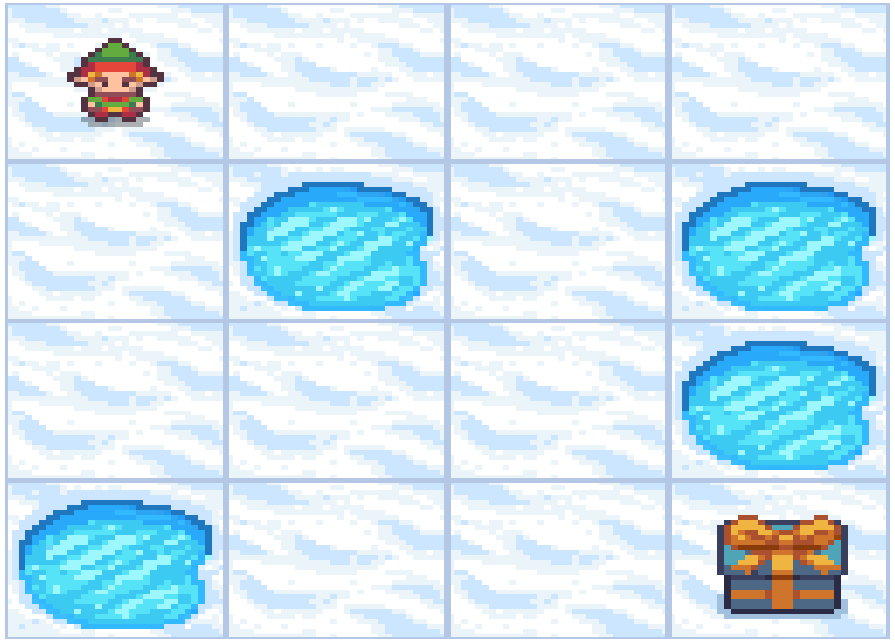
Solving Frozen Lake & Ant Maze Challenges with Branch and Bound & IDA*

Problem Statement

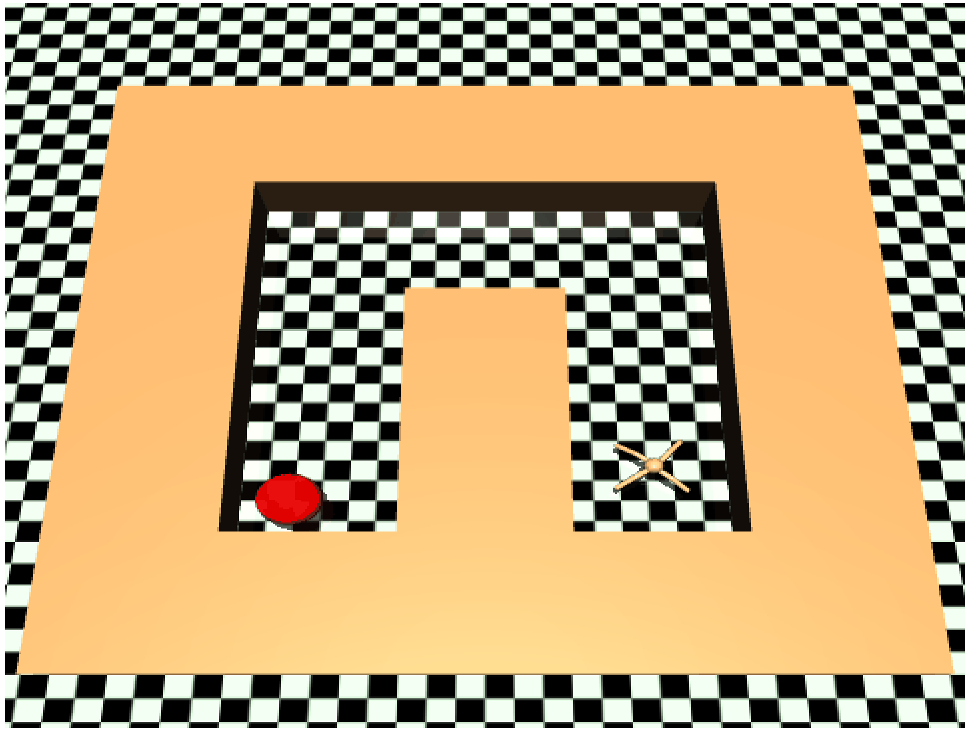
1. Frozen Lake

- The Frozen Lake problem is a classic reinforcement learning problem
- The agent has to navigate a 4x4 grid of tiles that represent either frozen ice (F) or holes (H).
- The agent starts at the top-left corner (S) and has to reach the bottom-right corner (G) without falling into any holes.
- The agent can move in four directions: left, right, up, or down.



