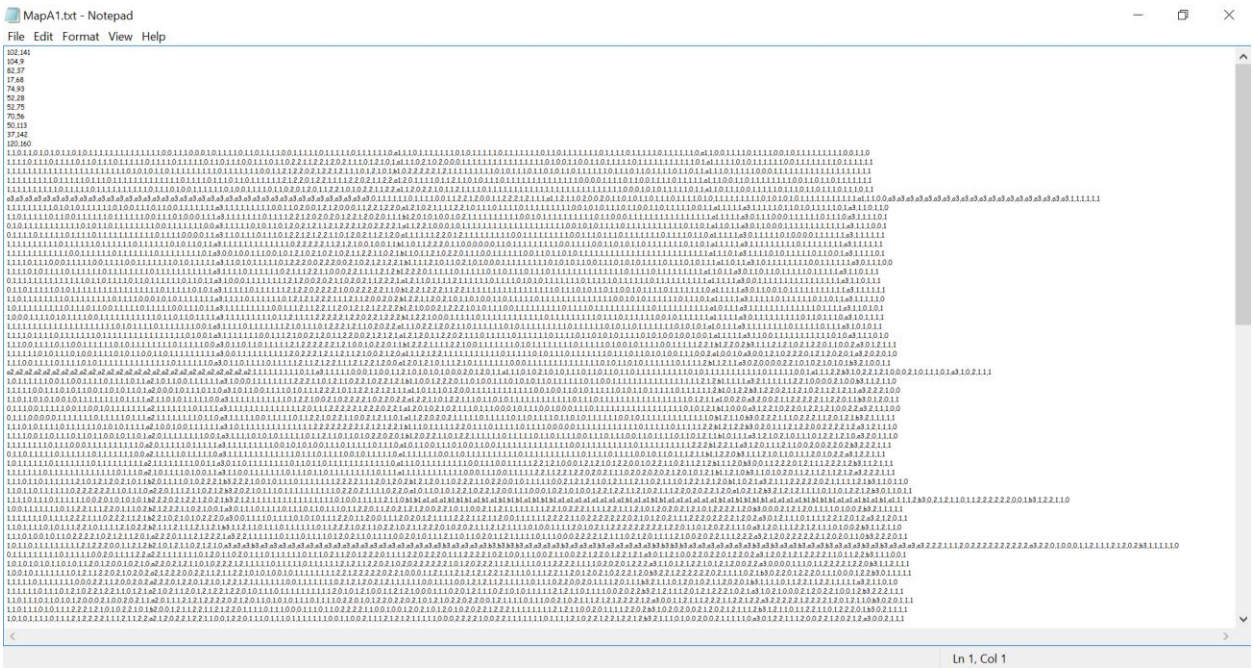

Figure 1 File containing a Map

- Hard to traverse cells on highway are represented by light blue squares.
- The path is represented by red squares.
- The explored cells are tinted with light yellow.

Figure 2 shows the interface with the visualization of a sample map on which we have executed A*. This interface has options to create, read or execute a loaded map by selecting an algorithm and the parameters (such as heuristic and weight) it requires. We can a click on a particular cell on the visualized map to get its h, g and f values computed the A* family algorithms.

