Project 1: Solving Mazes with A*

Paul Beggs
GitHub Link

November 13, 2025

1 Heuristic Descriptions & Implementations

1.1 Manhattan Distance Only

The Manhattan heuristic calculates the Manhattan distance between the current location of the `MazeExplorer` and the end location of the maze. If we let n_1 and n_2 be the coordinates of the current location and e_1 and e_2 be the coordinates of the end location, then the Manhattan distance is calculated as $md = |n_1 - e_1| + |n_2 - e_2|$. This implementation is admissible because we must make at least md moves to get the goal. Thus, the shortest path will never be larger than the Manhattan distance, as the actual path will certainly have more than moves due to the walls in the maze.

```java
public class Manhattan implements ToIntFunction<MazeExplorer> {
    @Override
    public int applyAsInt(MazeExplorer value) {
        return value.getLocation().getManhattanDist(value.getM().getEnd());
    }
}
```

Listing 1: Manhattan Distance Heuristic

1.2 Maximum Distance

The maximum distance heuristic works by getting a set of all treasures on the map that have not been collected yet, and by initializing an integer `maxDist` value that is just the Manhattan distance from the `MazeExplorer` to the exit. Then, for each remaining treasure in the maze, it determines if that treasure's distance is larger than the any other treasure, or the exit, and then returns that distance.

To show this heuristic is monotonic, we can start by looking at if $h(\text{goal}) = 0$. If we are at the goal state, then we have no remaining treasures, and $h(\text{goal})$ returns the Manhattan distance to the end of the goal, which is 0 (this reasoning also ensures that in any other case when there are no treasures left, $h(n)$ is monotonic). Now, take the case of the `MazeExplorer` moving, and landing on a treasure T. Before landing (at state n_1), $h(n_1)$ is either the distance to T (which is 1, since we are right next to it) or the distance to some other, further target. After landing (at state n_2), that treasure is removed from the list, and the new heuristic $h(n_2)$ is the maximum distance from our new location to all remaining targets. Now, because we only took one step, the maximum distance to all other targets cannot have dropped by more than 1. If T was the furthest target (meaning $h(n_1) = 1$), the new value $h(n_2)$ will be 0 or greater. So, in all scenarios, the heuristic value never drops by more than 1, so the condition $h(n_1) - h(n_2) \leq 1$ holds. Therefore, the maximum distance heuristic is monotonic.

```java
public class MaxDist implements ToIntFunction<MazeExplorer> {
    @Override
    public int applyAsInt(MazeExplorer value) {
        Pos currentLocation = value.getLocation();

        Set<Pos> allTreasures = value.getAllTreasureFromMaze();
        Set<Pos> foundTreasures = value.getAllTreasureFound();

        Set<Pos> remainingTreasures = new HashSet<>(allTreasures);
        remainingTreasures.removeAll(foundTreasures);

        int maxDist =
         ↪  currentLocation.getManhattanDist(value.getM().getEnd());

        for (Pos treasure : remainingTreasures) {
            int distToTreasure = currentLocation.getManhattanDist(treasure);
            if (distToTreasure > maxDist) {
                maxDist = distToTreasure;
            }
        }
        return maxDist;
    }
}
```

Listing 2: Maximum Distance Heuristic

1.3 Counting Treasures

This heuristic works by simply counting all the treasures in the maze. If we don't have any treasures remaining, then we default to using the Manhattan distance, which is monotonic. If we do have treasures remaining, then we return the number of treasures. This ensures that the difference between $h(n_1)$ and $h(n_2)$ will never be larger than 1 because we can only collect one treasure per move. Thus, the formula for monotonicity $h(n_1) - h(n_2) \leq 1$ is always true, and this heuristic is monotonic.

```java
public class CountTreasures implements ToIntFunction<MazeExplorer>  {
    @Override
    public int applyAsInt(MazeExplorer value) {
        Set<Pos> allTreasures = value.getAllTreasureFromMaze();
        Set<Pos> foundTreasures = value.getAllTreasureFound();
        Set<Pos> remainingTreasures = new HashSet<>(allTreasures);
        remainingTreasures.removeAll(foundTreasures);
        if (remainingTreasures.isEmpty()) {
            return
             ↪  value.getLocation().getManhattanDist(value.getM().getEnd());
        } else {
            return remainingTreasures.size();
        }
    }
}
```