2. Ant Maze

- The Ant Maze Problem involves navigating an ant from a starting point to a goal through a complex maze with multiple paths, obstacles, and potential dead ends.
- The maze consists of interconnected nodes (or grid cells) representing possible movement positions. Some paths may be blocked by walls or obstacles.
- The ant can move in predefined directions (e.g., up, down, left, right, or diagonally, depending on the maze design).
- The goal is to find the optimal path with the minimum number of steps or the least cost.



Implementation Details of Branch and Bound for Frozen Lake Problem

1. Node Representation

- Each state in the search space is represented as a Node object, containing:
  - position: The current (row, column) coordinate.
  - cost: The sum of the actual cost (g) and heuristic cost (h).
  - parent: A reference to the parent node (for path reconstruction).

2. Heuristic Function

- The Manhattan Distance heuristic is used
  - $h(n)=|x_1-x_2|+|y_1-y_2|$
  - This provides an estimate of the number of moves required to reach the goal.

3. Priority Queue (Min-Heap)

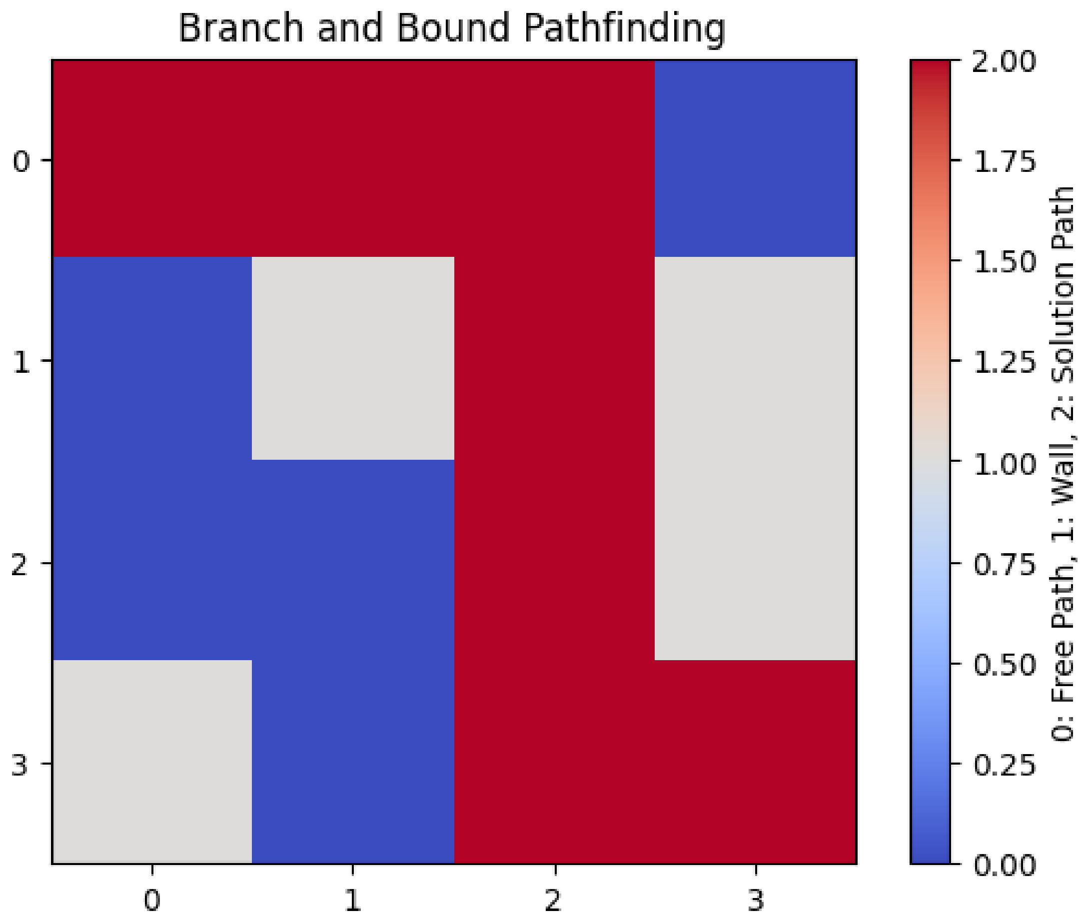
- A priority queue (min-heap) is used to always expand the node with the lowest cost = g(n) + h(n), ensuring an optimal path.

