Project I Report for COM S 4720 Spring 2025: Subtitle*

Noah Miller

***Abstract*— This paper will attempt to describe the searching algorithm written by the author in order to solve a path finding question in which the criteria used to define better algorithms are those that find the shortest path by number of steps, and the quickest algorithm to run. The finalized algorithm is an attempted implementation of A-star [1]. The method is created mostly from what was described in class, and what could be recalled at time of writing. The algorithm has room for improvement, but succeeds in most all cases.**

I. INTRODUCTION

Searching algorithms have been a question for computer scientists for a very long time. Searching algorithms are primarily used in attempts to solve mazes or to traverse trees, which can be thought of much the same. The criteria for a good searching algorithm is that which finds the optimal path, i.e. shortest path, and finds it quickly. Many undergraduate computer science students are required to learn and understand some of the most integral searching algorithms. These algorithms included Breadth First Search [2]. This algorithm has been around since 1959, with A-star breaking ground in 1968 [1]. Breadth First Search and other algorithms like it do not look forward, taking into account the goal. This means that in order to find the optimal path and be guaranteed of it, one must search every single node. This is not optimal in the criteria of the problem. As such A-star uses the future cost of a branch to decide whether or not to take it [1]. This means A-star more commonly finds the optimal path, and finds it quickly. When designed as intended it can be proved to find the optimal path.

II. IMPLEMENTATION

A. Pseudocode

There are some assumptions in this algorithm. These assumptions are that all movement costs are positive and that the goal is reachable from the starting point. Without these assumptions, there is no guarantee that the algorithm will find the shortest path. This algorithm states to take every point in the visited section that has moves to be made and check all moves to find the cheapest. The cheapest here is the cost of moving to that section plus an expected future cost. This cost is normally some kind of loss function, such as L2 or the distance equation, and allows A-star to utilize the future in determining the present move. The pseudocode as written does have some redundancy in it, leading to a very slow algorithm, but when implemented with optimizations it does perform up to speed.

Algorithm 1 Attempted Implementation

Input: grid, start, end
Output: path
Initialization :
visited = {}
reached = FALSE
Main Loop
while not reached **do**
  minCost = ∞
  move = a direction to move in
  for all point, p, in visited **do**
    directions = moves possible
    for all direction in directions **do**
      $p_{new} = p$ + direction
      if p_{new} in visited **then**
        continue
      end if
      g(p_{new}) = cost to get to p_{new}
      h(p_{new}) = distance to end from p_{new}
      f(p_{new}) = g(p_{new}) + h(p_{new})
      if f(p_{new}) $<$ minCost **then**
        minCost = f(p_{new})
        move = p_{new}
      end if
    end for
  end for
  if move = end **then**
    return path = path taken to move
  end if
  visited.add(move: (cost, path), where cost is the cost to move, and the path is the path taken to the move and the move taken
end while
return path

Some of the non optimal section of the algorithm is in the repeated computation of the cost of each movement. There is also redundancy in rechecking each direction for each point every single loop. These both will be addressed in actual implementation, although as it will be touched on later in the discussion, it has lead to some confusion.

B. Actual Implementation

The actual implementation follows the psuedocode closely, diverging in some key ways, some of which are beneficial and some of which seem to introduce errors. That is not to say they do not find a correct path which will be discussed in the results. The first implementation choice is that for ”visited” set, that is a hash set in which each point keeps

track of the cost to it and the path taken to it. This is to give an ease of which to get both values and use them later in the algorithm. This does take up more space in the process, but keeps the speed up and the ease of accessing information needed very near. The other implementation choice is to keep track of the best move from each point. This stores every top move choice in another hash set. It does not store all moves sorted, but rather only the top move. This does save on space but reduces the effectiveness. This will be discussed further in the ”Discussion” section.

III. Results

The implemented algorithm pulling from the concept of A-star is able to complete all examples. When running the ”main.py” supplied for the project, each example passes. There is also the question of speed. For an average of 10 runs over the 100 tests, the a-star adjacent algorithm ran for 239836443.4 nanoseconds, whereas Depth First Search (dfs) provided by the professor took 330706057.3 seconds. That is not the full story though, as the path taken also matters.

