Maze Runner: Solving Mazes Using and Comparing Search Algorithms

Submitted by

Elhusseain Aboulfetouh 202202239

Amr Khalid 202201502

Mohamed Hassan 202201257

Seif Mohamed 202201554

Course

CSAI 301 Artificial Intelligence

1. Introduction

During the Classical antiquity era, many philosophers, such as Aristotle sought to formulate a set of rules to mechanise the human thought process. That is, given a set of correct initial premises, one can mechanically generate a conclusion. In the 1940s, scientists built on those theories and assumptions to invent the first programmable computer. A few years later, the AI research field was founded. This paper discusses the implementation of different AI search algorithms from different categories like: informed search, uninformed search, and local search and applying those algorithms to solve the maze runner problem (path finding problem).

2. Problem Modeling

The Maze Runner problem is represented after discretization as a grid of M x N dimensions for the height and the width of the grid, respectively. Each cell in the grid could have a border on any of its 4 dimensions preventing the movement in the direction of the border. The runner (Problem's Agent) could be present in any of the cells in the M x N grid.

2.1 Problem's State Space Representation

Each State that the runner could be present in is represented using its row (X) and its column (Y) in a tuple pair.

$$\text{State} = (X,Y) \tag{1}$$

The Problem's State Space starts from (1,1) for the top left cell in the grid, reaching (M,N) for the bottom right cell.

2.2 Initial State

The initial State represents the starting position of the runner. Both X_O and Y_O have to belong to the State Space

$$\text{Initial State} = (X_O,Y_O) \tag{2}$$

2.3 Goal State

The Goal State represents the target position that the runner should reach by the end of the problem.

$$\text{Goal State} = (X_G,Y_G) \tag{3}$$

2.4 Actions

At each state, an agent can either go one step to east, west, north, or south. If there is a wall in the direction of the agent's movement, then the agent can not take a step towards that direction.

2.5 Transition Functions

Given a state and a valid action, the agent moves one step in the direction decided by the action.

2.6 Problem Size Estimation

The problem size could be determined by the number of cells and the number of borders and their locations.

$$\text{Number of Cells} = M \times N \quad (4)$$

The state space is always smaller than or equal to the number of cells, this is due to the probability of having borders blocking the way to any of the cells in the grid.

$$\text{State Space Size} <= M \times N \quad (5)$$

3. Modeling assumptions

- The problem assumes random cost to reach each cell from any other cell during the implementation of some of the cost based search algorithms such as the Uniform Cost Search (UCS) and the A* Search.
- The problem assumes the heuristic used in Informed Search methods to be the distance between each cell and the Goal State determined by either a Manhattan Distance Function or the Euclidean Distance Function.
- The problem assumes the scheduling function used in the Simulated Annealing Search Algorithm is a linear function.
- The problem assumes the fitness function used in the Genetic Algorithm search is how close each gene in the chromosome is close to each gene in the goal's chromosome.
- The problem assumes that there will always be a goal which is predetermined before the search.

4. Uninformed Search Algorithms

This section contains tables recording the time taken in microseconds, visited states, and the convergence status for each algorithm for different maze sizes. It also states time complexity, space complexity, completeness, and optimality for the general case of each algorithm. For each maze size across all algorithms, the agent's starting position is the upper leftmost cell (i.e., (1, 1)) and the goal position is the lower rightmost cell (i.e., (25, 25)).

4.1 Breadth-First Search

- Space Complexity: $O(b^d)$
- Time Complexity: O(N)
- Completeness: Complete
- Optimality: Optimal in case of equal costs.

Grid Size	Time in microseconds	Visited states	Converged
25 x 25	2503.6	584	Yes
50 x 50	5506.3	2489	Yes
100 x 100	22454.9	9971	Yes

4.2 Depth First Search

- Space Complexity: O(bd)
- Time Complexity: $O(b^m)$
- Completeness: Complete (for finite space)
- Optimality: Not Optimal.

Grid Size	Time in microseconds	Visited states	Converged
25 x 25	1000	126	Yes
50 x 50	14061.7	1250	Yes
100 x 100	124738.7	4145	Yes

4.3 Uniform Cost Search

- Space Complexity: $O(b^d)$
- Time Complexity: O(N)
- Completeness: Complete (for non-negative costs)
- Optimality: Optimal.