4. Branching (Expanding Nodes)
   - The algorithm explores all valid moves (Right, Down, Left, Up).
   - It avoids revisiting already explored positions.

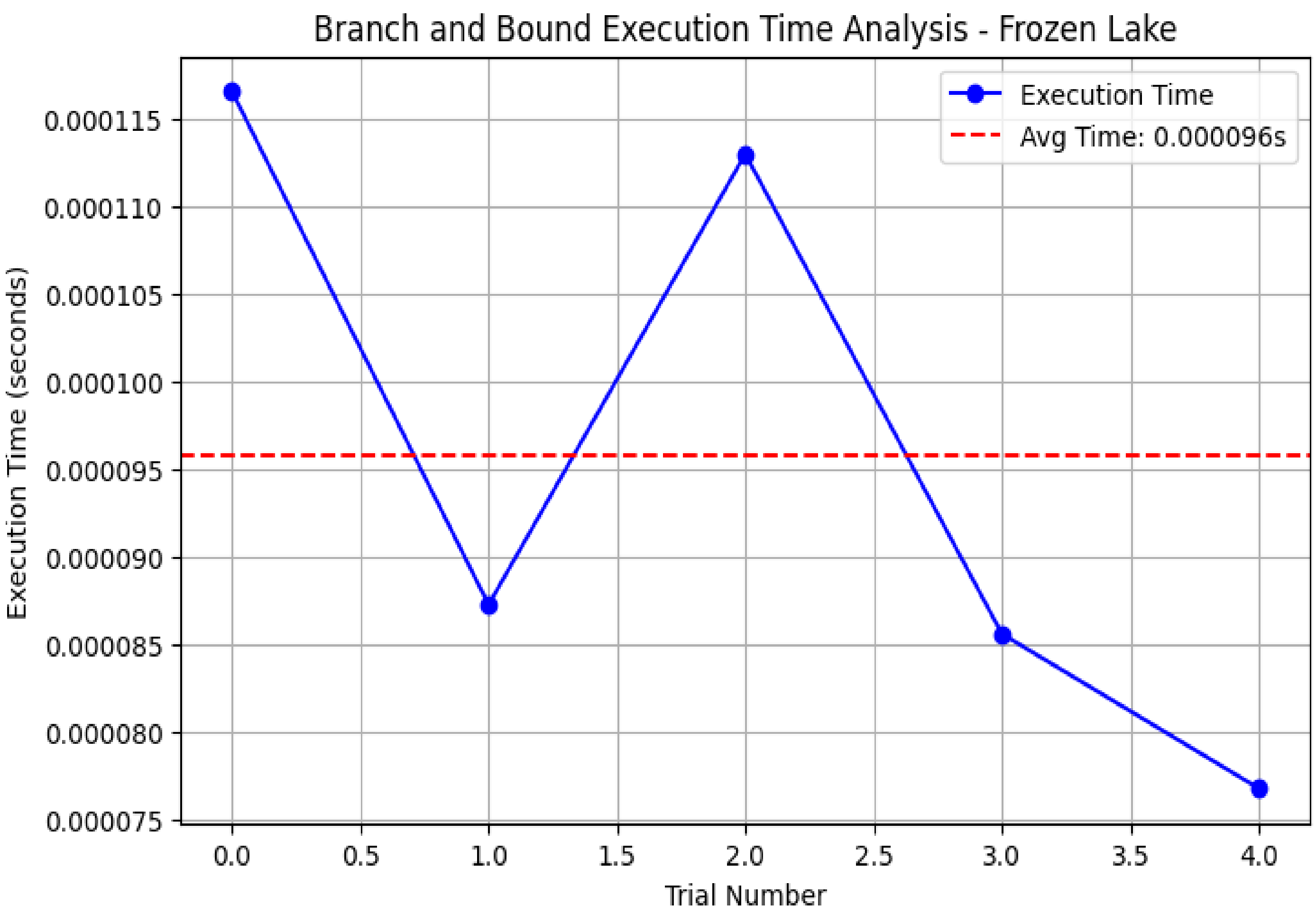
5. Solution Extraction
   - When the goal is reached, the path is reconstructed by backtracking from the goal to the start.

