University of Puerto Rico
Mayagüez Campus
Department of Electrical and Computer Engineering

Search

ICOM 5015 - 001D
Prof. J. Fernando Vega Rivero

David A. Castillo Martínez
Christian J. Perez Escobales
Ramón J. Rosario Recci
April 1, 2024

Abstract

This report delves into the significance and application of search algorithms in programming, particularly within the domain of artificial intelligence (AI). It begins by highlighting the prevalence of search algorithms in AI, where they are utilized to seek the best or most optimal solution within a set of possibilities. Additionally, it categorizes search algorithms into informed and uninformed categories, based on their utilization of heuristics. Two key exercises are then explored to exemplify the practical implementation of search algorithms: finding the shortest path between two points amidst polygonal obstacles and transporting missionaries and cannibals across a river. The report presents hypotheses and methodologies for each exercise, along with an in-depth analysis of the employed search algorithms' performances. Furthermore, it provides insights into the challenges posed by the exercises and the importance of systematic approaches in navigating them. Ultimately, the report underscores the indispensable role of search algorithms in programming, offering powerful tools for addressing complex problems across various domains of computer science and AI.

Introduction

Search algorithms are fundamental tools when it comes to coding, and are essential for programmers in various domains such as problem-solving, optimization, data structures and artificial intelligence. In the AI field, search algorithms are most commonly used to find the best or most optimal solution to a problem by exploring a set of possible solutions. [1] Additionally, searches can be divided into informed or uninformed searches. Informed search algorithms use heuristics to guide the search process and determine how close a state is to the goal. On the other hand, uninformed search algorithms have no additional information on the goal node, and reach the goal state from the start state differ only by the order and length of actions. [2] In this assignment, we use multiple search algorithms to solve two key exercises that are popular in the artificial intelligence field. The first exercise involves finding the shortest path between two points on a plane that has convex polygonal obstacles. For this exercise, our hypothesis states that Breadth First Search will be the most optimal search algorithm since it is commonly used to find the shortest path between two nodes in a graph. The second exercise involves finding a way to get three missionaries and three cannibals from one side of a river to the other with a boat that can hold one or two people, and without ever leaving a group of missionaries in one place outnumbered by the cannibals in that place. For this case, our hypothesis states that a search algorithm like Breadth First Search is optimal as long as we check for repeated states to not repeat any previous moves.

The first part of the assignment required finding the shortest path between two points on a plane that has convex polygonal obstacles as shown in Figure 1. In a state space consisting of all positions (x, y) in a plane, the number of states would be theoretically infinite, and so would be the number of paths, since there are infinitely many possible positions in a continuous plane. Assuming the state space is a grid the number of states would depend on the size of the grid. If the grid has dimensions MxN (M rows and N columns), then the number of states would be M * N. As for the number of paths to the goal, it depends on the specific layout of obstacles and the start and goal positions. In a grid-based representation, if we consider only movements to adjacent grid cells (up, down, left, right), the number of paths from the start to the goal would depend on the connectivity of the grid and the obstacles present. The shortest path between vertices is a straight line, therefore, if reaching the goal through a straight line is not possible, then one must traverse the obstacles that are found in between the vertices, which would cause this straight line to be warped to fit the vertices in between the start and end points.

