

Artificial Intelligence: Heuristic Search

Nathan Morgenstern, Seo Bo Shim

November 27, 2017

Contents

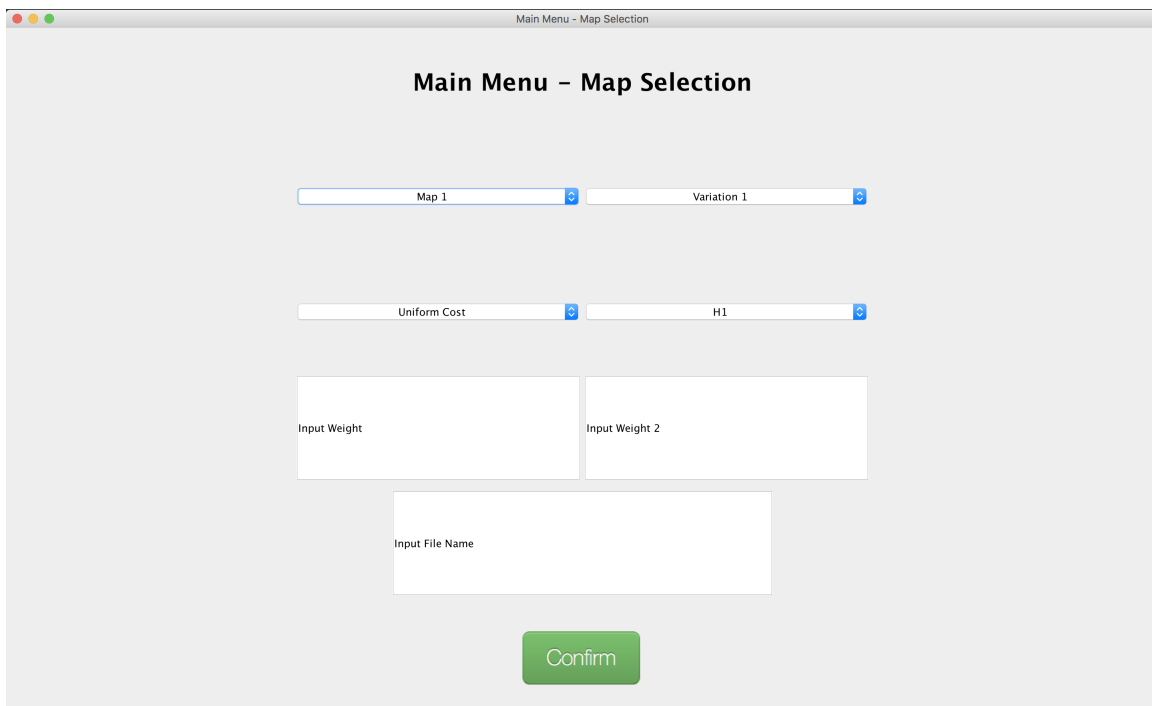
(a) Introduction to GUI	3
(b) Correct Solutions from instantiations of heuristic algorithm	4
(c) Optimization of heuristic algorithm	6
Overall Optimizations	6
(d) Heuristic Proposals	7
Best admissible/consistent heuristic	7
Inadmissible heuristics	8
(e,f,g)Heuristic Results and Comparisons	13
Experimental Results	13
Comparison	14
(h) Sequential A*	17
Implementation	17
Sequential Heuristic	17

(i) Sub-optimality of Sequential A^*	19
--	----

(a) Introduction to GUI

When the application first starts the user has the option to select either their own map, or a map that has been included by default. If the user wants to load their own map, they should select user generated map and input the file path into the file input text field. Otherwise, they can simply select a pre-generated map from the drop down menu along with its variation.

There are a total of 4 algorithms included in the main menu that the user can then select from to find a path from the start to goal. Options include: Uniform cost search, A*, Weighted A*, and Sequential A*. If the user selects weighted or sequential A* they should input two weights into the weight text fields. Once all the parameters are selected by the user, they can select confirm and the map along with its solution using the selected algorithm will be loaded.



