Introduction to Artificial Intelligence: Assignment 1

Andrew Cannon and Matthew Purri

February 6, 2017

1 Introduction

The objective of this project was to become familiar with various search algorithms. We needed to create a map of obstacles according to specific instructions. The Uniform-cost search, A*, and Weighted A* search algorithms were implemented to traverse our created map.

2 Questions

2.1 Section A

Our program was successfully able to create a map according to the given rules and load a generated map with a start-goal vertex pair. The generated text file includes 10 start-goal pairs, 8 coordinates for the center of hard to traverse regions, and a map of terrain features. When the file has been successfully been loaded the user can click on a pixel or cell on the map to see the coordinate, f(n), g(n), and h(n) values generated from the solving algorithm. An example of the MATLAB interface is shown in figure 1. In figure 2 we show the path along with opened nodes and fringe nodes. Our color code is shown in a table which is located in the appendix section. Our user interface asks the user to select a path file which will load the corresponding map, start-goal pair, and open nodes.

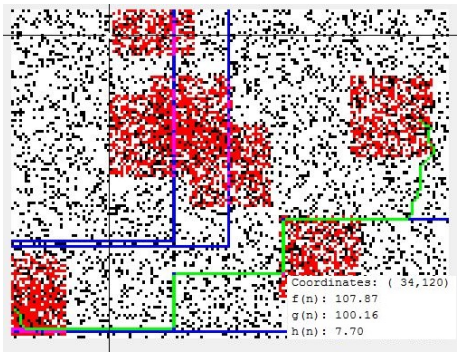

Figure 1: The user is able to click on a cell to display a cells coordinates, f(n), g(n), and h(n) value.

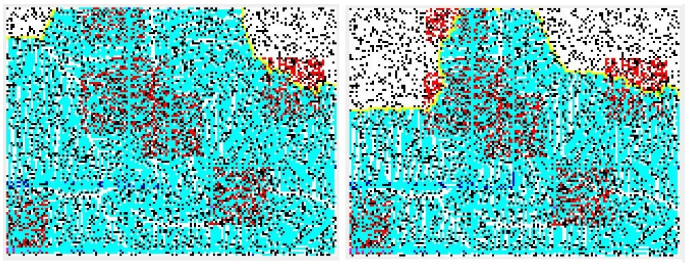

Figure 2: The image on the left is a uniform cost algorithms and the right is the A* algorithm with a weight of 0.25.

2.2 Section B

Three pictures of three different algorithms solving one set of maps and start-goal pairs. Each picture will have a map with and without fringe. The true optimal path can be found by using the uniform cost search, which is an uniformed search. Our A* generated paths will be compared against the uniform cost path to see whether they are optimal or not.

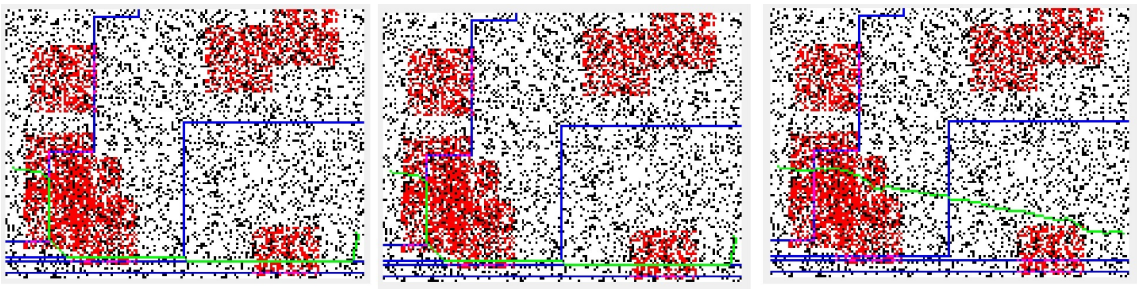

Figure 3: From left to right: Path produced from uniform-cost search, using Euclidean distance heuristic (weight of 0.25), using Euclidean distance heuristic (weight = 2)

2.3 Section C

In order to optimize our A* algorithm we have, in general, two options. We can produce the optimal path by using a heuristic that is consistent or we can produce a path that may be acceptable but not optimal. Depending upon the type of type of problem A* is solving, a reduction in run time and memory usage may be needed. As shown later in our report, the degree to which these amounts can be reduced can be adjusted by increasing the weight on the heuristic value.

