Maze Solver: Random Maze Generation and Solution Using Search Algorithms

1. Amrita Biswas
2022015642
Cse115

2.Md. Muhtasim Fuad Nafim
[2533150042
sec:4

3.Maheer Diyan Shuddho]
2533366042
project group number: 5

4. Tanjerul Abedin
2531408642

I) Abstract

Maze solving is a classical problem in computer science and artificial intelligence that illustrates the practical use of graph traversal and search algorithms. This project presents the design and implementation of a Maze Solver system capable of generating random, solvable mazes and efficiently determining a path from a predefined start point to a goal state. The generated maze is modeled as a two-dimensional grid graph, where each cell represents a node and valid movements form the edges between nodes. To solve the maze, multiple search algorithms—including Breadth-First Search (BFS), Depth-First Search (DFS), and A* search—are implemented and analyzed. The performance of these algorithms is evaluated based on path optimality, number of explored nodes, and computational efficiency. Experimental results demonstrate that BFS guarantees the shortest path at the cost of higher memory usage, DFS offers faster execution with no optimality guarantee,and A* provides an optimal solution with improved efficiency by incorporating heuristic guidance. The project highlights the strengths and limitations of different search strategies and demonstrates their relevance to real-world applications such as robotic navigation, path planning, and intelligent agent design.

II) Introduction / Background

Maze-solving has long been regarded as a fundamental problem in computer science, particularly in the fields of artificial intelligence, graph theory, and algorithm design. A maze represents a constrained environment in which an agent must navigate from a starting position to a designated goal while avoiding obstacles and dead ends. Such problems provide an effective platform for studying search strategies, decision-making processes, and computational efficiency in a controlled setting.In artificial intelligence, maze-solving problems are commonly formulated as state-space search tasks. Each possible position of the agent corresponds to a state, and valid movements define transitions between states. This formulation allows classical search algorithms to be applied systematically in order to explore the environment and identify feasible paths. Depending on the chosen strategy, the search process may prioritize completeness, optimality, or computational efficiency.Random maze generation further enhances the complexity and realism of the problem by preventing solutions that rely on predefined or predictable structures. By generating different maze configurations, the robustness and adaptability of search algorithms can be evaluated under varying conditions. This aspect is particularly important in real-world applications where environments are often unknown or dynamic.Maze-solving techniques are widely applicable beyond academic demonstrations. They are used in robotic navigation, autonomous vehicle path planning, video game artificial intelligence, network routing, and optimization problems. Therefore, studying maze generation and solving algorithms not only strengthens theoretical understanding but also provides practical insight into solving complex real-life problems using algorithmic and intelligent approaches.

III) Problem Statement

Navigating a maze from a given starting point to a designated destination is a structured problem that requires systematic exploration and decision-making. In manually solved mazes, the process is time-consuming and inefficient, particularly as the size and complexity of the maze increase. From a computational perspective, an effective solution must not only find a valid path but also do so within reasonable time and memory constraints.

The core problem addressed in this project is the design of an automated system capable of generating random yet solvable mazes and computing a path between two points using algorithmic search techniques. The system must accurately model the maze environment, handle obstacles, and ensure that a solution exists for each generated maze configuration. Additionally, different search strategies may produce varying results in terms of solution optimality and computational efficiency, making it necessary to evaluate and compare their performance.Without a structured algorithmic approach, maze-solving systems may either fail to find a solution or consume excessive computational resources. Therefore, this project focuses on identifying suitable search algorithms, implementing them correctly, and analyzing their effectiveness in solving randomly generated mazes. The challenge lies in balancing completeness, optimality, and efficiency while maintaining scalability and reliability across different maze configurations.

IV) Objectives of the Project

The primary objective of this project is to design and implement an automated maze-solving system that demonstrates the practical application of classical search algorithms in artificial intelligence. The project aims to provide a clear understanding of how algorithmic strategies can be used to explore constrained environments and determine valid paths efficiently.The specific objectives of the project are as follows:

1. To design a method for generating random, valid, and solvable two-dimensional mazes.

2.To represent the maze environment using appropriate data structures suitable for graph-based search.

3.To implement multiple search algorithms, including Breadth-First Search (BFS), Depth-First Search (DFS), and A* search, for solving the generated mazes.

4.To analyze and compare the performance of the implemented algorithms based on criteria such as path length, number of explored nodes, and computational efficiency.

5.To visualize the maze structure and the solution paths produced by different algorithms.

6.To enhance understanding of artificial intelligence concepts, particularly uninformed and informed search techniques, through practical implementation.