Performance Analysis

Visualization



Average Execution Time



Implementation Details of IDA* for Frozen Lake Problem

1. IDA Algorithm Overview

- IDA* is a memory-efficient heuristic search algorithm that combines:
  - Depth-First Search (DFS) to explore paths recursively.
  - A heuristic (Manhattan Distance)* to estimate remaining cost.
  - Iterative Deepening with increasing cost bounds for optimality.

2. Environment Representation

- The 4×4 Frozen Lake grid consists of:
  - S - Start (0,0)
  - F - Frozen surface (walkable)
  - H - Hole (fall in and lose)
  - G - Goal (3,3)

3. Heuristic Function

- The Manhattan Distance heuristic is used
  - $h(n)=|x_1-x_2|+|y_1-y_2|$
  - This provides an estimate of the number of moves required to reach the goal.
  - Efficient for grid-based problems.

4. Get Neighbours

- Retrieves valid next states.
- Considers all 4 possible moves.

5. Recursive Search with Cost Bound Pruning

- Computes f-cost:

  $$f(n) = g(n) + h(n)$$

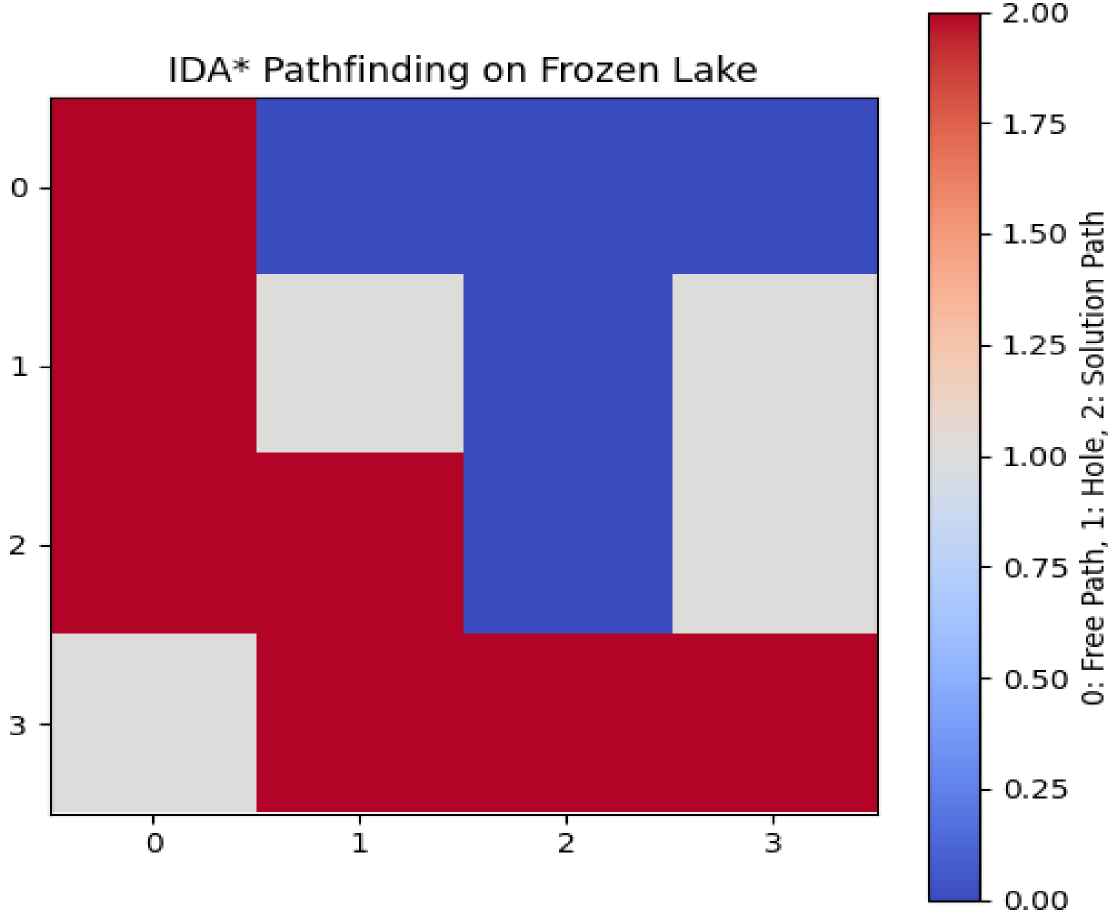
- If f-cost exceeds bound, return new bound.
- If the goal is reached, return the path.
- Recursively explores paths with DFS
- Prunes paths exceeding bound.

