Phase 1 Report

Matthew Ya & Aleece Randall

February 5, 2017

1 Quick How-to for Interface and Images

Colors:
- Blocked: Black
- Regular unblocked: White
- Hard to traverse: Orange
- Regular unblocked with a highway: Cyan
- Hard to traverse with a highway: Blue
- Start: Lime Green
- Path: Purple
- End: Red

2 Implementation of Algorithms

We designed an abstract class called Search and 3 concrete classes UniformCostSearch, AStarSearch, and WeightAStarSearch which implemented the abstract class. Each concrete class shares the findPath() and findSuccessorSet() methods to traverse the map and expand the fringe. The abstract class Search has two abstract methods updateVertex() and setupFringe() that are implemented and defined by the concrete class. The reason for this is that each Search algorithm has its own way of calculating the value of h(n), the value of f(n) and setting up the fringe.

3 Optimization

The data structures used to implement the algorithms were chosen to achieve the best time and space complexities. The fringe or open list is a priority queue that is implemented using a binary heap. This data structure provides worst case complexities of O(log N) for search, inserts, and delete and O(1) for removal of the cell with the lowest value. The successors of each node are stored in a HashSet because of its O(1) insert. A HashSet was also chosen for the closed list after time/space complexity tradeoff. The HashSet has an average case search time of O(1) but worst case time O(n). The HashSet was chosen over a 2D boolean array due to the array's higher space requirement. Although, the Boolean array would yield a better worst case search time of O(1) , the space complexity is always the total number of nodes in the graph. On average, the HashSet is a much smaller value. The HashSet and Priority Queue were initialized to a higher but reasonable initial capacity in an attempt to reduce possible rehashing.

4 Heuritistics

There are several possible ways to calculate the distance between the start node and goal node. Let dx represent the horizontal distance and let dy represent the vertical distance between the start node and goal node. Let D represent the vertical/horizontal cost and D2 represent diagonal cost. The costs used in the following heuristics focus on regular cells to try to avoid creating an inadmissible heuristic.

4.1 Euclidean Distance Formula

$$h = D2 * \sqrt{dx * dx + dy * dy} \tag{1}$$

Pro: considers diagonals
Con: computationally expensive, only considers a straight line

Using the diagonal cost, this function fails to take into account the faster speeds of the highways. Using an average of the highway horizontal/ vertical costs and unblocked cell diagonal s cost, the heuristic tries to consider the fact that an L-shaped movement on a highway is less costly than moving on a diagonal. We consider this to be the worst heuriistic.

4.2 Manhattan Distance Formula

$$h = D * (dx + dy) \tag{2}$$

Pro: computationally inexpensive
Con: does not consider diagonal movement

This formula considers the faster movements of highways. If the optimal path does not make use of the highways, this heuristic is not close to the actual cost. This heuristic is admissible in most cases, see below for exceptions.

4.3 Diagonal Distance Formula

$$h = D * (dx + dy) + (D2 - 2 * D2) * min(dx, dy) \tag{3}$$

Computes the number of steps if a diagonal is not taken and adds the minimum diagonal steps. The cost of the minimum diagonal steps is the net cost of not moving vertical/ horizontal. The equation tries to take into account both directions of travel. Considering all directions, highways for vertical/horizontal and regular cells for diagonal (extremely low probability for diagonal highway movement), might provide a good estimate that is admissible in most cases. However, using all the costs for a highway, this is the best optimal heuristic. It largely underestimates most paths but holds up for rare cases such where an optimal path is diagonal movements through parallel highways.

5 Experimental Results from 50 benchmarks

5.1 Heuristic 1

$$h = \sqrt{2} * min(dx, dy) + max(dx, dy) - min(dx, dy) \tag{4}$$

5.1.1 A* Search Experimental Results

Total average run time: 2.88ms
Total average path length: 153
Total average nodes expanded: 1202
Total average memory used: 476.64625KB

5.1.2 Weighted A* Search Experimental Results using weight 0.75

Total average run time: 0.0ms
Total average path length: 125
Total average nodes expanded: 129
Total average memory used: 113.72140625KB

5.1.3 Weighted A* Search Experimental Results using weight 1.25

Total average run time: 0.32ms
Total average path length: 132
Total average nodes expanded: 361
Total average memory used: 212.0971875KB

5.2 Heuristic 2

$$h = 0.25 * (dx + dy) \tag{5}$$

5.2.1 A* Search Experimental Results

Total average run time: 14.38ms
Total average path length: 180
Total average nodes expanded: 8740
Total average memory used: 3398.65203125KB

5.2.2 Weighted A* Search Experimental Results using weight 1.25

Total average run time: 10.64ms
Total average path length: 178
Total average nodes expanded: 7518
Total average memory used: 2762.37875KB