Grid Size	Time in microseconds	Visited states	Converged
25 x 25	3122.3	625	Yes
50 x 50	12107.6	2500	Yes
100 x 100	50394	10000	Yes

4.4 Iterative Deepening Search

- Space Complexity: O(bd)
- Time Complexity: $O(b^d)$
- Completeness: Complete
- Optimality: Optimal in case of equal costs.

Grid Size	Time in microseconds	Visited states	Converged
25 x 25	3509	519	Yes
50 x 50	49046.7	1827	Yes
100 x 100	910209.9	7757	Yes

5. Informed Heuristic Search Algorithms

5.1 Greedy Best First Search

- Space Complexity: $O(b^m)$
- Time Complexity: $O(b^m)$
- Completeness: Complete (for finite space)
- Optimality: Not Optimal.

Grid Size	Time in microseconds	Visited states	Converged
25 x 25	1000	83	Yes
50 x 50	1026.63	142	Yes
100 x 100	2998.828	417	Yes

5.2 A* Search

- Space Complexity: $O(b^m)$
- Time Complexity: $O(b^m)$
- Completeness: Complete (for finite space)
- Optimality:A* is **not optimal** in all cases of our project. In our project , which includes cost of effort done by climbing up or down heights, cost is randomized. Therefore, at instance the cost could drop lower than the chosen heuristics, therefore leading to **inadmissibility**.

Grid Size	Time in microseconds	Visited states	Converged
25 x 25	1018.047	185	Yes
50 x 50	2997.15	701	Yes
100 x 100	5997.65	1138	Yes

6. Local Search

6.1 Hill Climbing Search

- Space Complexity: O(1)
- Time Complexity: O(n), where n is the number of visited states.
- Complete: No
- Optimal: No

Grid Size	Time in microseconds	Visited states	Converged
25 x 25	0.0	21	No
50 x 50	0.0	26	No
100 x 100	0.0	91	No

6.2 Simulated Annealing Search

- Space Complexity: O(1)
- Time Complexity: O(T), where T is the scheduler.
- Complete: No
- Optimal: No

Grid Size	Time in microseconds	Visited states	Converged
25 x 25	2049.9	236	No
50 x 50	2643.8	298	No
100 x 100	2996.9	199	No

6.2 Genetic Algorithm Search

For each trial of the algorithm the population size was set to 20 and maximum number of generations allowed was set to 10000.

- Space Complexity: O(P), where P is the population size.
- Time Complexity: O(P * G), where P is the population size and G is the number of generations.
- Complete: No
- Optimal: No

Grid Size	Time in microseconds	Visited states	Converged
25 x 25	548236.6	15740	Yes
50 x 50	1158477.5	35780	Yes
100 x 100	6462856.7	200020	Yes

7. Comparison

	Algorithm	Time in microseconds	Visited states	Converged
Uninformed Search	BFS	22454.9	9971	Yes
	DFS	124738.7	4145	Yes
	UCS	50394	10000	Yes
	IDS	910209.9	7757	Yes
Informed Search	A*	5997.65	1138	Yes
	GBF	2998.828	417	Yes
Local Search	HC	0.0	91	No
	SA	2996.9	199	No
	GA	6462856.7	200020	Yes

8. Conclusion

- **Algorithms** like **BFS** and **UCS** are well-suited for small problems:
    - They are complete and optimal under specific conditions, making them more reliable for precise problem-solving.
    - However, Their high usage and computational costs make them less practical for larger, more complex problems.
- **A*** is an excellent choice for practical Scenarios:
    - With good heuristic functions, it offers a balance of efficiency, completeness, and optimality.
    - It is especially effective for moderately large problems with structured state spaces.
- For **large problems**, Hill Climbing **(HC)** and Simulated Annealing **(SA)** are generally **impractical**:
    - Their lack of completeness and optimality limits their reliability.
    - Despite their low space complexity, they are better suited for approximations rather than precise solutions.
- **Genetic Algorithms (GA)** are versatile for **scaling to large problems**:
    - They explore the search space effectively, making them applicable in cases where exact solutions are unnecessary.
    - However, their inefficiency and tendency to produce non-optimal solutions limit their use in tasks requiring precision.