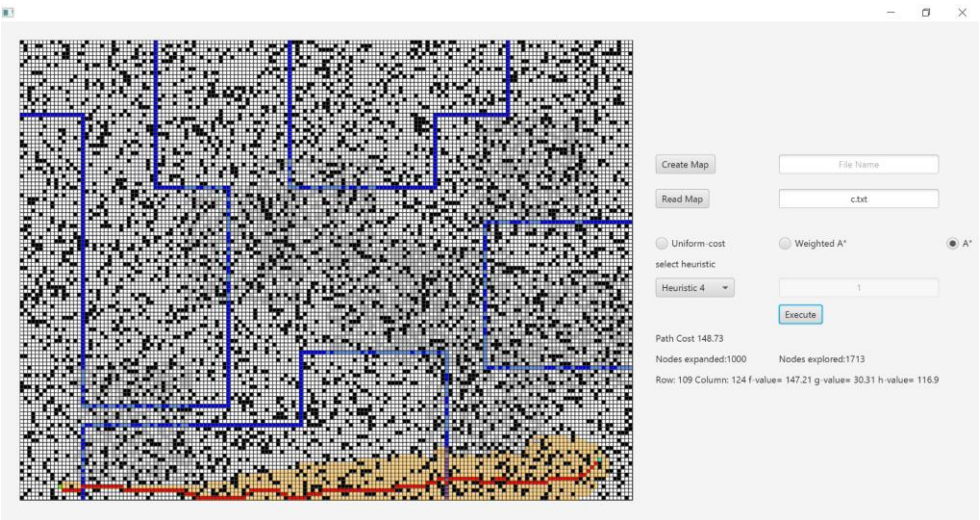

Figure 2 Interface visualizing the map

B. ABSTRACT HEURISTIC ALGORITHM

The family of A* algorithms namely, Uniform-cost search, A* and Weighted A* have been implemented by using a generalized heuristic algorithm written in Java. The evaluation function we used is as follows:

$$f(s') = g(s') + w * h(s')$$

Where, $g(s') = g(s) + c(s, s')$,

w is the weight

and $h(s')$ is the heuristic of node s'

The function g(s) represents the cost to reach the node s from the start node and $c(s, s')$ represents the cost to reach s' from s.

When we want to compute the path using Uniform-cost search, we set the heuristic to zero.

For executing A* on the map, as shown in Figure 2 from the interface, we can select the heuristic we want to use and the weight will be set to 1.

And for weighted A*, we can select both, the heuristic and its weight.

After execution, the generated path and the explored nodes will be shown on the map. The path cost, the number of nodes explored and the number of nodes expanded will also be displayed on the interface.

The following three figures show the execution of the three algorithms on the same map with same start and goal node (we use the same heuristic for A* and weighted A*).