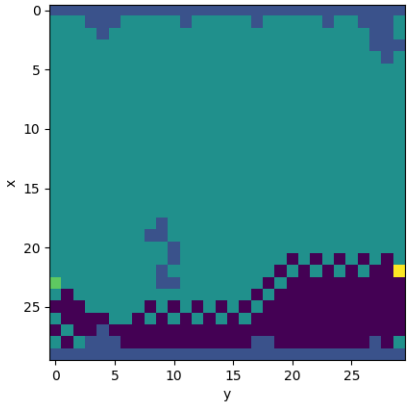

Fig. 1. DFS ran on test example with ID: 5

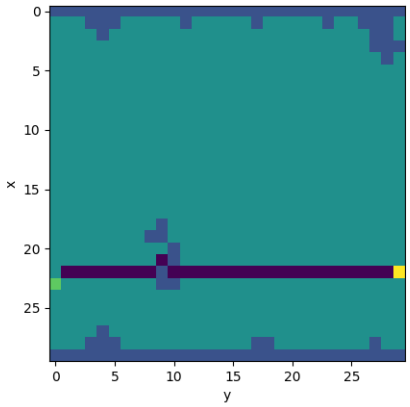

Fig. 2. A-star ran on test example with ID: 5

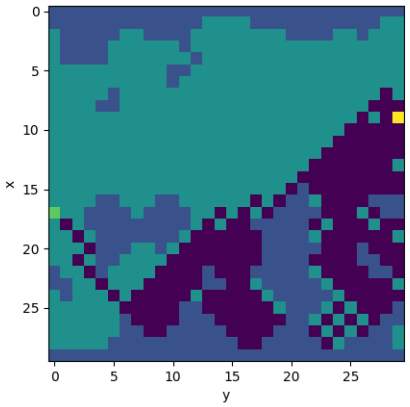

Fig. 3. DFS ran on test example with ID: 10

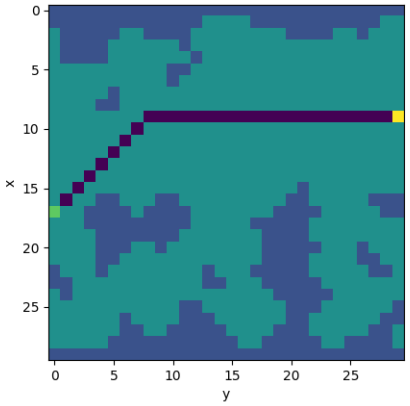

Fig. 4. A-star ran on test example with ID: 10

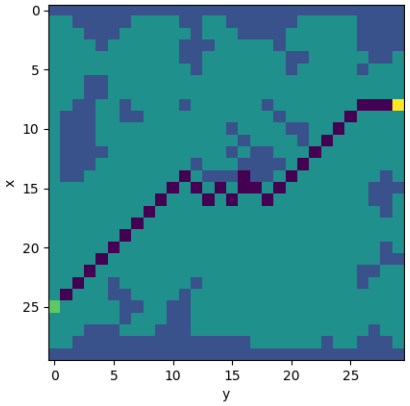

Fig. 5. A-star ran on test example with ID: 24

IV. Discussion

As can be seen comparing Figures 1 3, to Figures 2 and 4, the A-star algorithm takes a much shorter path to the goal. This is not always the case though as Figure 5 shows. It is my contention that this is how best moves from each point is stored. As such the algorithm takes an extra step after finding itself near a boundary. There were many attempts to ramify the issue, but each resulted in much worse bugs such that the algorithm no longer finished each task. As can be seen from the results as well, the modified A-star algorithm operates at a similar speed to the dfs algorithm provided by the professor. This means that in the pursuit of the lowest cost path there is not a large tradeoff for speed.

V. CONCLUSIONS

This report explains an implementation of A-star used for finding the best path in a maze. The assumptions for each maze are such that each move takes a positive amount to move, and that each goal can be reached from the starting point. This implementation seems to perform much better than the provided algorithm from the professor, although does seem to have a flaw or two. Further implementation and work is needed to implement an optimal solution that is guranteed to find the optimal path. The algorithm still does find a better path than the naive implementation of dfs, but can be improved upon.

ACKNOWLEDGMENT

We would like to acknowledge Professor Wang, as without his work and teaching this report and implementation would not be possible.

References

[1] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[2] E. F. Moore, "The shortest path through a maze," in *Proc. of the International Symposium on the Theory of Switching*. Harvard University Press, 1959, pp. 285–292.