The screenshot shows a window titled "Main Menu - Map Selection". The window has a light gray background. At the top, the title "Main Menu - Map Selection" is centered in bold black text. Below the title, there are two horizontal dropdown menus. The first dropdown is labeled "Map 1" and the second is labeled "Variation 1". Below these, there are two more horizontal dropdown menus. The first is labeled "Uniform Cost" and the second is labeled "H1". Below these dropdowns, there are two input fields side-by-side. The left field is labeled "Input Weight" and the right field is labeled "Input Weight 2". Below these two fields, there is a single input field labeled "Input File Name". At the bottom center of the window, there is a green button with the text "Confirm" in white.

Figure 1: The main menu of the application

(b) Correct Solutions from instantiations of heuristic algorithm

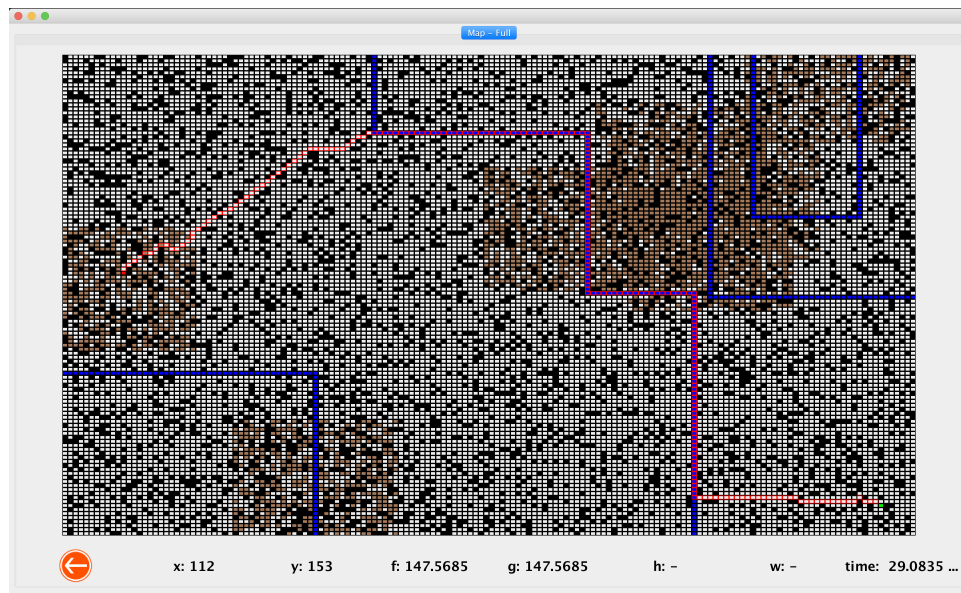


Figure 2: Uniform Cost Search example

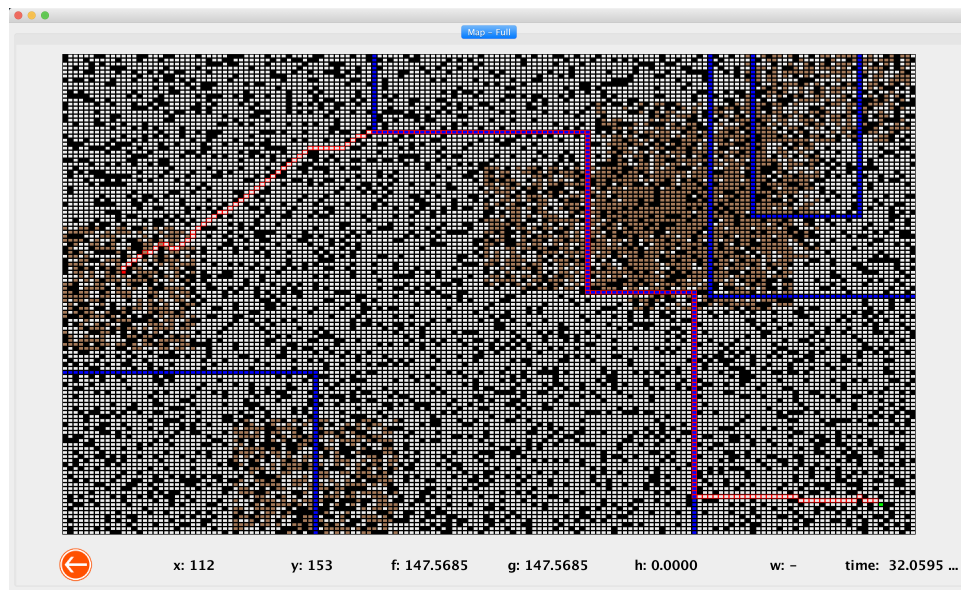


Figure 3: A* Search example - Heuristic 1

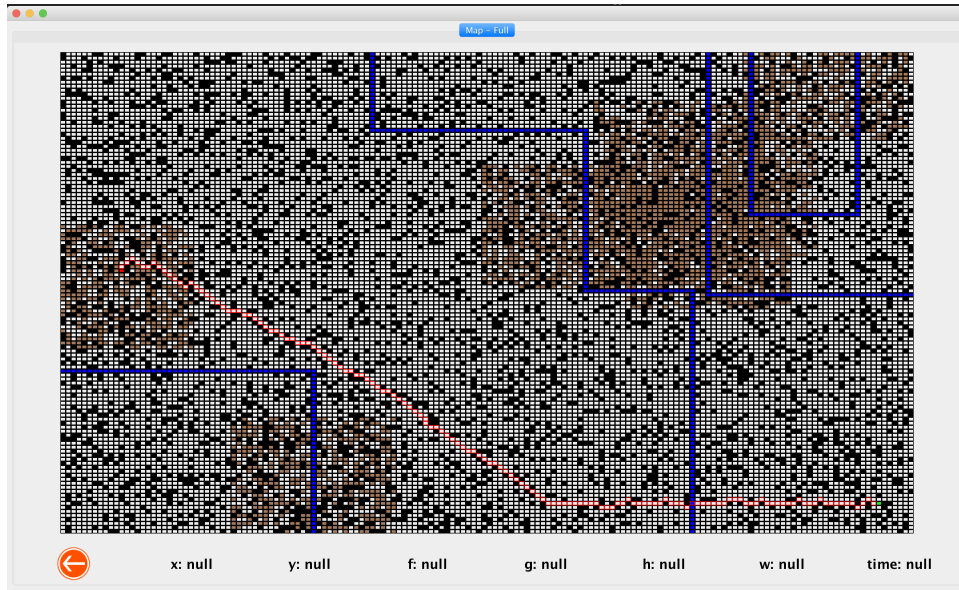


Figure 4: Weighted A* Search example - Heuristic 2 $w_1 = 1.25$, $w_2 = 2$

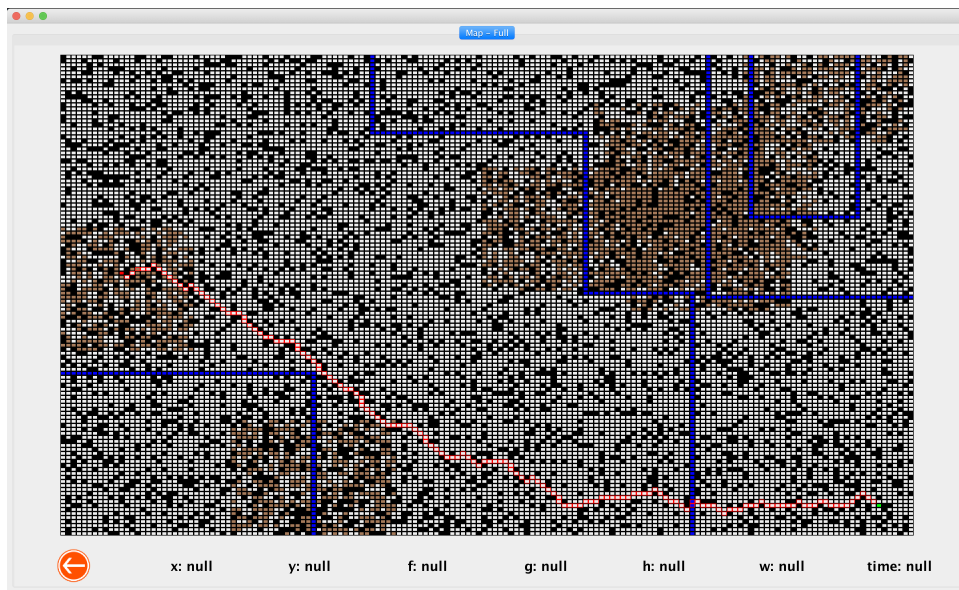


Figure 5: Sequential A* Search example - $w_1 = 1.25$, $w_2 = 2$

(c) Optimization of heuristic algorithm

Overall Optimizations

When searching for the neighboring nodes to expand to, we can ignore the neighbor that is the parent because the parent was already explored. This saves the computation time of calculating the path and adding/removing from the binary heap.