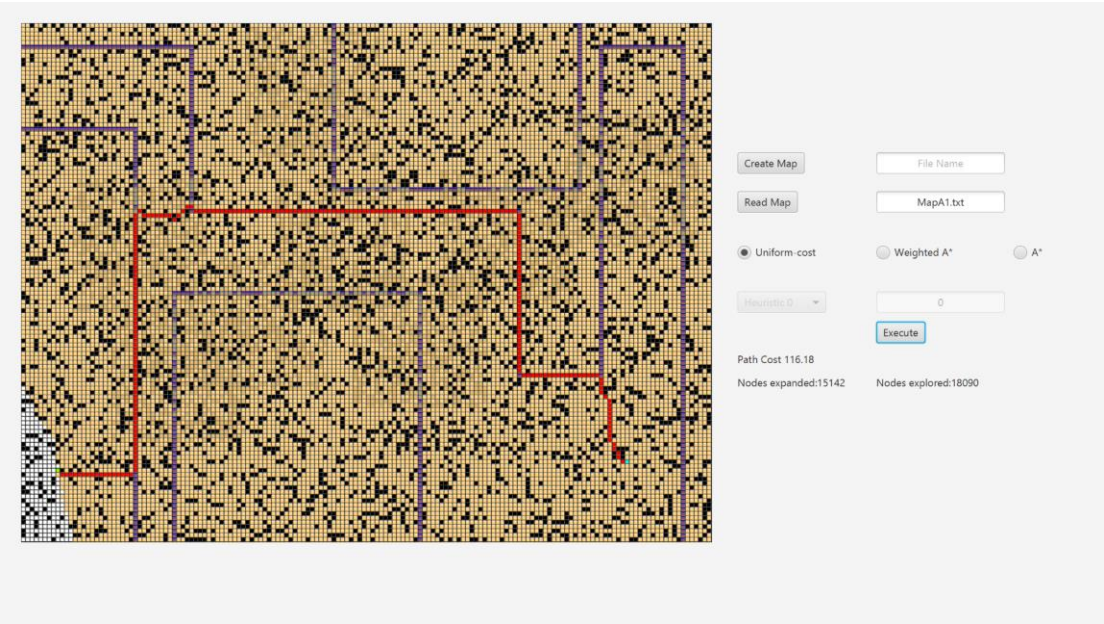

Figure 3 Path using Uniform-cost Search

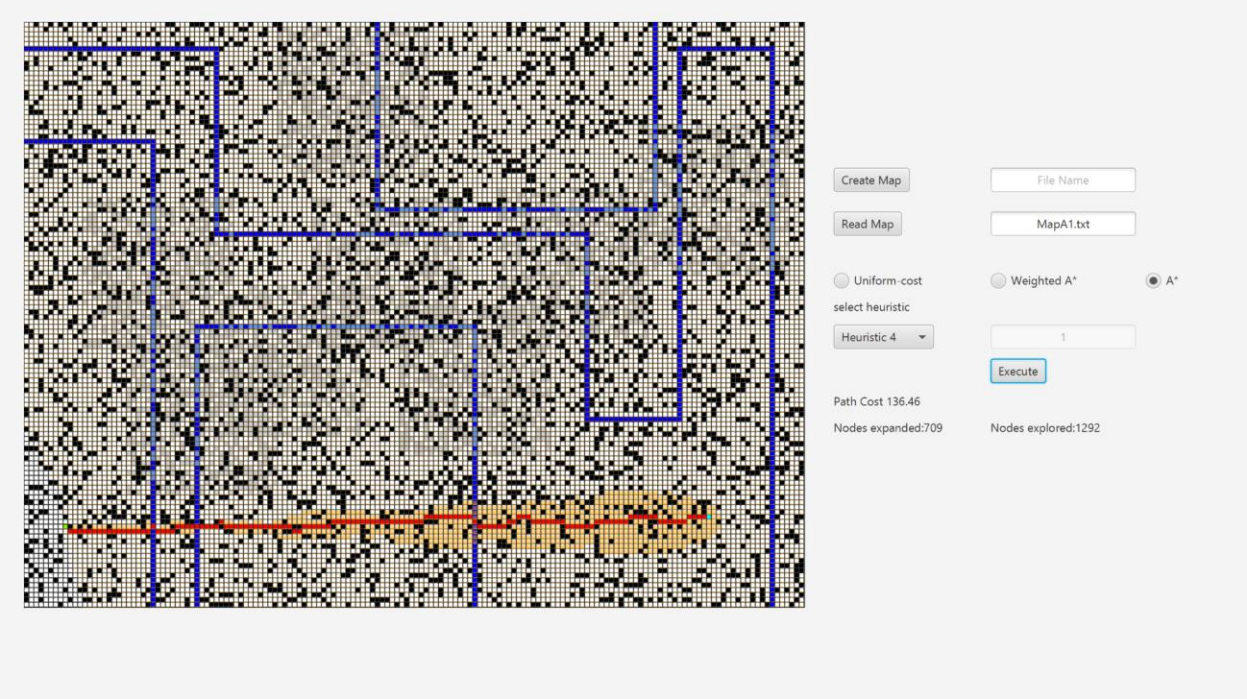

Figure 4 Path using A*

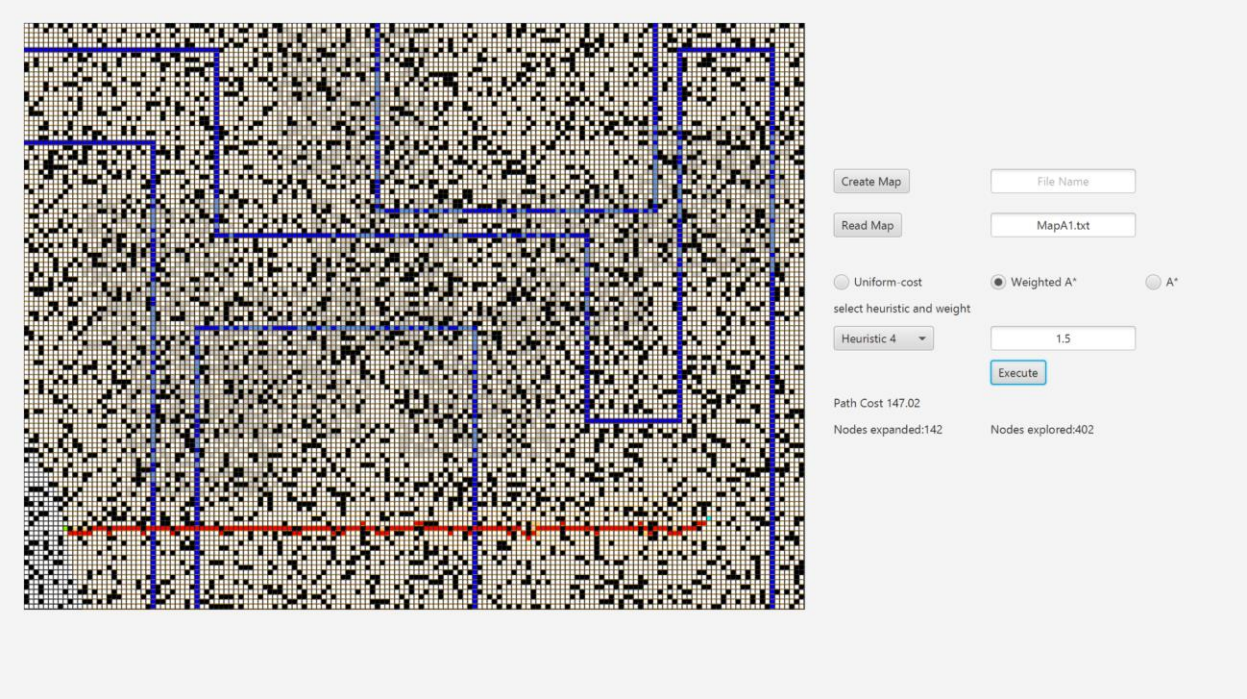

Figure 5 Path using Weighted A*

C. OPTIMIZATION

Binary Heap

We implemented binary heap for fringe nodes in order to get the node with lowest f value in logarithmic time. There for cost of getting the node out of the fringe was significantly reduced, from n to $\log n$ to be precise, where n is the size of the fringe. However, the updating a specific node in the fringe still takes polynomial time with binary heap. On the practical note we found the running time with built-in heap structure of Java slightly better than our implementation. We suspect it uses more advance heap structure than binary heap.

Our binary heap used array of fixed size to store elements. It turns out that if numbered layer by layer starting from 1, children of the node k will be $2k$ and $2k + 1$. We used this property to access the children and parent of nodes, while a node is added or removed.

Closed List

In A* algorithm one has to check weather a node is in closed list or not. Nodes in the closed list are expanded and need not to be expanded again. However, checking for a specific node in closed list of size n requires n operations. As the size of the closed list grows it can lead to a significant overhead as this process need to be done for every generated successor. Instead we used a flag associated with every node which indicates weather the node is in closed list or not. Hence the time for checking gets reduced to constant time.

Ignoring Blocked Cells

We use a special condition to for not to add blocked cells in the fringe. Since, they are blocked no path can pass through blocked cells. Hence the size of fringe gets reduced and the cost for updating node in fringe is reduced.

D. HEURISTICS

For the base of all our heuristics we used three different heuristics, the straight line distance, Manhattan distance and the minimum distance on the plain grid following the rules of the game, we call it diagonal distance. As expected two of this three does not performs well in terms of cost path cost and computation cost. However, it turned out that straight line even being simple works quite well, we will discuss it in detail in the analysis section.