2.4 Section D

There are many heuristics that can be used for this problem. The simplest is the Manhattan distance heuristic between the current and goal nodes because the map is made of discrete cells that have one neighbor above, below, and on both sides. This heuristic is admissible because it adds the constraint that the path cannot move diagonally and therefore an additional cost. Crossing diagonally between two cells costs less than going horizontally to a cell then vertically then a cell, so limiting our estimation of the cost to the goal in this way may not help our algorithm. Another simple heuristic is to use the Euclidean distance from the current node to the goal node. However, this breaks the admissibility requirement because it can underestimate the cost to the goal. Based on the requirements for our map, often the vertical distance between start and goal nodes is lower than the horizontal distance. If we lower the heuristic cost of traversing in a horizontal direction, we favor moves in that direction. This could lead to finding the goal node faster but is obviously not a consistent heuristic. An admissible and consistent heuristic could be solving the subproblem of finding the lowest cost route if there were no blocked cells. This is a similar method used for the 8-puzzle. In the 8-puzzle problem finding the optimal solution to only getting a couple tiles to the correct position sometimes we better than using the Manhattan distance. Since we are solving a more relaxed problem, i.e. there are no blocked cells, we are assured that we have an admissible and consistent heuristic. Instead of finding the distance between the start and goal node we could try to find the closest distance to perimeter around the goal node. The heuristic would find the best path to that region and then could solve the smaller problem of getting to the goal from the surrounding region.

2.5 Section E

The colors in the bar graphs correspond to different weights, red is a weight of 2, green is a weight of 1.25, and yellow is a weight of 0.25.

Heuristics:

1. Euclidean distance

2. Manhattan distance
3. No blocked cells
4. Horizontal cells cost less than vertical
5. Solve smaller problem - Get to perimeter region surrounding cell

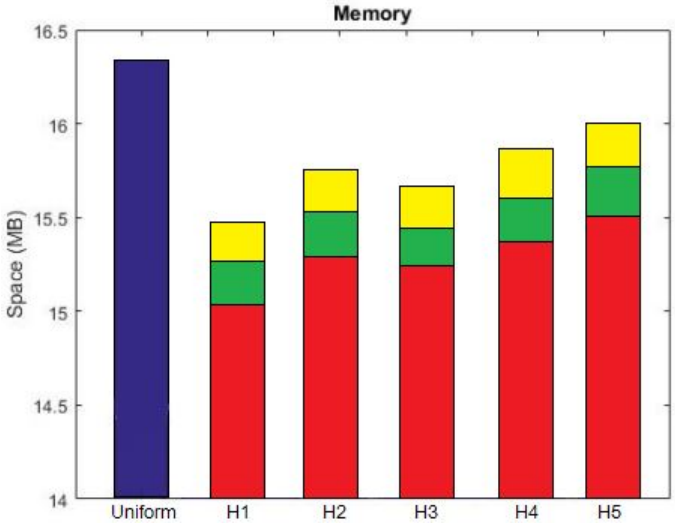

Figure 4: The memory usage across all algorithms used.

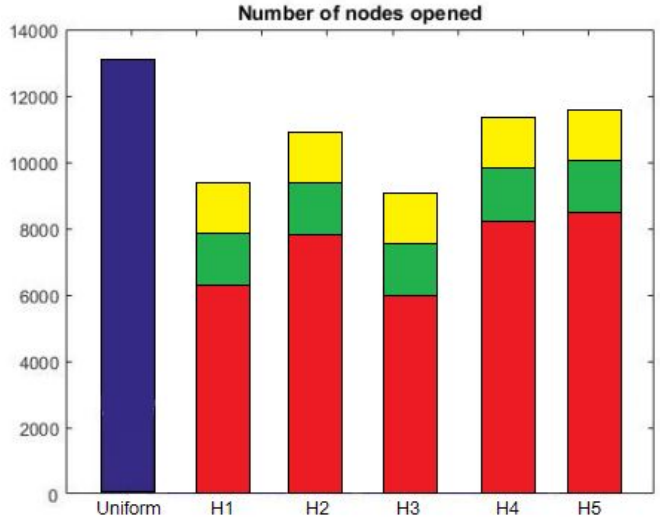

Figure 5: The average number of nodes opened when finding the path from the start to goal node.

2.6 Section F

In terms of memory the uniform cost algorithm performed significantly worse when compared to the other algorithms. The Euclidean distance heuristic used the least amount of memory and this due to the lower number of nodes that it opened. The fifth heuristic had the worst performance among the A* algorithms since the problem had to be put into smaller subproblems. We see a correlation between the memory usage and the number of nodes opened. As the algorithm weights the heuristic more the memory usage goes down by a proportional amount. The memory space measurement was calculated using the Visual Studio performance diagnostic in the 2015 version.