The table that keeps the explored set of nodes is a 2 dimensional array. Since our grid is a 2d table, each node is assigned a coordinate. In the grid object that contains the graph of the map and all the nodes, we store a two dimensional array of nodes that represents the grid. This array allows for a $O(1)$ complexity in random access time.

Also for keeping the fringe or the frontier of the graph search, we use a priority queue which is equivalent to a priority heap. We desire a minimal time for a search operation, and a binary heap provides this in $O(\log(n))$ time. Finding the highest element also takes $O(1)$ and inserting an element takes $O(\log(n))$ time, which is also significant in choosing this data structure to store our possible nodes in our frontier.

Since the overall algorithm is the same for all the types of searches, there were no optimizations specific to each type of search. Each search is different in how it calculates the value, which can consist of the actual path cost to the node or include the heuristic as well. In the abstract algorithm we call an abstract method, where the implementation of this method is the only difference between the searches.

(d) Heuristic Proposals

Best admissible/consistent heuristic

The best heuristic can be achieved by considering the minimum path cost from the start to end goal.

We will build the shortest path based on two cases. A straight line and diagonal movement.

A path from the start node to the goal node is a straight line

The shortest possible path can occur if a river runs straight from the start to the goal node. This is possible because the river can possibly continue straight across the whole grid. This configuration allows the minimal past cost for this situation. The heuristic value of a node, the predicted cost from the node to the goal:

s is the current node, s_g is the goal node coordinates.

$t = 0.25$: multiplier for the type of block.

$d = 1.0$: multiplier for direction of movement.

$$h(s) = t * d * \max(|s^y - s_g^y|, |s^x - s_g^x|)$$

$$h(s) = 0.25 * 1 * \max(|s^y - s_g^y|, |s^x - s_g^x|)$$

A path from the start node to goal node has diagonals and straight lines.

We note that it costs less to move non-diagonally on highways than than moving diagonally along non-rivers.

The cost of moving diagonally is:

$$h(s) 4 * 0.25 * \sqrt{2} * \sqrt{2} * \min(|s^y - s_g^y|, |s^x - s_g^x|)$$

We note that moves can only be made vertically or horizontally on a highway. So we calculate the total

amount of vertical and horizontal moves.

$$h(s) = t * d * (vertdist + horizdist)$$

$$h(s) = 0.25 * 1 * (|s^y - s_g^y| + |s^x - s_g^x|)$$

We can clearly see that this heuristic is minimized. The heuristic can theoretically be minimized further by taking into account the maximum possible occurrences of the rivers along the path, however we decided to keep this heuristic for simplicity's sake.