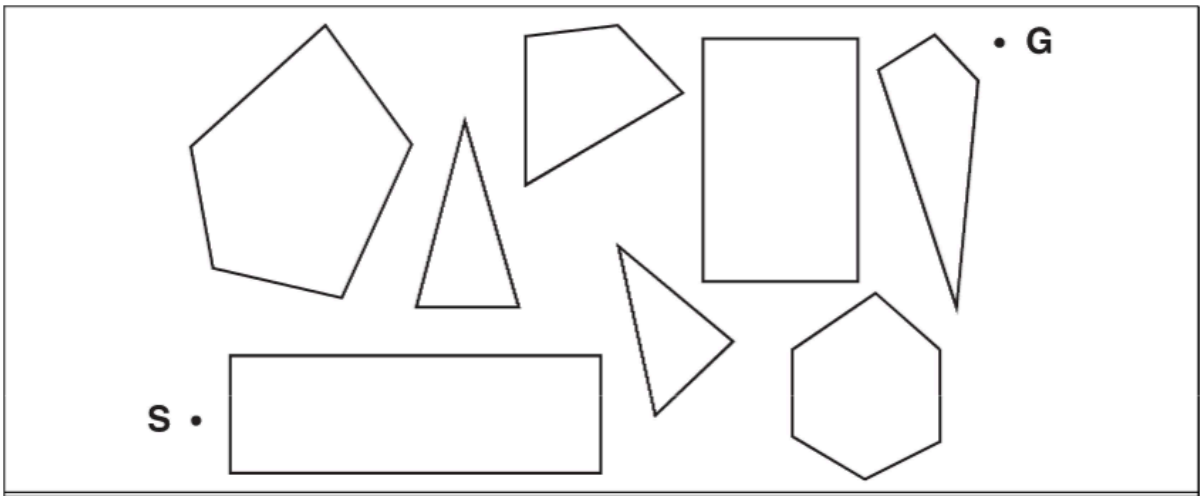


Figure 1: Example of environment with obstacles

To implement the search problem, we need to define the following functions:
First a successor function, this function takes a vertex as input and returns a set of dictionaries, each of which maps the current vertex to one of the vertices that can be reached in a straight line. This function should consider the neighboring vertices on the same polygon as well. The set of dictionaries could be obtained by considering all possible straight-line paths from the current vertex to reachable vertices. These dictionaries would represent the movements in terms of (dx, dy) from the current vertex to its neighbors. Second a heuristic function, this function calculates an estimate of the cost from the current state to the goal state. We can use the straight-line distance between the current state and the goal state as the heuristic function. We achieved these by creating our actions and h functions.

We utilized breadth-first search (BFS), depth-first search (DFS), and uniform-cost search (UCS) to solve the shortest path problem in this domain. BFS explores all possible paths from the start node outward, guaranteeing the shortest path but potentially requiring a lot of memory for large state spaces. DFS may find a path quickly but doesn't guarantee the shortest path and may get stuck exploring one branch of the graph. UCS is optimal and guarantees the shortest path, but its performance depends on the edge costs and the size of the state space. We measured the performance of each of these algorithms by counting the amount of vertices or nodes that were visited in total. Based on the characteristics of these algorithms and the nature of the problem domain, we observed that BFS and UCS performed well in finding the shortest path, with BFS being optimal. However, UCS could have been more optimal if the length between the nodes varied. In this exercise, the length between each adjacent node was 1; therefore, UCS did not result as the most optimal search algorithm. DFS might find a path quickly, but it's not guaranteed to be the shortest. Based on our results, DFS had the worst performance with the highest total number of vertices visited of 431, and found a path of 234 moves to reach the goal. Then, UCS obtained the second best performance, visiting a total of 426 vertices. Finally, BFS performed the best, visiting 419 vertices and finding the shortest path of 53 moves to reach the goal. All of these results were obtained from the output of the implemented code for this exercise, and can be observed in Figure 2.

```
Breadth First Search:
Goal: <Node B38>
Vertices in Path: 53
Total Visited Vertices: 419

Depth First Search:
Goal: <Node B38>
Vertices in Path: 234
Total Visited Vertices: 431

Uniform Cost Search:
Goal: <Node B38>
Total Visited Vertices: 426
```

Figure 2: Output Example for Exercise 3.9

The second part of the assignment focuses on solving the missionaries and cannibals problem optimally using breadth-first search (BFS) and addressing the challenges posed by its constraints and complexity. For our Initial State we have 3 missionaries, 3

cannibals, and the boat is on the left bank of the river. For our Goal State, all missionaries and cannibals are on the right bank of the river. Actions: Move one or two people (missionaries, cannibals, or both) from one bank to the other, but we do have constraints. At no point can the number of cannibals on either bank outnumber the number of missionaries. Additionally, the boat can only hold a maximum of two people.

