Project 1: Solving Mazes with A*

Paul Beggs
GitHub Link

November 10, 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.

```
1 public class Manhattan implements ToIntFunction<MazeExplorer> {
2     @Override
3     public int applyAsInt(MazeExplorer value) {
4         return value.getLocation().getManhattanDist(value.getM().getEnd());
5     }
6 }
```

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 that this heuristic is monotonic, we can start by looking at if $h(\text{goal}) = 0$. We know that this is true because at the goal state, we have no remaining treasures, and `maxDist` is

just the Manhattan distance to the end of the goal, so $h(\text{goal})$ returns 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.

```
1  public class MaxDist implements ToIntFunction<MazeExplorer> {
2      @Override
3      public int applyAsInt(MazeExplorer value) {
4          Pos currentLocation = value.getLocation();
5
6          Set<Pos> allTreasures = value.getAllTreasureFromMaze();
7          Set<Pos> foundTreasures = value.getAllTreasureFound();
8
9          Set<Pos> remainingTreasures = new HashSet<>(allTreasures);
10         remainingTreasures.removeAll(foundTreasures);
11
12         int maxDist =
           ↪  currentLocation.getManhattanDist(value.getM().getEnd());
13
14         for (Pos treasure : remainingTreasures) {
15             int distToTreasure = currentLocation.getManhattanDist(treasure);
16             if (distToTreasure > maxDist) {
17                 maxDist = distToTreasure;
18             }
19         }
20         return maxDist;
21     }
22 }
```

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 true, and this heuristic is valid.

```
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 $h(\text{goal}) = 0$.

```
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 and Analysis