- Avoids cycles by checking visited nodes.
- Keeps track of the minimum f-cost.

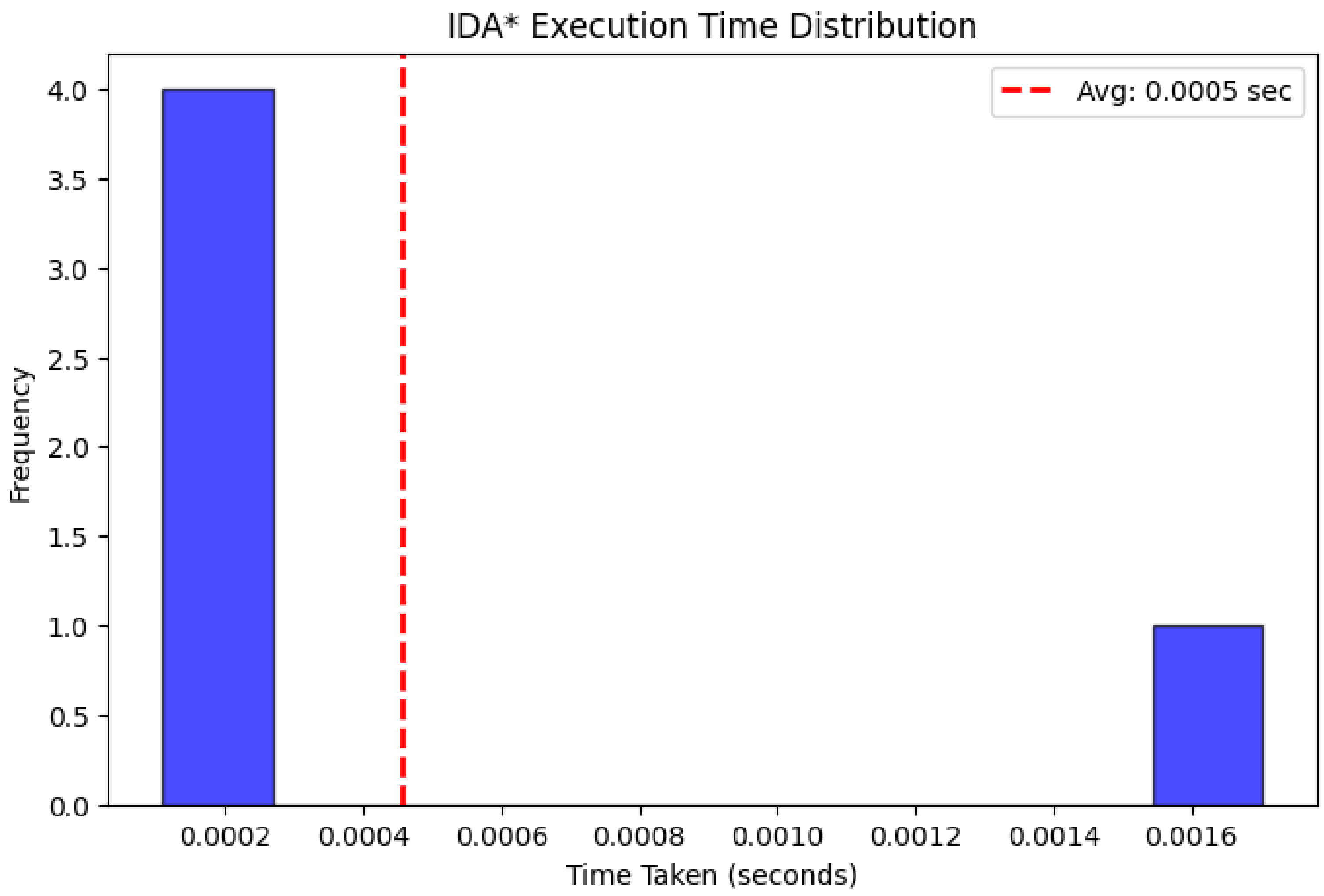
6. Iterative Deepening A Search
   - Initializes cost bound using heuristic.
   - Iterates until a solution is found.
   - Dynamically updates the cost bound.

Performance Analysis

Visualization



Average Execution Time



Implementation Details of Branch and Bound for Ant Maze Problem

1. Problem Definition
   - The maze is represented as a 5×5 grid.
   - 0 represents a free path, and 1 represents a wall (impassable).
   - The agent must move from (0,0) (top-left corner) to (4,4) (bottom-right corner) while taking the shortest route.
2. Node Representation

- Each state in the search space is represented as a Node object, containing:
  - position: The current (row, column) coordinate.
  - cost: The sum of the actual cost (g) and heuristic cost (h).
  - parent: A reference to the parent node (for path reconstruction).

