Project I Report for COM S 4720 Spring 2025

Logan Roe and Zachary Foote

***Abstract*— The document shows the implementation of a heuristic function used to determine optimal paths on a 30x30 grid with obstacles, where the path starts at a designated start point and has a designated goal point to end at. We will first explain the implementation method that we chose for this problem, and explain a few points on our program. Then, we will prove the optimality of the path returned using proven facts about heuristic functions. The document will conclude with a summary of the project and references.**

I. INTRODUCTION

The presented problem involved a 30x30 grid with obstacles that could be anywhere within the grid. A starting coordinate and goal coordinate was also presented along with the grid. The goal was to arrive at the goal coordinate, from the starting coordinate, in a continuous, optimal path. This could be done using orthogonal and/or diagonal movements, all costing the same value of one.

In order to achieve this goal, a combination of past costs and a heuristic function was used to calculate next steps at any given coordinate. Using the Chebyshev function for the heuristic function, an estimated move could be made to determine the optimal path. With this approach, an optimal, consistent path is able to be found for each scenario within the given problem.

II. IMPLEMENTATION OF CODE

A. General Overview

The Python function used, named *find_path(grid, start, end)*, has three input parameters, as shown by the function definition, named *grid*, *start*, and *end*. Parameter explanations:

- grid: List of lists showcasing the grid environment.
  - 0: Represents a walkable cell.
  - 1: Represents an obstacle.
- start (Tuple[int, int]): The (row, column) coordinates of the starting position.
- end (Tuple[int, int]): The (row, column) coordinates of the ending (goal) position.

B. Algorithm Functionality

In order to use the provided information and find the optimal path, as the algorithm is intended to do, a priority queue, namely heapq in Python, is used to ensure the next most optimal step is used at each move. For the priority queue to work as intended, it is expected to sort by up to three values, depending on if there are ties or not.

For sorting, firstly, it will sort based on the f(n) cost, i.e. the total estimated cost which includes g(n), the cost up to this point, and h(n), the estimated remaining cost. If a tie occurs at the f(n) level, then g(n) will be considered, which is purely just the cost up to this point. Finally, if a tie occurs at both the f(n) and g(n) levels, then the weight of the directions will be considered as we have applied a 0 weight to orthogonal movements and a weight of 1 to diagonal movements. This weighting system will result in prioritization of orthogonal movements, for a smoother path, but only if all other moves are of equal weight. This will **not** affect the optimality of the path.

To be considered in the priority queue in the first place, the given grid coordinate must meet the following criteria:

- Must be within the bounds of the grid. In this case, this would be between [0, 30] inclusive, in each direction.
- The grid value must be a 0 which indicates that this is a walkable cell and not an obstacle.
- One of the following two conditions must remain true:
  - The new coordinate must *not* be in the visited set of coordinates.
  - The newly calculated past value, g(n), must be strictly less than the g(n) cost already in the visited set for this coordinate.

If a coordinate has met all of these requirements, it will be added to the priority queue with a total of four values. The first value being the new f(n) which, as stated prior, is g(n), the past cost, plus h(n), the predicted future cost. In order to achieve this h(n), Chebyshev distance is used since it treats all directions at an equal cost of 1, as expected for this implementation. The optimality of this approach is discussed in Section III. For the second value, it is simply just the past cost, g(n), the third value being the weight of the given direction (0 for orthogonal & 1 for diagonal), and the final value being the (x, y) coordinate.

Finally, once the goal has been reached, the algorithm will back-trace it's steps using the parent set to generate the path that it determined to be optimal. This path will then be returned as the optimal path to take to arrive at the desired end point from the starting point, with respect to all obstacles and bounds.