Here, are the formulas used to calculate the distances.

$$Manhattan\ distance = (x_{goal} - x_{node}) + (y_{goal} - y_{node})$$

$$Straight\ line\ distance = \sqrt{(x_{goal} - x_{node})^2 + (y_{goal} - y_{node})^2}$$

$$\begin{aligned} Diagonal\ distance \\ &= \min\big((x_{goal} - x_{node}), (y_{goal} - y_{node})\big) * \sqrt{2} \\ &+ absolute\ difference\big((x_{goal} - x_{node}), (y_{goal} - y_{node})\big) \end{aligned}$$

After that we designed the best admissible heuristic, which due to the highways turned out to be the Manhattan distance on highway, which is just the weighted Manhattan distance. Although, we explored some alternatives this was the best we could get without loss of admissibility and simplicity.

Admissible

The best admissible heuristic we can find was cost to the nearest highway added to Manhattan distance from that highway to the goal. But finding nearest highway is not trivial problem, it requires exploring on its own which is undesirable for any heuristic function. A simplification of this heuristic is used instead. For any cell h is Manhattan distance on highway.

$$h = Manhattan\ distance\ on\ highway$$

$$h = Manhattan\ distnce\ from\ goal * 0.25$$

Inadmissible

For inadmissible heuristic we started with combination of different distances with varying weights. The idea behind it was to minimize combine the best parts of all this heuristic. As we have noticed a particular heuristic performs good over a specific pattern in map and start and goal node. We tried different weights to get the better results on all maps. However, it turned out that it always suffered from the bad part of other heuristics yielding a mediocre result.

The next thing we tried was to use the information of the current cell for better approximation. We designed a heuristic which favored highway cell overs regular cells, using all the distances techniques above. This heuristics, with highway biased showed significant improvement over the regular heuristics.

We event tried to penalize the hard to traverse cells as cells nearby them also hard to traverse with probability 0.5. However, to our surprise the performance of the algorithm degraded. And the idea has to be discarded.