Inadmissible heuristics

We argue that inadmissible heuristics should be used for the A* algorithm on this graph because the range of possible values for the path cost in this grid is high. This is due to the presence of rivers and possibility of the path intersecting a region of hard to travel cells. Since the range is high, to select a admissible heuristic, we must calculate the heuristic conservatively to not exceed the minimum path cost to keep the admissibility criteria. This is why the admissible and consistent A* implementation almost exactly follows the Uniform Cost Search algorithm, because the heuristic is determined to be low and less significant than the actual cost. Ideally, we wish for the minimum path cost to be close to the average, or the range to be smaller so that the heuristic value is comparable to the actual cost. This is the admissibility condition:

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

$$h(s) - h(s') \leq \min(c(s, s'))$$

To find the maximal heuristic that fits this condition, we must find the minimum possible cost value. Based on our argument, we'd like to propose a more relaxed condition that results in a heuristic closer to the average of the actual path cost:

$$h(s) - h(s') \leq \text{average}(c(s, s'))$$

We desire a heuristic that is comparable to the average cost of traversal, as this heuristic will be inadmissible, but it will likely make a larger difference in how nodes are prioritized.

H2: Type-based Heuristic

Currently, the abstract algorithm tends to follow rivers because the path cost to these rivers is small. However this is computationally expensive, as these nodes will have to continuously be expanded along and not along the river. We can argue that the path should follow the river as long as this does not increase the Euclidean distance from the current node to the goal.

If the current node is on a river, then the heuristic can be calculated to be:

$$h(s) = 0.38 * 1 * (|s^y - s_g^y| + |s^x - s_g^x|)$$

If the river takes us away from the goal, then the heuristic will be large enough that a step of 1 will be enough to have a lower path cost than following the river away from the goal.

The result is that nodes along the river going away from the goal will be expanded less, thus reducing computation time when the path is on a river.

When the current node is a hard to traverse cell, we make the assumption that this region will be full, so we avoid these cells by raising the heuristic value. We calculate that traversing around a 31x31 grid of hard-to-traverse cells is slightly shorter than traversing through the 31x31 grid. Since there's only a 50 percent chance of a hard-to-traverse region of having a cell as hard to traverse. Note: traveling through the difficult blocks on average will yield a path cost of 1.5.

Traversing around the 31x31 block:

$$30 + 30 + \sqrt{2} = 61.41$$

Traversing through the 31x31 block

$$\sqrt{2} * 30 * (1 + (1 * (0.50))) = 65.76$$

61.41 < 65.76. It is advantageous to travel around the 31x31 grid. Though it may have a higher path cost to travel around the block, because with a larger number of blocks. So this heuristic should be designed to have the path best avoid these hard-to-traverse cells, not necessarily avoid them completely.

$$h(s) = 1.5 * 1 * (|s^y - s_g^y| + |s^x - s_g^x|)$$

When the current node is a regular unblocked cell, we a modified version of the admissible heuristic:

$$h(s) = t1 * d1 * \max(vert_{dist}, horiz_{dist}) + t2 * d2 * \min(vert_{dist}, horiz_{dist})$$

$$h(s) = 1 * 1 * \min(|s^y - s_g^y|, |s^x - s_g^x|) + 1 * \sqrt{2} * \max(|s^y - s_g^y|, |s^x - s_g^x|)$$

s is the current node, s_g is the goal node coordinates.

$t1 = t2 = 1$: multiplier for the type of block.

$d1 = 1.0$, $d2 = \sqrt{2}$: multiplier for direction of movement.

This heuristic is advantageous because since it uses the current node type, it is computationally inexpensive.

H3: Expected Value

This method will take into account the number of blocked cells and the number of hard to traverse cells on average on a grid. The value that's generated from the heuristic will take the probability of each type of cell occurring and the required cost to traverse this path. The heuristic value will be the average cost of each step combined with a value thats proportional to the node's distance to the goal. The Euclidean distance will be used.

Average cost of step = summation of: probability of scenario * cost of scenario.

$$average(s) = \sum P(s) * C(s)$$

For each blocked cell, we can consider the cost to navigate around it as the cost of having the blocked cell. We consider the different cases of how a blocked cell can be overcome. Approaching from a corner:

$$(2 + \sqrt{2}) + 2 * (1 + \sqrt{2}) + 2 * 2 + 2 * 1 / 7 = 2.03$$

Approaching the blocked cell perpendicularly:

$$(2 * \sqrt{2}) + 2 * (1 + \sqrt{2}) + 2 * (\sqrt{2}) + 2 * 1 / 7 = 1.78$$

Averaging those two values together divided by the average number of blocks traversed: $1.91/1.5 = 1.27$ cost for each blocked cell.