III. PROOF OF OPTIMALITY

A. Proof of Admissibility

First we are going to determine whether our heuristic function is admissible. In order for a heuristic function to be admissible, the following formula must be met:

$$\forall n \in \mathcal{N},\ h(n) \leq h^*(n)$$

This essentially means that the distance estimated by our heuristic function must always be equal to or less than the

actual distance from the goal node. Due to the grid layout and the fact that diagonal directions are allowed at no additional cost, it can be concluded that the cost of an optimized path from node N to node G can never be less than the max between (Gx - Nx) and (Gy - Ny), where Gx and Gy are the x and y coordinates of the goal node, respectively, and Nx and Ny are the x and y coordinates of the current node, respectively. This is because the distance change that is lesser between the two could be covered by doing diagonal moves, which are able to close the distance between both the x and the y-axis. Therefore, the minimum threshold for the number of moves can be bounded by the Chebyshev's equation:

$$d_{\text{Chebyshev}} = \max\left(|x_2 - x_1|, |y_2 - y_1|\right)$$

We additionally chose Chebyshev's as our heuristic function, to estimate the remaining distance of each next move. Therefore, because our heuristic function is the same as the lower bound for our cost of an optimal path, we can conclude that h(n) is less than or equal to h*(n) for all values of n in the scope of the grid, and therefore the heuristic function is admissible.

B. Proof of Consistency

Next, we are going to determine whether our heuristic function is consistent. In order for a heuristic function to be consistent, the following formula must be met:

$$h(n) \leq c(n, n') + h(n') \quad \forall n \in \mathcal{N},\ n' \in R(n)$$

For our implementation, the cost between one node to its successor, c(n, n'), is always one. Additionally, there are three ways that the path can advance towards the goal: a vertical move (on the x-axis), a horizontal move (on the y-axis), or a diagonal move (both axes). In our code, our equation for h(n) follows Chebyshev's theorem:

$$d_{\text{Chebyshev}} = \max\left(|x_2 - x_1|, |y_2 - y_1|\right)$$

Stepping through the three ways that a node in the path can move, the first would be a vertical move. The lowest posible value that h(n') could have is h(n)-1, which would be in the scenario that node n's x-value is further away from the goal's x-value than node n's y-value is from the goal's y-value. This is because our heuristic function only cares about the max between these two values when determining remaining distance, so if the max value was the difference in x, and that was reduced by one, then we would get h(n)-1.

The second move would be a horizontal move. Using the same logic as above, we can conclude that again the lowest possible value for h(n') is h(n)-1, because in the scenario the node n's y-value is further from the goal's y-value than node n's x-value is from the goal's x-value, the value for h(n') would still only be one h(n)'s value reduced by 1.

The last possible move would be a diagonal move. The most significant impact this move could have would be to create a successor node that is both one space closer on the x-axis and one space closer on the y-axis. However, h(n') will still only equal h(n) - 1 because our equation for the heuristic function is only grabbing the change on the axis that is the farthest from the goal, and the change on that axis would just be one less than the previous value. Even if both axes (x and y) had the same distance between the current node and the goal, the heuristic function would grab either, as both will be equal to h(n) - 1 after diagonal move, because the path only moved once space in each direction.

Therefore, the only value that c(n, n') can be is one, and the minimum value that h(n') can equal is h(n) - 1. Therefore, plugging in, we have:

$$h(n) \leq 1 + h(n) - 1 \quad \forall n \in \mathcal{N},\ n' \in R(n)$$

The minimum value possible value for h(n') satisfies the equation, and therefore the equation will hold true for all nodes (n) with succesor n' in the scope of the grid, proving that the heuristic function is consistent.

C. Decision on Optimality

Therefore, because the heuristic function outlined above is both admissible and consistent, it will produce an optimal path, assuming an optimal path exists.

IV. CONCLUSIONS

In summary, to solve the presented problem for Project 1, i.e. finding the optimal path for any given 30x30 grid from start to end, as discussed, in detail, in Section II, an approach using the Chebyshev equation for the heuristic function was used. As proved in Section III, this approach provides an admissible, consistent, and therefore optimal solution to the problem, which will provide an optimal path in all cases where such a path exists.