5.2.3 Weighted A* Search Experimental Results using weight 2.00

Total average run time: 6.1ms
Total average path length: 168
Total average nodes expanded: 4541
Total average memory used: 1510.2809375KB

5.3 Heuristic 3

$$h = ((0.25 + \sqrt{2})/2) * \sqrt{dx * dx + dy * dy} \tag{6}$$

5.3.1 A* Search Experimental Results

Total average run time: 7.72ms
Total average path length: 162
Total average nodes expanded: 2957
Total average memory used: 954.09546875KB

5.3.2 Weighted A* Search Experimental Results using weight 1.25

Total average run time: 2.18ms
Total average path length: 154
Total average nodes expanded: 1413
Total average memory used: 424.416875KB

5.3.3 Weighted A* Search Experimental Results using weight 2.00

Total average run time: 0.0ms
Total average path length: 126
Total average nodes expanded: 149
Total average memory used: 106.0834375KB

5.4 Heuristic 4

$$h = 0.25 * (dx + dy) - (\sqrt{2} - 2 * 1) * min(dx, dy) \tag{7}$$

5.4.1 A* Search Experimental Results

Total average run time: 44.98ms
Total average path length: 171
Total average nodes expanded: 5747
Total average memory used: 2441.0371875KB

5.4.2 Weighted A* Search Experimental Results using weight 1.25

Total average run time: 29.2ms
Total average path length: 164
Total average nodes expanded: 4388
Total average memory used: 1913.21125KB

5.4.3 Weighted A* Search Experimental Results using weight 2.00

Total average run time: 15.16ms
Total average path length: 141
Total average nodes expanded: 2753
Total average memory used: 956.2090625KB

5.5 Heuristic 5

$$h = \sqrt{2} * \sqrt{dx * dx + dy * dy} \tag{8}$$

5.5.1 A* Search Experimental Results

Total average run time: 0.14ms
Total average path length: 134
Total average nodes expanded: 223
Total average memory used: 158.99515625KB

5.5.2 Weighted A* Search Experimental Results using weight 1.25

Total average run time: 0.0ms
Total average path length: 126
Total average nodes expanded: 137
Total average memory used: 0.0KB

5.5.3 Weighted A* Search Experimental Results using weight 2.00

Total average run time: 0.0ms
Total average path length: 123
Total average nodes expanded: 123
Total average memory used: 0.0KB

5.6 Heuristic 6

$$h = .25 * (dx + dy) + (\sqrt{2} * .25 - 2 * .25) * min(dx, dy); \tag{9}$$

5.6.1 A* Search Experimental Results

Total average run time: 12.22ms
Total average path length: 180
Total average nodes expanded: 7899
Total average memory used: 3035.316875KB

5.6.2 Weighted A* Search Experimental Results using weight 1.25

Total average run time: 10.4ms
Total average path length: 175
Total average nodes expanded: 6556
Total average memory used: 1912.20203125KB

5.6.3 Weighted A* Search Experimental Results using weight 2.00

Total average run time: 4.56ms
Total average path length: 159
Total average nodes expanded: 3693
Total average memory used: 1062.43KB

5.7 Results from UniformCost Search

Total average run time: 20.7ms
Total average path length: 180
Total average nodes expanded: 12636
Total average memory used: 5370.14921875KB

6 Discussion

Overall, it's obvious that the most expensive search algorithm is the non-heuristic Uniform Cost Search algorithm. This algorithm has the highest average run time, nodes expanded, and memory used. The average performance of the A* search algorithm sits between the average performance of the Uniform Cost search algorithm and the average of the Weighted A* algorithm. For Weighted A* search algorithm, we noticed that the algorithm does significantly better in terms of memory and run time when the weight is set to 2 compared to when the weight is set to 1.25.

Out of the 5 tested heuristics, heuristic 5 had a positive influence in the behavior of the A* search algorithms. The heuristic with the worst result was heuristic 2. It seems that heuristic functions that come closest to predicting the actual path of two points will have a positive influence of the algorithm. Heuristic 5 is the least conservative heuristic function in regards to cost. Heuristic 5 provided the best computational performance but grossly overestimates the optimal cost.

In contrast, Heuristic 6 is an admissible/consistent function and a drastically worst performance. From these results, we can conclude that the more weight/value that originates from the heuristic leads to a better computational performance. The higher the heuristic values, the more influence it has in directing the algorithm. Heuristic 6 provides a fairly low estimate of the true optimal cost. The Weighted A* at 1.25 increased the computational performance without sacrificing the optimal cost nearly all of the trials. The Weighted A* at 2.00 more accuracy is lost. Heuristic 3 provides lower overestimates compared to Heuristic 5. As a result, the optimal path deviated a lot less than Heuristic 5 despite having the same Euclidean Distance Formula basis. The simulations demonstrate a tradeoff between computational performance and optimality.