Heuristic	Description
Heuristic 0	No heuristic, Uniform-cost search
Heuristic 1	Admissible Heuristic
Heuristic 2	Straight line distance
Heuristic 3	Manhattan distance
Heuristic 4	Diagonal distance
Heuristic 5	Highway biased heuristic using straight line
Heuristic 6	Even more highway biased heuristic using straight line
Heuristic 7	Highway biased heuristic using Manhattan distance

E. EVALUATION

The following tables give the average values (over the fifty benchmarks) of path cost, nodes expanded, nodes explored, running time and the memory requirements for these three algorithms:

Table 1 A* and Uniform-cost

Heuristics	Path Cost	Nodes Expanded	Nodes Explored	Run Time (in ms)			Mem Req (for 50 in MB)	Avg. Mem Req (MB)
				Avg	Min	Max		
H0	109.27	12361	14816	41	34	51	155	3.1
H1	109.27	8651	10951	36	22	45	106	2.12
H2	124.41	1820	2971	6.4	4	17	90	1.8
H3	155.38	805	1573	2.8	2	8	75	1.5
H4	137.49	1280	2319	4.9	4	13	85	1.7
H5	120.76	1623	2784	7.6	5	13	90	1.8
H6	138.90	418	1167	3.5	2	10	70	1.4
H7	131.88	876	1880	3	4	10	82	1.64

Path cost using H0 = H1 = OPC (optimum path cost)

Path cost using H2 = 1.14*OPC

Path cost using H3 = 1.42*OPC

Path cost using H4 = 1.26*OPC

Path cost using H5 = 1.10*OPC

Path cost using H6 = 1.23*OPC

Path cost using H7 = 1.21*OPC

Table 2 Weighted A* with w=2

Heuristics	Path Cost	Nodes Expanded	Nodes Explored	Run Time (in ms)		
				Avg	Min	Max
H1	111.41	4514	7110	18	14	33
H2	169.02	142	432	0.8	~0	5
H3	185.80	149	452	0.9	~0	4
H4	178.02	143	442	0.8	~0	3
H5	139.05	397	1114	3.6	3	11

H6	142.27	422	1185	3.7	3	16
H7	151.79	393	1105	3.6	2	8

Table 3 Weighted A* with w=1.25

Heuristics	**Path Cost**	**Nodes Expanded**	**Nodes Explored**	**Run Time (in ms)**		
				Avg	**Min**	**Max**
H1	109.56	7412	9677	32	29	42
H2	154.91	336	794	1.8	1	12
H3	172.89	265	672	1.4	1	7
H4	161.46	316	770	1.6	1	5
H5	131.98	538	1364	4.5	3	12
H6	141.73	415	1162	3.7	3	7
H7	141.37	481	1262	3.9	3	10

As we can see from the tables, when we used H1 (Manhattan Distance on Highway) for A* and weighted A*, it always returns the path with optimum cost which is equal to the path cost returned by Uniform-cost search.

F. RESULTS AND OBSERVATIONS

Uniform-cost and A*will return the same path cost when we use an admissible heuristic for A*. But, the drawback with Uniform-cost is that it explores almost all the nodes of the map and hence will have more running time compared to the other two. This can be a drawback for real time applications where the maps may sometimes have several thousand grids.

Weighted A* will explore even lesser nodes than A* but it will return a suboptimal path which costs more than the optimal path returned by the other two algorithms. The path cost increases in proportion to the weight of the heuristic.

The impact of highways on path cost

The superfast highways make it hard to design a good heuristic function. Highways significantly reduce path cost and hence a good heuristic should look for nearby highway, even if they are in opposite direction to the goal node. However, in doing that it explores too many nodes, resulting in bad computation time.

The impact of hard to traverse region was less than the impact of highways.

Straight line distance

Straight line distance and its variants turned out to be very good heuristics compared to others. Some of the other heuristics showed potential, but the heuristics with straight line as their backbone almost always performed better. A significantly good one was heuristic 5, biased towards highways it demonstrated path 10% longer than the optimal, expanding only 13% of nodes compared to uniform cost search. It’s weighted variants also performed quite well. With weight 1.25 it yielded a 20% longer path expanding only 4.3% of nodes compared to uniform cost search.

Highway bias

Highway biased variants always performed well compared to the simple one. All the best heuristics we have uses highway bias along with straight line.

Effect of weight

With different weight significant performance gain was noticed without losing much of optimality. Weighted variant of our admissible heuristic provided 2% longer path, reducing the number of nodes expanded to about half.