CS440: Introduction to Artificial Intelligence Spring 2017

**Assignment 1:**

*Heuristic Search Using Information from Many Heuristics*

**Team:**

Eric Cajuste and Thurgood Kilper

**Phase 1 Report**

**A). Map Interface**

We used Java as the platform for visualizing the generated test maps. In addition, the map displays nodes that are in the open and closed lists while the algorithm operates.

Blocked squares = black

Weighted squares = gray

Unblocked squares = white

Highway squares = blue

Start/goal/path = red

closed node = light purple

fringe node = dark purple

**B). Abstract Heuristic Algorithm Implementation**

We define HeuristicAlgorithm.java as an abstract class which contains the base implementation of the A\* algorithm so that we may instantiate three different versions of it. UniformSearch.java and AStar.java each extend the HeuristicAlgorithm abstract class. The pathfinding algorithm is inherited from HeuristicAlgorithm.java, which means that the inheriting classes only need to handle the calculation of f and h values for checked nodes. In UniformSearch, h is ignored, and f=g. For A\*, f=g+h, and h is determined by a heuristic function. WeightedAStar.java extends Astar.java, and simply multiplies h by a weight value.

**C). Algorithm Optimization**

The implementation of the pathfinding algorithm is handled by HeuristicAlgorithm.java. It follows an optimized implementation close to the pseudocode for A\* detailed in the Assignment Sheet. The getNeighbors method is used to populate succ(s), and the cost method determines c(s, s'). The updateVertex method calls on the abstract methods hOfNeighbor and fOfNeighbor, which are implemented by the extending classes to determine the h and f values of the updated vertex. The closed list is stored in a HashSet, which allows closed membership to be checked in constant time.

**D). Heuristic Considerations**

The optimal heuristic for an 8-way grid problem is the euclidean distance, which calculates the straight line distance from a node to the goal. We use h = sqrt[(x1-x2)^2 + (y1-y2)^2], and multiply by 0.25, the smallest cost of moving to an adjacent node, in order for the heuristic to be admissible.

Other considered heuristics are the manhattan distance, which counts the minimum number of nodes between a node and the goal. We used a version that considers 8 way movement and one that only considers 4 way movement. The 8 way heuristic is a good alternative to euclidean distance, while the 4 way heuristic takes longer to expand due to ignoring diagonals.

One heuristic is to use a constant h = negative. It will always be admissible and find the optimal path, but will not help A\* in choosing which nodes to expand upon.

A very fast solution would be to use a euclidean heuristic that overestimates the remaining distance. For example, the earlier implementation without multiplying by 0.25. This gives A\* extreme bias for nodes closer to the goal, resulting in much faster pathfinding while losing out on optimality. This is how weighted A\* operates. Depending on the weight value, paths can be found much more quickly while still being very close to optimal.

**E). Benchmark Test Results**

In general, we observe that the total costs of uniform search and A\* are equal. This is because they use admissible heuristics and return the same optimal path. Weighted A\* will often return a higher total cost as its path is not necessarily optimal. Runtime wise, we observe that uniform search is the slowest, while weighted A\* is the fastest. In terms of cells traveled, uniform and A\* will provide the same optimal amount. Weighted A\* will often provide a path that is shorter in cell length, but not in cost. For expanded nodes, uniform expands the most, while weighted A\* expands the least. The difference between weighted A\* and A\* is much greater than the difference between uniform and A\*.

**F). Benchmark Result Discussion**

Uniform search does not use a heuristic. As a result, it expands without preference towards the goal. Compared to A\* which uses heuristics, it will expand many more nodes before reaching the goal, resulting in longer run times. A\* and weighted A\* both use admissible heuristics, however weighted A\* breaks admissibility by using a weight on h. In general, we observe that inadmissible heuristics provide the most efficient results while sacrificing optimality. Admissible heuristics are faster than not using heuristics while still finding an optimal path. Relatively, weighted A\* provides a great increase in performance while sacrificing very little in terms of optimality. If precise optimal paths are not a requirement, then weighted A\* is vastly preferred.