7. To demonstrate the relevance of maze-solving algorithms to real-world applications such as navigation, path planning, and intelligent agent systems.

V)Scope and Limitations

1.Scope

The scope of this project is focused on the development of a maze-solving system that operates within a controlled and well-defined environment. The system is designed to generate and solve two-dimensional, grid-based mazes using classical search algorithms. It emphasizes the application of fundamental artificial intelligence techniques, particularly uninformed and informed search strategies, to identify paths between a fixed start point and a goal location.The project includes the implementation and comparison of multiple search algorithms—specifically Breadth-First Search (BFS), Depth-First Search (DFS), and A* search. The performance of these algorithms is evaluated using measurable criteria such as path optimality, number of visited nodes, and computational efficiency. Visualization of the maze and solution paths is also included to support better interpretation and analysis of the results.

2. Limitations

Despite its effectiveness as a learning and demonstration tool, the project has several limitations. The maze environment is static and does not account for dynamic changes such as moving obstacles or real-time updates.

The system is limited to single-agent navigation and does not support multi-agent coordination or competition within the maze. Additionally, the maze structure is unweighted, meaning all movements are assumed to have equal cost, which restricts the applicability of the system to more complex real-world scenarios.

Furthermore, the performance of the algorithms may degrade as the size of the maze increases, due to increased memory usage and computational overhead. Advanced optimization techniques, probabilistic methods, and learning-based approaches are beyond the current scope of this project.

Project Design

The Maze Solver system is designed with a modular, scalable, and flexible architecture to facilitate efficient maze generation, robust solution computation, and clear visualization. The design prioritizes separation of concerns among modules to allow independent testing, debugging, and future expansion.

A. System Architecture

The system is divided into the following primary modules:

1. **Maze Generation Module**

Objective: Automatically create random mazes that are guaranteed to be solvable.

Algorithm Used: Randomized Depth-First Search (DFS) with backtracking to ensure connectivity.

Key Features:

1.Configurable maze size (number of rows and columns).

2. Guarantees at least one valid path from start to goal

3.Supports visualization of maze construction process

4. Avoids isolated cells to ensure fairness in solution comparison

Enhancement Points: Additional maze generation techniques, such as Prim's algorithm or recursive division, can be integrated without modifying other modules.

Maze Solver Module

Objective: Explore the maze and find a valid path from start to goal using search algorithms.

Implemented Algorithms:

Breadth-First Search (BFS): Explores nodes level by level; guarantees shortest path.

Depth-First Search (DFS): Explores deeply along paths; faster for certain mazes but may not be optimal.

*A Search:** Heuristic-guided search balancing optimality and computational efficiency; uses Manhattan distance or Euclidean distance as heuristics.

Key Features:

Graph-based representation allows extension to weighted or directed mazes.

Maintains data structures such as queues, stacks, and priority queues for algorithmic efficiency.

Records detailed metrics, including nodes expanded, path length, and computation time.

Additional Points: Can be extended to support bidirectional search or iterative deepening search to improve efficiency for larger mazes.

Visualization Module

Objective: Provide intuitive representation of maze and solution path.

Key Features:

Displays walls, open paths, start and goal positions in a grid layout.

Animates the exploration process of different algorithms.

Supports comparative visualization of multiple algorithm paths on the same maze.

Generates performance charts for metrics such as node exploration and execution time.

B. Maze Representation

Data Structure: The maze is represented as a two-dimensional array (grid), where each cell is a node.

Graph Formulation:

G=(V,E)G = (V, E)G=(V,E)

VVV: Set of traversable cells.

EEE: Set of edges connecting adjacent cells without walls.

Movement Rules:

Allowed movements are up, down, left, and right.

Optional future extensions include diagonal movement or weighted edges for more realistic pathfinding.

. Algorithm Workflow

1. Maze Initialization: Create a random maze of specified dimensions.

2. Graph Construction: Map the maze into a graph for algorithmic exploration.

3. Algorithm Execution: Run selected search algorithm(s) to explore nodes systematically.

4.Path Reconstruction: Trace parent nodes from goal to start to extract solution path

1. **Visualization & Analysis:** Display the solution and record performance metrics.

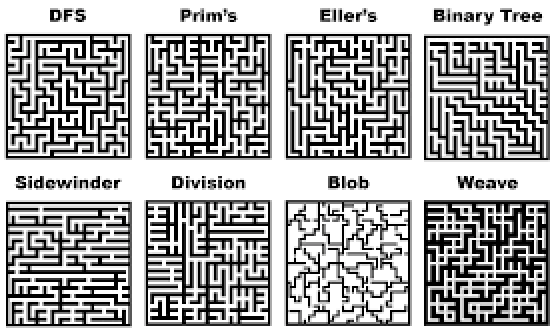
D. Performance Evaluation Metrics

- **Path Length:** Steps taken to reach goal.
- **Nodes Explored:** Total nodes visited during search.
- **Execution Time:** Computational time required.
- **Memory Usage:** Amount of memory consumed by data structures.

