Finding a Path - By Will Gibson

Finding a path between two points is always an interesting process. For this assignment, we will explore a number of different algorithms which can be used to find these paths. Such algorithms include Breadth First Search, Dijkstra's Algorithm, and A* Heuristic algorithms. We will use this write-up to explain each algorithm, how it works in our problem, and which algorithm works best in this situation.

Breadth First Search

The first algorithm we worked with was the breadth first search. This algorithm took and looked at every single space on the grid, finding the path that covers the least squares. Normally, this algorithm just went down the edge of the board, and over to the side. It searched through 16276396 states, and produced the shortest number of locations in the grid (8188). Because it doesn't take into account the cost of each location, the path cost is the highest of any algorithm (44420), which gives it an average cost of over 5 (5.4) for each location it passes through. It takes 6 and a half minutes to run, one of the worst.

Depth First Search

One of the simplest, and yet one of the most well optimized search algorithms of the lot, the depth first search looks at the locations right around the current location, and then moves to whichever one is the deepest into the grid. All it does is go straight down, avoid water, and then hit the bottom and go straight over to the goal, avoiding water. It runs in just about a second, but produces some of the worst results. A path length of 8466 and a path cost of 41528 puts it second to last on the average cost per location, at 4.9 (the lower the better).

Least Cost Path

This algorithm finds the single best path through the grid, and it does it in a fairly impressive amount of time. It takes the cost of each node, and works backwards using a slightly simplified version of dijkstra's algorithm[1] to find the path through the grid with the lowest cost. That cost is 18387, and at a runtime of 5 minutes 50 seconds, it is one of the fastest algorithms. It is also the most efficient path in terms of average cost, at 2.1999. Least cost path done this way is the most efficient algorithm, to find the best path in the least amount of time.

[1] https://www.programiz.com/dsa/dijkstra-algorithm

Manhattan Distance

This is our first heuristic algorithm. This algorithm takes into account the cost of the path, but also cuts options out by pruning options that take the path away from a line running through the middle of the grid. It runs just barely worse than the least cost path with a length of 8360, and a cost of 18393. The average cost for this algorithm is barely higher at 2.2001. The only situation where this ends up being a bad algorithm comparatively is if there was a large lake in the center which caused the line to backtrack around it.

Manhattan Distance (Weighted)

This algorithm works the same as the regular Manhattan algorithm, but with just a slight tweak to add a weight to the distance. This weight is 1.2, and it gives more importance to finding a path through the middle, not necessarily to cost. This algorithm presents a shorter path (8332 locations) but with a slightly worse cost (18428). It does take the longest of any algorithm, at just over 11 minutes.

Comparing Algorithms.

Algorithm	States Visited	Path Length	Cost	Time (Seconds)
Breadth First Search	16276396	8188	44420	636.3227
Depth First	8487	8466	41528	0.9153
Least Cost Path	16276371	8358	18387	591.3304
Manhaten Distance	16276153	8360	18393	584.0815
Manhaten Weights	16276109	8332	18428	667.2303

Between everything, it is clear, if you just want a path, depth first search will give it to you real quick. If, however, you want the best path, things get a bit more complicated. Breadth first is out immediately, coming in last or second to last in every metric. The final three algorithms are incredibly close to each other, and it comes down to exactly which algorithm you want.

Algorithm	Cost/Time	Cost/Length	States/time
Least Cost Path	31.09429179	2.199928212	27525.00294
Manhaten Distance	31.49046837	2.200119617	27866.23613
Manhaten Weights	27.61864981	2.211713874	24393.53998

.

Least cost path is a wonderful algorithm, and though it takes a bit of extra time, it produces the best number in our most important category of cost/length. It minimizes cost real well, while also doing it in a good amount of time. Both versions of the Manhattan algorithm are similarly good, and the main difference will be on what the middle of the map looks like, and how much the Manhattan algorithm will have to circumvent.

Combining statistics.

If we combine some statistics to find a winning algorithm, we can take average cost and multiply it by time, giving us a good heuristic to measure the overall success of an algorithm. This metric shows that Manhattan distance (1285) is just over 1% better than least cost algorithm (1301), and the weighted Manhattan algorithm (1476) slows things down a bit, coming in about 14% worse than the regular Manhattan.