There is an 80 percent chance that a path will be unblocked. For each 31x31 grid of 50 percent hard to traverse blocks, this covers up roughly 5 percent of the 120 x 160 grid. Eight 31 x 31 grids result in about 40 percent of the grid covered, consequently about 20 percent of the map is covered by hard to traverse cells.

$$average(s) = P(blocked) * Cost(blocked) + P(unblocked) * Avg(C(unblocked))$$

$$Avg(C(unblocked)) = P(unblocked) * Cost(unblocked) + P(hard) * Cost(hard)$$

The result is:

$$average(s) = (0.2 * 1.91 + 0.8 * (0.8 * 1 + 8 * 0.05 * 0.5 * 1.5)) = 0.3 + 0.8 = 1.355$$

Not exactly 20 percent of the map is covered by hard to traverse cells though, as there is a chance of overlap; each hard to traverse cell is not uniquely placed. There's also a chance the cell will land on the edge and not all of the 31x31 chosen cells will be on the grid.

This heuristic also ignores the probability of there being a river, so this heuristic will overestimate the path cost of reaching the goal. This heuristic can possibly be optimized by having a constant value that can be adjusted to account for the probability of rivers occurring in the optimal path.

H4: Manhattan Distance

This heuristic will take the Manhattan distance from the current node to the end goal. This heuristic is useful when diagonal steps are not possible in the grid. This heuristic is similar to the Euclidean distance, except we assume that diagonal steps are not taken.

$$h(s) = |s^y - s_g^y| + |s^x - s_g^x|$$

This heuristic will focus on finding the straightest route from the node to the goal. This heuristic is expected to cause the algorithm to be quick.

H5: Euclidean Distance

This heuristic will take the Euclidean distance from the current node to the end node. In other words, the distance will be calculated as a line directly from the node to the goal node. This heuristic may provide the most accurate representation of the cost from the node to the goal disregarding the presence of rivers.

$$h(s) = \sqrt{|s^y - s_g^y|^2 + |s^x - s_g^x|^2}$$

This heuristic is useful because it'll prioritize the actual distance of the goal from the current node. It will result in little nodes being expanded and focus on the straightest route to the goal.

(e,f,g)Heuristic Results and Comparisons

Experimental Results

Algorithm	Mean Time	Mean Nodes	Optimal Path Length	Path Cost
Uniform Cost	21.9412 ms	12907	179.66	106.6067

Algorithm	Mean Time	Mean Nodes	Percent from optimal	Path Cost
A* - H1	21.6008 ms	10374	0.000 %	0.000 %
A* - H2	2.9807 ms	460	-5.755 %	23.6357 %
A* - H3	0.9796 ms	217	-28.877 %	36.2839 %
A* - H4	1.8209 ms	754	-16.932 %	18.6720 %
A* - H5	3.3776 ms	1660	-15.886 %	9.1820 %

Algorithm	Mean Time	Mean Nodes	Percent from optimal	Path Cost
A* - H1 W=1.25	23.4999 ms	9486	-1.747 %	0.0662 %
A* - H2 W=1.25	4.2639 ms	417	-4.386 %	25.6196 %
A* - H3 W=1.25	1.9081 ms	136	-33.074 %	47.9680 %
A* - H4 W=1.25	1.4664 ms	288	-30.669 %	55.3635 %
A* - H5 W=1.25	0.9395 ms	376	-22.821 %	27.5237 %

Algorithm	Mean Time	Mean Nodes	Percent from optimal	Path Cost
A* - H1 W=2	14.2124 ms	6773	-5.143 %	2.2443 %
A* - H2 W=2	2.0119 ms	398	0.256 %	29.7685 %
A* - H3 W=2	0.4456 ms	120	-34.220%	58.8972 %
A* - H4 W=2	0.5921 ms	131	-31.882 %	70.2318 %
A* - H5 W=2	0.3917 ms	127	-33.819 %	51.0543 %

Algorithm: Sequential A*	Mean Time	Mean Nodes	Percent from optimal	Path Cost
W1=1.25 W2=2	10.0905 ms	817	-29.967 %	38.3617 %
W1=1.05 W2=1.25	5.4086 ms	1637.3400	-24.2569 %	32.4471 %
W1=1.00 W2=1.1	10.1010 ms	4648.3600	-16.5980 %	20.9319 %

The memory requirements for each of these heuristics were the same.