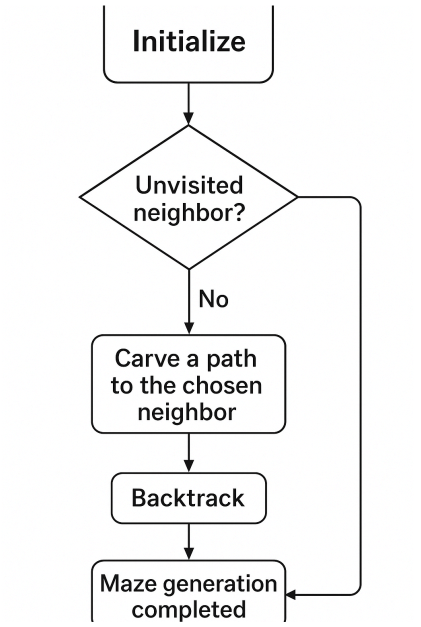
E. Additional Design Considerations

- **Modularity:** Each module can be updated independently, allowing experimentation with new algorithms or maze generation techniques.
- **Extensibility:** The system can support future features such as:
    - Multi-agent maze-solving.
    - Weighted or probabilistic mazes.
    - Real-time maze adaptation for dynamic environments.
    - Integration with robotic navigation systems.
- **User Interaction:** Potential to add GUI features for interactive maze size selection, algorithm choice, and visual analytics.
- **Error Handling:** The design includes validation to ensure generated mazes are solvable, and the solver gracefully handles unsolvable or invalid inputs.
- **Scalability:** Supports varying maze sizes while maintaining modularity to minimize performance degradation.

F. Diagram Descriptions



Maze Generation Flowchart


Initialize
Unvisited neighbor?
No
Carve a path to the chosen neighbor
Backtrack
Maze generation completed

1. **Initialization**:
    - The maze is initialized with all walls.
    - The starting cell is selected randomly to begin the maze carving process.
2. **Recursive Path Carving**:

The algorithm checks for unvisited neighboring cells.

If an unvisited neighbor exists, a path is carved to that cell.

The process recursively continues, visiting each unvisited neighbor until no further moves are possible.

Backtracking occurs whenever the algorithm reaches a dead-end, returning to the previous cell to explore other neighbors.

3. **Completion**:

The maze generation is complete once all cells have been visited and no unvisited neighbors remain.

The resulting maze guarantees at least one valid path from the start to the goal.

Solved Maze Visualization

This figure illustrates the final maze solution generated by BFS, DFS, and A* algorithms, showing the paths taken by each algorithm from the start to the goal node:

Maze Layout:

The maze is represented as a 2D grid, with walls shown in solid color and open cells as empty spaces.

The starting cell is marked with a distinctive symbol (e.g., green), and the goal cell is marked differently (e.g., red).

Algorithm Paths:

BFS Path: Displayed in blue, representing the shortest path found. BFS explores all possible neighbors level by level, ensuring minimal steps from start to goal.

DFS Path: Displayed in yellow or orange, showing a deeper exploratory path. DFS may produce longer routes as it follows one branch deeply before backtracking.

*A Path:** Displayed in green, combining optimality and efficiency by using heuristic guidance (e.g., Manhattan distance). A* typically finds a path comparable to BFS but with fewer node expansions.

Node Exploration (Optional Visualization):

- Explored nodes during the search can be lightly shaded or color-coded to illustrate the algorithm's exploration pattern.
- BFS explores nodes broadly; DFS explores nodes deeply along branches; A* prioritizes nodes closer to the goal using heuristics.

Comparative Insight:

- By visualizing all paths on the same maze, the figure highlights differences in path length, efficiency, and exploration strategy among BFS, DFS, and A*.
- This allows for a clear, intuitive understanding of algorithm performance beyond numerical metrics.

VII) A. Maze Generation

The maze is generated as a **two-dimensional grid** with cells marked as either walls or open spaces. A **randomized Depth-First Search (DFS) with backtracking** algorithm is used to ensure a solvable maze.