Listing 3: Counting Treasures Heuristic

1.4 Overly Confident

The overly confident heuristic is just the Manhattan distance, but we multiply the value returned by 3, and add 5. This fails the criteria for monotonicity because when we are at the goal, our state will be $h(\text{goal}) = 5$, when it should be $h(\text{goal}) = 0$.

```java
public class OverlyConfident implements ToIntFunction<MazeExplorer> {
    @Override
    public int applyAsInt(MazeExplorer value) {
        int manhattanDistance =
         ↪  value.getLocation().getManhattanDist(value.getM().getEnd());
        return manhattanDistance * 3 + 5;
    }
}
```

Listing 4: Overly Confident Heuristic

2 Experimentation Analysis

2.1 Heuristics Performance

For each maze, you can see that the heuristic that consistently has the lowest branch factor and needs to explore the lowest amount of nodes is the maximum distance heuristic. While each monotonic heuristic had the same solution length (as would be expected by a monotonic heuristic by definition), there were variations among the amount of nodes searched, and the branching factors for each. As expected, breadth first search performed the worst among the monotonic heuristics, as it consistently explored the most nodes, and had either the same branching factor, or a higher one (this effect is most prominently seen in Table 5). In a similar vein, the count treasures heuristic also did relatively poorly (when compared to maximum distance). We see that it explores almost as many nodes as breadth first search, but this is expected as it only updates when we reach a treasure, so it doesn't tell us which way to go.

The most surprising heuristic was the one that only employed the Manhattan distance. Given its relative simplicity, it performed well and explored relatively few nodes in comparison to the other heuristics. Looking at the only non-monotonic heuristic, `OverlyConfident`, we see that even when the Manhattan distance is the only computation employed, it can go awry if it is arbitrarily scaled, making the heuristic non-admissible. Furthermore, this dubious scaling essentially sent the search on a wild goose chase, which led to the most nodes explored, and the greatest search depth in nearly every test case.

2.2 Maze Design

Depending on how the maze was set up, there can be a lot of variation in how many nodes we explore, and the branching factor. When controlling for only the dimensions of the maze, every heuristic was able to solve mazes up to 2000×2000 easily, as the branching factor stayed constant at 1.0 (anything higher than 2000×2000 would cause the GUI to lag too much). Then, when controlling mainly for perfection, it appears that the solvers also succeeded with relative ease. In general, it appears that the number of treasures is what has the largest effect upon the branching factor. As we can see that when we compare the maze designs, Maze 3 had the highest perfection, the smallest size, and the most treasures, supporting this conclusion.

Appendix A: Maze Data

Maze 1

Heuristic	b*	Depth	Nodes	Sol. Len.
Breadth First	1.04	238	1118685	239
Manhattan Only	1.04	239	785921	239
Max. Distance	1.04	238	464355	239
Count Treasures	1.04	238	1086557	239
Overly Confident	1.04	289	15603978	269

Table 1: Heuristic Results

Settings	Values
Treasures	5
Width	100
Height	100
Perfection	25%

Table 2: Maze 1 Settings

Maze 2

Heuristic	b*	Depth	Nodes	Sol. Len.
Breadth First	1.07	181	9117955	181
Manhattan Only	1.07	181	3716463	181
Max. Distance	1.06	180	1721702	181
Count Treasures	1.07	180	8949824	181
Overly Confident	1.06	222	8588447	213

Table 3: Heuristic Results

Settings	Values
Treasures	10
Width	50
Height	50
Perfection	0%

Table 4: Maze 2 Settings

Maze 3

Heuristic	b*	Depth	Nodes	Sol. Len.
Breadth First	1.14	107	10646594	107
Manhattan Only	1.13	107	4469867	107
Max. Distance	1.12	106	1944758	107
Count Treasures	1.14	106	9133748	107
Overly Confident	1.11	133	10817669	113

Table 5: Heuristic Results

Settings	Values
Treasures	13
Width	25
Height	25
Perfection	75%

Table 6: Maze 3 Settings