One appropriate search algorithm that we used to solve this problem optimally is the breadth-first search (BFS) algorithm. Checking for repeated states is necessary to avoid entering infinite loops, as the search space is finite but can involve many steps. Here's a brief outline of the BFS implementation. Initialize a queue with the initial state. While the queue is not empty: Dequeue a state. Generate all possible successor states by applying valid actions. Enqueue valid successor states by ensuring there are never more cannibals than missionaries on either side of the river. If a successor state is the goal state, return the solution path. BFS ensures optimality by exploring all possible paths from the initial state outward, guaranteeing the shortest path to the goal. Figure 3 has the output of the implemented code for this exercise. By observing this figure, it is evident that the algorithm successfully completed the problem in 11 actions using breadth-first search (BFS), showcasing every state it traversed along the way.

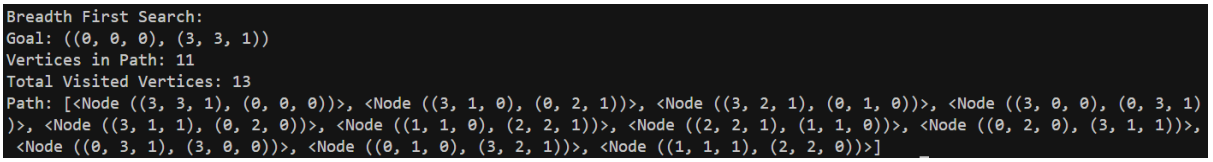


Figure 3: Output Example for Exercise 3.11

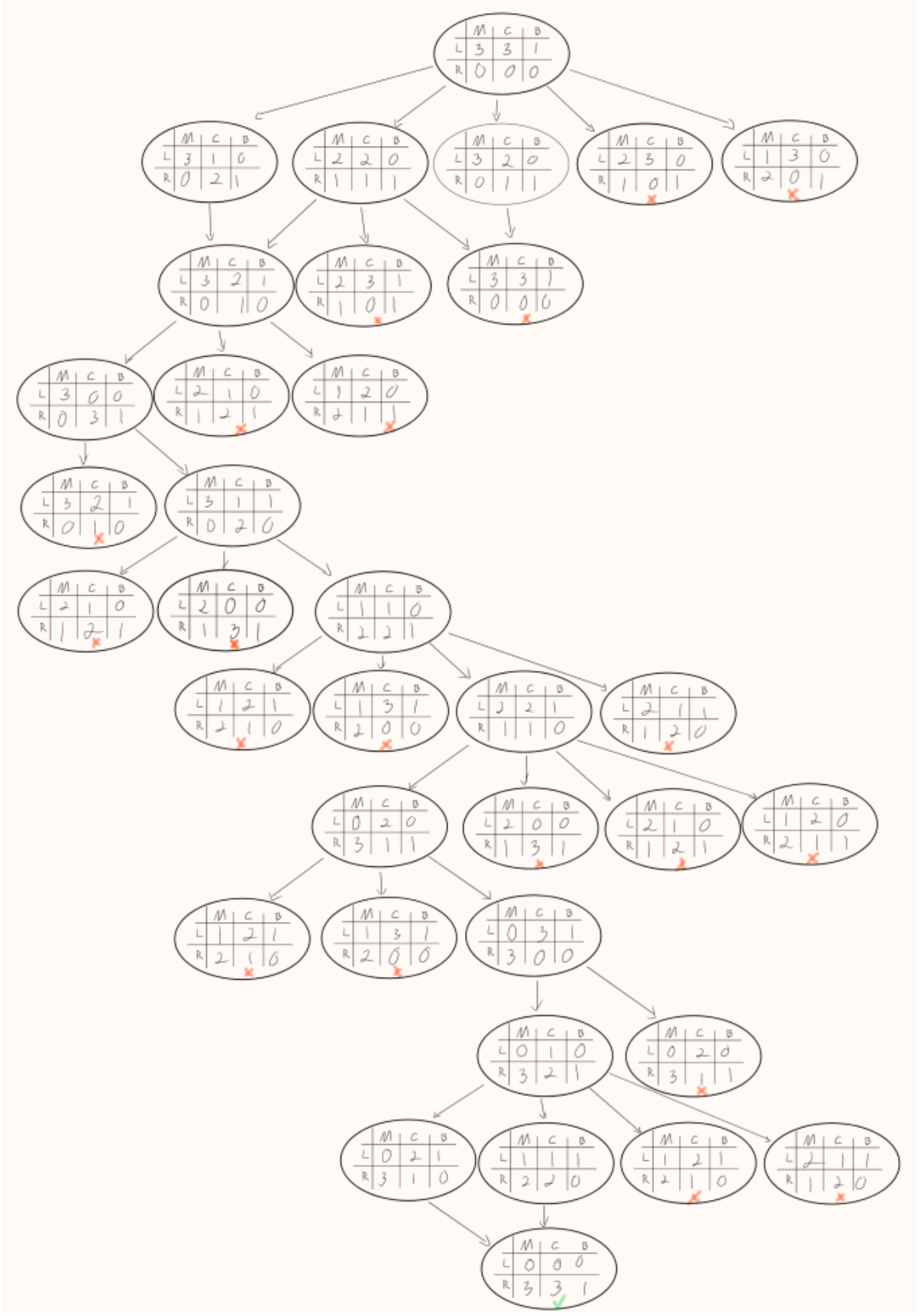
Despite the simple state space, people may have a hard time solving this puzzle due to several factors. The puzzle involves multiple constraints and considerations (e.g., boat capacity, cannibal-missionary ratio) that need to be satisfied at each step, requiring careful planning and logical thinking. There are many possible moves and combinations of moves, making it easy to get lost in the search space without a systematic approach. The puzzle requires forward thinking to anticipate the consequences of each move and ensure that no group is left outnumbered, which can be challenging for some individuals.