Comparison

Mean Time: Average amount of time it took for the search to execute.

Mean Nodes: Average amount of nodes that were expanded during the search.

Percent from optimal: Average percentage from the optimal path length.

Path Cost: Average percentage from the optimal path cost.

As expected, the consistent and admissible heuristic H1 behaved similarly to the uniform cost search. It expands less nodes than the uniform cost search because the heuristic prioritizes the nodes moving towards the goal than away from the goal. The mean time was also decreased because of the less amount of nodes

expanded. The consistent and admissible heuristic is optimal.

Comparison of H2-H5

Comparing the other four heuristics, we found that H3-5 were far from optimal and took the least amount of time. These three heuristics were primarily based on the distance between the node and the goal. H2 also depended on the distance to the goal, but also took into account whether a river was present to determine the heuristic value.

We can see the effect of having a higher weight. Especially since the heuristics are proportional to the node's distance from the goal node, we find a more direct path as the weight increases. This is evident in the weighted A* approach and from a general observation of the heuristics.

H3 was based on expected value, where we took a very rough consideration for the cost of the average path traversal. Each path cost was expected to be around 1.5, when in reality this is not the case because of the presence of rivers. We omitted the probability of the path including a river, because the probability is difficult to estimate and many assumptions would have to be made. We also found it useful to notice the difference between heuristics that take into account the existence of rivers and those that didn't. H2 accounted for rivers where the others did not.

If H2 does not encounter any rivers, it behaves very similarly to H4 and H5. H2 performs better as the weight on the heuristic increases, as H2 is weaker than the other heuristics, so the weight does not affect the search as much.

H3 is expected to perform the worst, because there are only several different maps and 10 different variations within the maps. As the heuristic value was determined based on probability, we expect it's performance to increase with a larger dataset. Through the study of the H3 heuristic, we realize that it is highly probable that the path will be aided greatly by a river and this should be accounted in our heuristic to improve it.

H4 and H5 are very similar, as they are the Manhattan and Euclidean distances. It appears though that path cost for the Euclidean distance is much lower than the path cost for the Manhattan distance. There's a significant difference of about 10 percent while there's not much difference in path length. We can attribute this to how the Euclidean distance expands more nodes, likely because it's heuristic value is also guaranteed to be lower than the Manhattan distance.

On average, H5 has the lowest heuristic value, with H4 and H2 coming close behind, and H3 having the highest heuristic value. This is how the heuristics would rank in terms of minimizing path cost.

Path Length vs Path Cost

Generally, we noticed that as the path length is higher, then the path cost is also lower. We recognized a pattern where if a path length is higher, it often means that a path included a river. In this case the path cost often ends up being lower. If the path length is shorter, it often means that the heuristic dominated and caused the path to go almost straight from the start to the goal without many deviations.

Mean Time

Overall, we observed that in general the mean time decreases as the mean number of nodes also decreases. This makes sense as less time is spend expanding nodes, the solution can be reached faster.

(h) Sequential A*

Implementation

The implementation of the sequential A* algorithm is efficient because it utilizes various data structures to minimize the time needed to access the data. The frontier priority queue and the 2D closed array for each heuristic are kept in a list. This ArrayList only holds five elements, equivalent to the number of different heuristics. The ArrayList allows for random access based on the index, which is useful because the algorithm switches between the different heuristics constantly so the access times should be minimized.

Again, a priority heap is utilized to minimize the search time and the access time for the minimal value in the heap.