3. Algorithm Steps

- Initialize a priority queue with the start node.
- Use a set (visited) to track explored positions.
- Loop until the queue is empty:
  - Pop the node with the lowest cost.
  - If the goal is reached, reconstruct and return the path.
  - If the node is already visited, continue.
  - Generate valid neighbors (avoiding walls and boundaries).
  - Calculate cost and add neighbors to the priority queue.
- If no path is found, return None.

4. Heuristic Function

- The Manhattan Distance heuristic is used
  - $h(n)=|x_1-x_2|+|y_1-y_2|$
  - This provides an estimate of the number of moves required to reach the goal.
  - Efficient for grid-based problems.

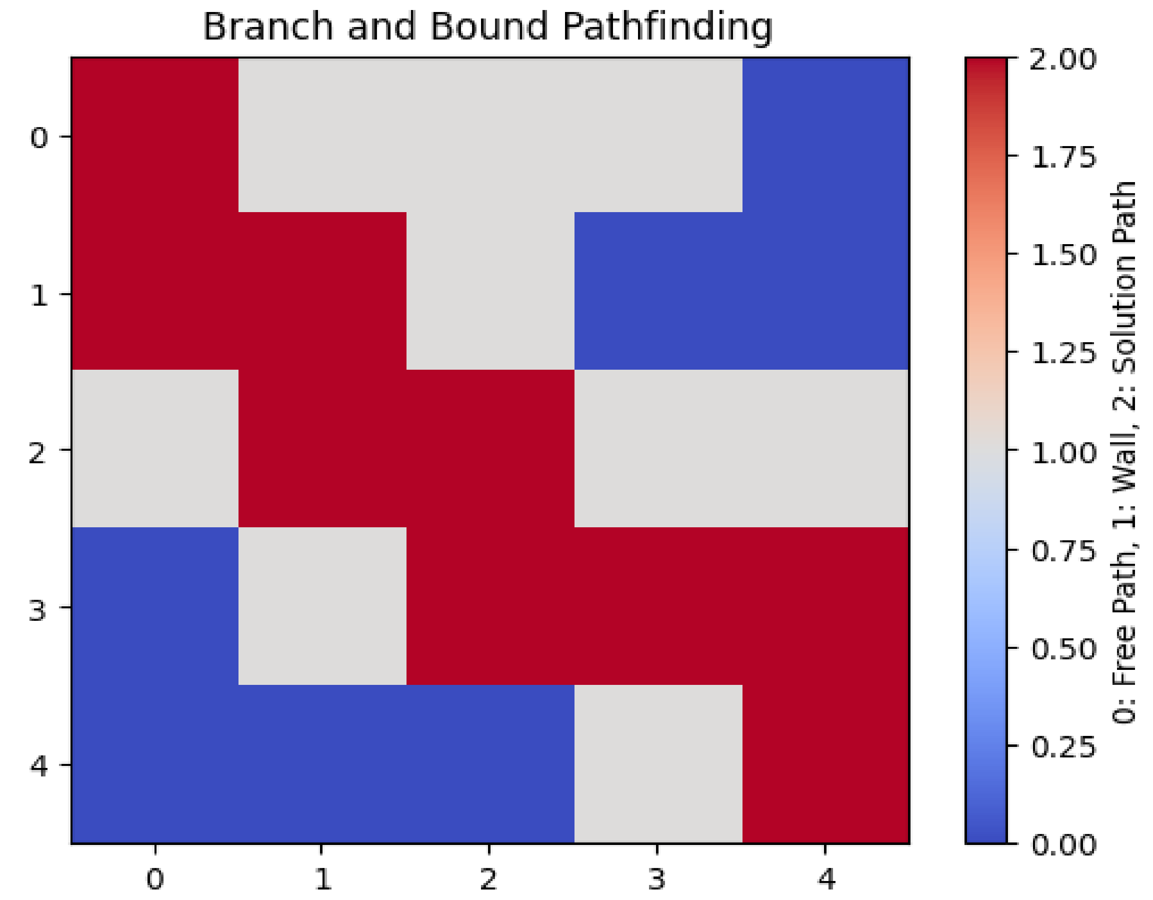
5. Cost Function

$$f(n) = g(n) + h(n)$$

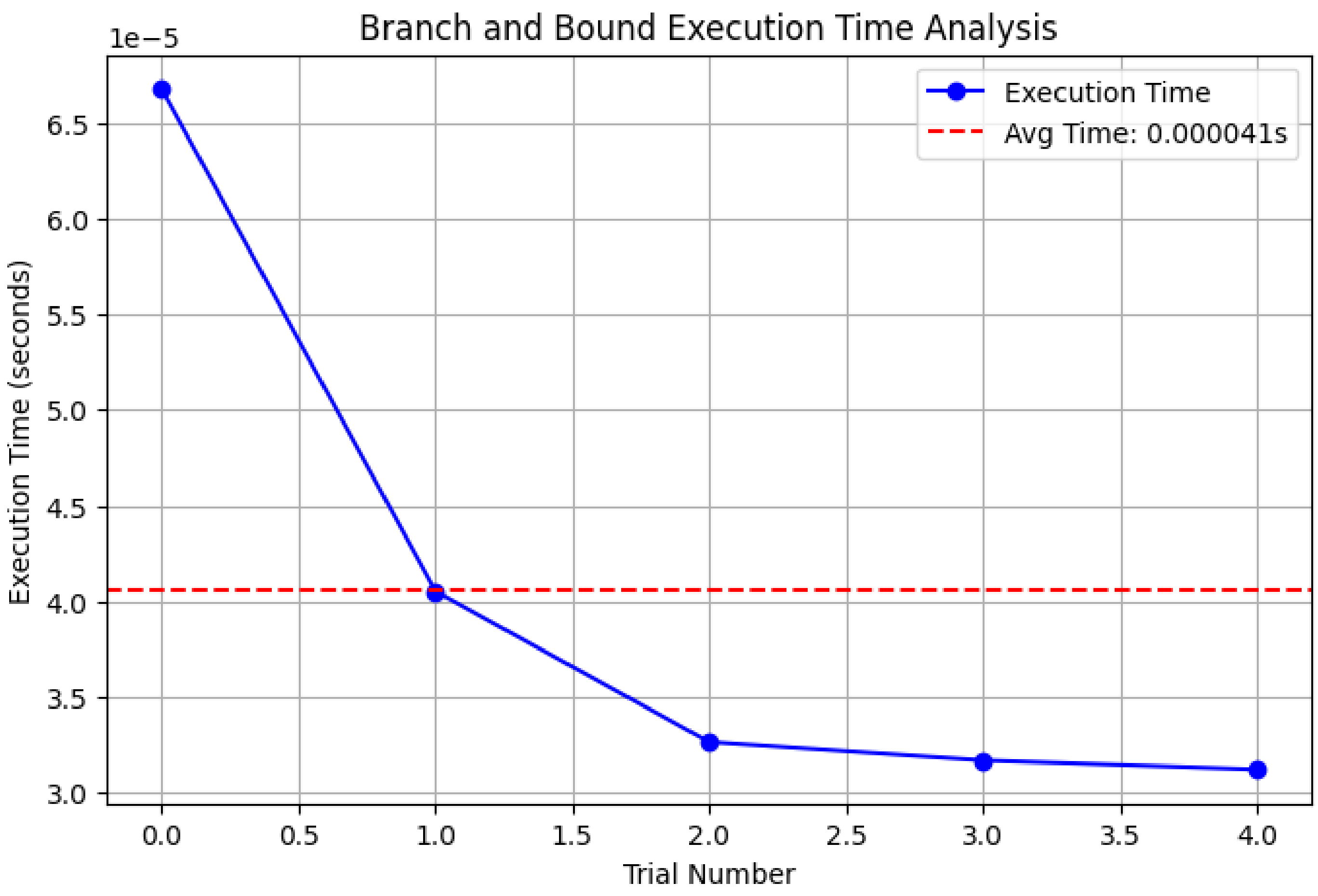
- g(n) = actual cost to reach node n from the start.
- h(n) = estimated cost from n to goal using Manhattan Distance.

Performance Analysis

Visualization



Average Execution Time



Implementation Details of IDA* for Ant Maze Problem

1. Problem Definition
   - The maze is represented as a 5×5 grid.
   - 0 represents a free path, and 1 represents a wall (impassable).
   - The agent must move from (0,0) (top-left corner) to (4,4) (bottom-right corner) while taking the shortest route.