Conclusion

In summary, search algorithms are vital tools for programmers, enabling systematic problem-solving across diverse domains such as optimization and artificial intelligence. In this report we explored their application in two classic problems: finding the shortest path and transporting missionaries and cannibals across a river. For the shortest path problem, breadth-first search (BFS) emerged as the optimal algorithm, systematically exploring the state space. Similarly, BFS was effective in navigating the constraints of the missionaries and cannibals problem, ensuring an optimal solution while adhering to safety constraints. These results followed our hypothesis of BFS being the most optimal algorithm for the first exercise while also working in the second exercise as long as we don't repeat any of the previous states. Despite the simplicity of the state space, both problems posed challenges that required careful consideration of constraints and logical planning. Through the systematic application of search algorithms, programmers can address such challenges and achieve optimal outcomes in problem-solving tasks. In conclusion, search algorithms serve as invaluable assets for programmers, offering powerful methods to tackle complex problems and achieve optimal solutions in various domains of computer science and artificial intelligence.

Appendix

Appendix A: Diagram of the complete State Space



References

[1] “AI | Search Algorithms | CodeCademy,” *Codecademy*. https://www.codecademy.com/resources/docs/ai/search-algorithms

[2] GfG, “Difference between Informed and Uninformed Search in AI,” *GeeksforGeeks*, Feb. 16, 2023. https://www.geeksforgeeks.org/difference-between-informed-and-uninformed-search-in-ai/

[3] “aimacode,” *GitHub*. https://github.com/aimacode

[4] S. Russell and P. Norvig, “Artificial Intelligence: A Modern Approach”, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall, 2010.

Task Distribution

- Ramon J Rosario Recci - Collaborated on the analysis, interpretation of results, full state space diagram, collaborated writing the assignment's formal report.
- DAVID A CASTILLO-MARTINEZ - Implemented Python code for the exercises using the Aimacode repository, and collaborated on writing the assignment's formal report.
- CHRISTIAN J PEREZ-ESCOBALES - Collaborated on the analysis, interpretation, writing the assignments formal report, created the presentation, and edited the video presentation.