The parents of all the nodes are only updated when the goal is reached. A 2D array is kept to temporarily store the parent of each node at a coordinate, and only at the end the node's actual parents are updated. Parents of a node can change often while the optimal cost path is in the process of being found, so this reduces the need to have access to the node object until the end.

Sequential Heuristic

Several weights were tested with the sequential heuristic value. Differences in W1 would be predictable as this what we observe through the Weighted A* algorithm. A higher W1 value causes a stronger heuristic and expands less nodes as it favors solutions closer to the goal.

We already know that W1 has a dramatic effect on the mean time for the search algorithm. A higher W2 means that the other heuristic values are desired over the admissible heuristic. As we decrease the W2 value, we see that more nodes are expanded, alike to the admissible heuristic case (H1). Additionally, we find the path length increasing, and the path cost decreasing, which results in the optimality of the solution.

It's important to note the difference in the number of nodes expanded from a change of $W1 = 1.05$ to $W1 = 1.00$, and $W2 = 1.25$ to 1.1 . This slight change in the weights highlights the sensitivity of the heuristic. If we wanted to create a more robust heuristic, we would study the optimal weights more closely at this level - fine tuning it according to the different implemented heuristics.

Overall, the sequential heuristic performed worse than the other heuristic searches. While having a high mean time (6.5ms), and only surpassing one heuristic in terms of its path cost. The sequential heuristic search performs worse, perhaps the overhead of switching between the heuristic values slows down the algorithm. There is more computation involved in this case.

We can hypothesize that because the heuristics we have implemented have high values, it's not the sequential A* algorithm was not the best. The algorithm will tend towards the goal directly instead of the optimal route. If we observe the path length of the sequential algorithm, we can see that these percentages are comparable to H3, H4, H5.

(i) Sub-optimality of Sequential A*

In the anchor search of the sequential A*, we know that the heuristic is admissible and consistent. The cost from the start to a node is less than the optimum cost * the weight.

$$Keys(s) \leq w_1 * g * (s_{goal})$$

$$Keys(s) = g(s) + w_1 * h(s)$$

So:

$$g(s) + w_1 * h(s) \leq w_1 * g * (s_{goal})$$

Remember that the heuristic is admissible and consistent.

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

The anchor key's heuristic value will never overestimate the path to the goal.

$$h(s) + g(s) \leq g(s_{goal})$$

The above statement is equivalent to: the path to s + the estimate path cost from s to the goal \leq the actual path cost to the goal. (Based on the admissible and consistent requirement of the heuristic).

$$h(s) \leq g(s_{goal}) - g(s)$$

We can apply this inequality to the condition now:

$$g(s) + w_1 * (g_{s_{goal}} - g(s)) \leq w_1 * g * (s_{goal})$$

$$g(s) * (1 - w_1) + w_1 * g_{s_{goal}} \leq w_1 * g * (s_{goal})$$

And since $w_1 \geq 1$, we know for sure that this inequality is true.

The sequential A* terminates when:

$$Open_iMinkey() \leq w_2 Open_0Minkey() \text{ and}$$

$$g_i(s_{goal}) \leq Open_iMinkey()$$

or

$$g_0(s_{goal}) \leq Open_0Minkey()$$

We'll pay attention to the first case where:

$$g_i(s_{goal}) \leq Open_iMinkey()$$

The cost to arrive at the goal via the i th heuristic is less than the least key value of the i th heuristic search.

Since the minimum key of the i th heuristic is less than w_2 * key of the anchor search value, we can replace the key of the open list with the key of the open list in the admissible heuristic.

$$g_i(s_{goal}) \leq w_2 * Open_0Minkey()$$

We know that the minimum key of the admissible heuristic is suboptimal according to w_1 .

$$Open_0Minkey() < w_1 * c * (s_{goal})$$

Making the substitution:

$$g_i(s_{goal}) \leq w_2 * w_1 * c * (s_{goal})$$

The cost of the path to the goal based on the i th heuristic is limited by the w_2 and w_1 factors and the optimal path to the goal.

In other words, w_1 and w_2 should be minimized for the more optimal solution with sequential A* search.