2. Node Representation

- Each state in the search space is represented as a Node object, containing:
  - position: The current (row, column) coordinate.
  - cost: The sum of the actual cost (g) and heuristic cost (h).
  - parent: A reference to the parent node (for path reconstruction).

3. Algorithm Steps

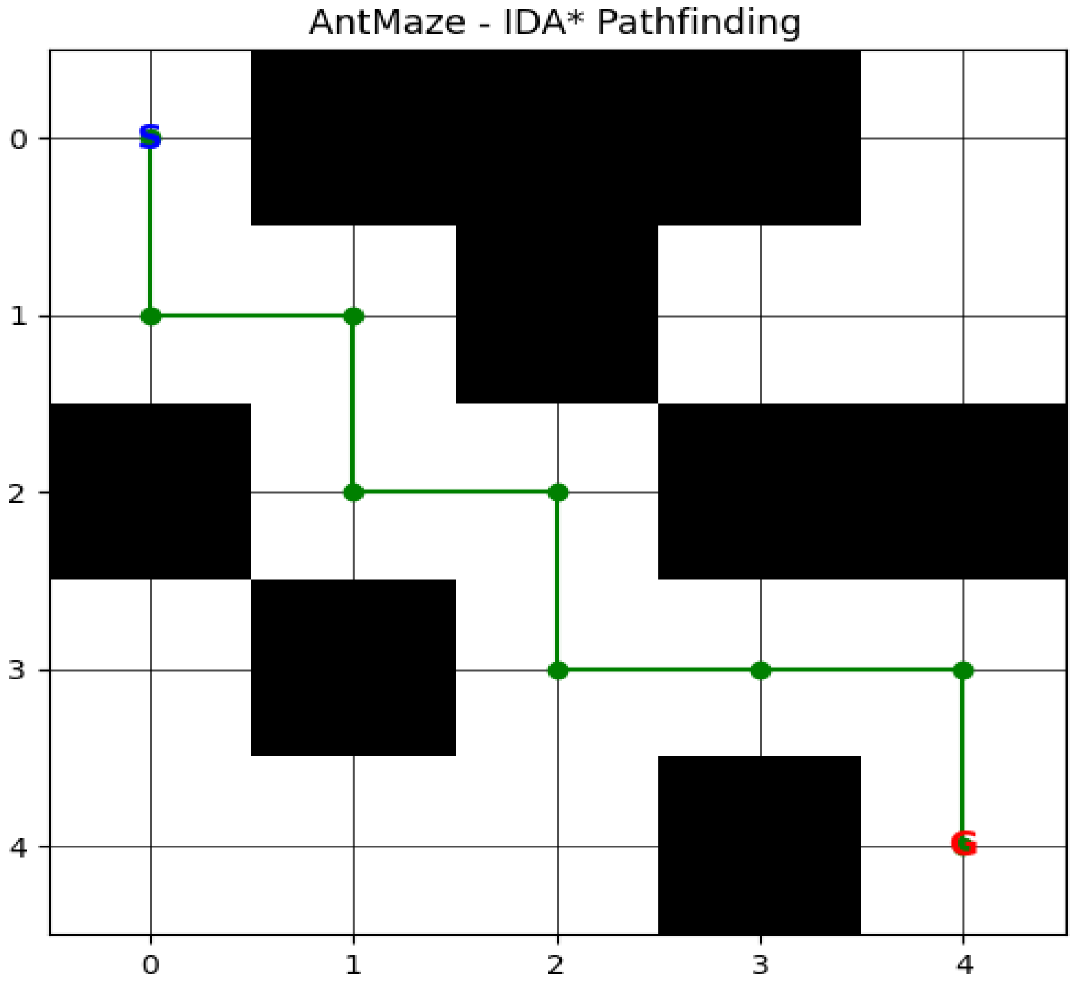
- Initialize the cost bound using the heuristic from the start to the goal.
- Perform depth-first search (DFS) within the cost limit:
  - If the current cost exceeds the bound, return the new minimum bound.
  - If the goal is reached, reconstruct and return the path.
  - Otherwise, explore all valid moves.
- If the path is not found in the current bound, increase the bound and repeat.
- If no path is found even after increasing the bound, return None.

4. Heuristic Function

- The Manhattan Distance heuristic is used
  - $h(n)=|x_1-x_2|+|y_1-y_2|$
  - This provides an estimate of the number of moves required to reach the goal.
  - Efficient for grid-based problems.

Performance Analysis

Visualization



Average Execution Time