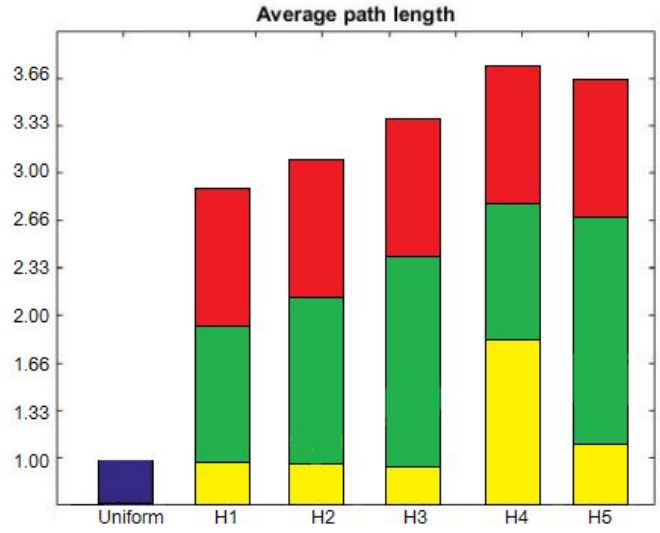

Figure 6: The average successful path length.

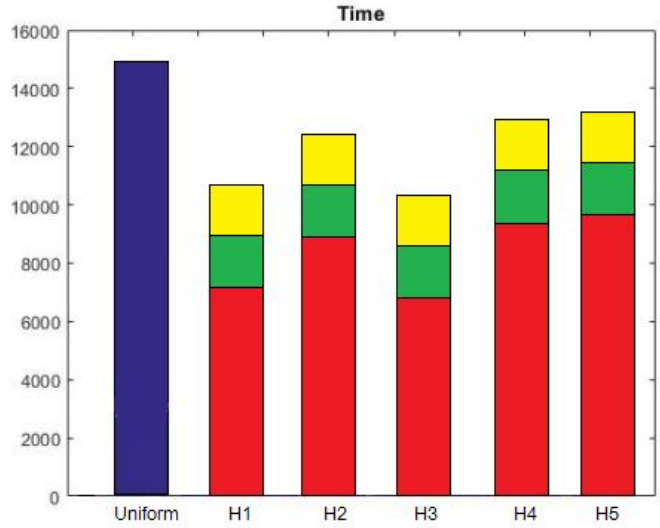

Figure 7: The average time the program ran to find a path.

The algorithm with the fewest opened nodes was the no blocked cells heuristic. It performed the best most likely due to how our maps were distributed. If there were more blocked cells in the way of the start and goal path this heuristic would not perform well. The next best was the L2 distance A* algorithm. The L2 distance is a good heuristic for this type of problem as we are traveling through a discrete map with obstacles. The worst of the A* algorithms was H5 and this was surprising since our guess was that making the problem smaller on average would reduce the number of nodes opened, this was not the case. If we altered the size of the smaller problem we may have seen a better performance. The worst again was the uniform cost algorithm which opened the most nodes. The results of this can be seen in figure 1.

The horizontal cost heuristic performed the worst among the A* algorithms. The heuristic may have performed worst as a result of the assumption that horizontal paths were favored above more diagonal paths. Our start goal pairs may have been too far away vertically for this assumption to help. The uniform cost algorithm average path length was by far the smallest when compared to the A* algorithms. This is because the uniform cost algorithm opens the largest number of nodes. The red, green, and yellow trend again is seen in this and all of the other charts. The higher the weight the worst A* algorithms perform.

In regard to time complexity the A* algorithms performed better than the uniform cost algo-

rithm. The uniform cost algorithm did not use a heuristic and routinely would reach the end of the map as shown in figure 1. The A* Euclidean distance with weight of 0.25 performed the best. This algorithm was nearly two times as fast as the A* algorithm with the H5 heuristic. We can assume that this heuristic has dominance over the other heuristics in this respect and is the least optimistic. When an algorithm relied on its heuristic less, a smaller weight value, the algorithm performed worse in our case. The heuristic with the worst performance was the smaller problem heuristic. It performed slightly worse than A* with horizontal heuristic.

3 Appendix

Color	Cell Type
White	Regular unblocked
Black	Blocked
Red	Hard to traverse
Blue	Unblocked highway
Magenta	Hard to traverse highway
Green	Path of algorithm
Yellow	Fringe