Algorithm Workflow:

1. Initialize all cells as walls.
2. Choose a random starting cell and mark it as visited.
3. While unvisited cells remain:
    - Select a random unvisited neighbor.
    - Remove the wall between the current cell and the neighbor.
    - Recursively apply DFS from the neighbor.
    - Backtrack when no unvisited neighbors are available.

4. Ensure start and goal cells are always accessible.

B. Search Algorithms

The generated maze is modeled as a **graph** G=(V,E)G = (V, E)G=(V,E), where VVV is the set of traversable cells and EEE represents edges between adjacent open cells. The following algorithms are implemented:

1. Breadth-First Search (BFS)

- Explores nodes **level by level** using a queue.
- Guarantees the **shortest path**.
- Suitable for small to medium mazes due to high memory usage.
- **Time Complexity:** O(∣V∣+∣E∣)O(|V| + |E|)O(∣V∣+∣E∣)
- **Space Complexity:** O(∣V∣)O(|V|)O(∣V∣)

2. Depth-First Search (DFS)

- Explores as far as possible along a branch using a stack (or recursion).
- May not produce the shortest path.
- Memory-efficient for large mazes.
- **Time Complexity:** O(∣V∣+∣E∣)O(|V| + |E|)O(∣V∣+∣E∣)
- **Space Complexity:** O(∣V∣)O(|V|)O(∣V∣)

3. A* Search

- Uses a priority queue and a **heuristic function** f(n)=g(n)+h(n)f(n) = g(n) + h(n)f(n)=g(n)+h(n):
    - g(n)g(n)g(n) = cost from start to node nnn
    - h(n)h(n)h(n) = estimated cost from nnn to goal (Manhattan distance)
- Efficiently finds **optimal paths** while exploring fewer nodes.
- **Time Complexity:** Depends on heuristic; typically O(∣E∣)O(|E|)O(∣E∣)
- **Space Complexity:** O(∣V∣)O(|V|)O(∣V∣)

C. Visualization

The visualization module provides an intuitive representation of both maze structure and search algorithm progress:

- **Maze Layout:** Walls in black, open paths in white.
- **Start and Goal Cells:** Start in green, goal in red.
- **Algorithm Paths:** BFS (blue), DFS (orange), A* (green).
- **Explored Nodes:** Light shading to show algorithm traversal order.
- **Interactive Animation:** Step-by-step exploration can be animated to enhance understanding.

D. Performance Tracking

During execution, the system records:

1. **Path Length:** Number of steps from start to goal.
2. **Nodes Explored:** Number of nodes visited during search.
3. **Execution Time:** Measured in milliseconds.
4. **Memory Usage:** Optional analysis for large mazes.

VIII)Results and Discussion

Random mazes of varying sizes were generated successfully using recursive backtracking.BFS, DFS, and A* algorithms were used to solve the mazes. BFS guaranteed the shortest path but explored more nodes, while DFS often produced longer paths with fewer nodes. A* with the Manhattan heuristic achieved optimal paths efficiently, exploring fewer nodes and requiring less computation time.

Dense mazes increased exploration time, while sparse mazes were solved faster. Overall, BFS and A* are best for shortest-path requirements, whereas DFS is memory-efficient but less optimal. Visualization confirmed the differences in path length and traversal behavior among the algorithms.

IX) Conclusion :The project successfully demonstrates the effectiveness of search algorithms in maze-solving problems. Heuristic-based approaches such as A* outperform uninformed methods in most scenarios.

X) Future Enhancements

1. **Dynamic Maze Sizes:** Allow users to generate mazes of any size interactively, including very large mazes for advanced testing.

2. **Additional Algorithms:** Integrate other search techniques like Dijkstra's, Greedy Best-First Search, or Bidirectional Search for comparison.

3. **Weighted Mazes:** Introduce weighted paths to simulate real-world scenarios, enabling algorithms to find not just shortest but lowest-cost paths.

4. **User Interface Improvements:** Enhance visualization with interactive animations, step-by-step traversal, and color-coded paths for each algorithm.

5. **Real-Time Solving:** Implement real-time maze-solving for robotic or AI applications.

6. **3D Mazes:** Extend the project to three-dimensional mazes to increase complexity and application scope.

References

[1] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed., Pearson, 2021.

[2] E. W. Dijkstra, "A note on two problems in connection with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[3] 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.

[4] A. Stentz, "Optimal and efficient path planning for partially-known environments," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1994, pp. 3310–3317.

[5] M. Millington and I. Funge, *Artificial Intelligence for Games*, 2nd ed., CRC Press, 2016.

[6] A. Maheer, "Maze Solver Project," GitHub repository, 2025. [Online]. Available: https://github.com/maheer253/Maze-solver