Maze Solving Algorithms: Comprehensive Analysis

May 30, 2025

Abstract

This report presents a comprehensive analysis of maze-solving algorithms, encompassing both traditional search methods and modern machine learning approaches. We evaluate 10 distinct algorithms implemented in Python on a 100×100 maze, comparing performance through quantitative metrics including success rate, path optimality, memory usage, and computational efficiency. The study provides insights into algorithm selection for pathfinding problems under varying constraints.

Contents

1 Introduction **1**

2 Problem Modeling **2**
2.1 State Space Representation 2
2.2 States and Actions 2
2.3 Transition Function 2
2.4 Problem Complexity 2

3 Algorithm Specifications **2**
3.1 Traditional Search Methods 2
3.2 Heuristic Methods 3
3.3 Local Search Algorithms 3
3.4 Machine Learning Approaches 3

4 Performance Analysis **4**
4.1 Theoretical Comparison 4
4.2 Empirical Metrics 4
4.3 Visualization 4

5 Conclusion **4**

1 Introduction

The primary objective is to solve a maze navigation problem where an agent moves from a start state (1,1) to a goal state (100,100) in a 100×100 grid. We implement and compare:

- Traditional search algorithms (BFS, DFS, UCS, IDS, A*)
- Heuristic methods (Greedy Best-First)
- Local search techniques (Hill Climbing, Simulated Annealing)
- Machine learning approaches (Q-Learning, Genetic Algorithms)

2 Problem Modeling

2.1 State Space Representation

- 100×100 grid (10,000 discrete states)
- Each state: (x, y) where x=row, y=column
- Movement constraints: Walls block transitions in cardinal directions (N,S,E,W)

2.2 States and Actions

- **Initial state**: (1,1) (top-left corner)
- **Goal state**: (100,100) (bottom-right corner)
- **Actions**:

$$\begin{aligned} \text{N: } (x, y) &\to (x-1, y) \\ \text{W: } (x, y) &\to (x, y-1) \\ \text{S: } (x, y) &\to (x+1, y) \\ \text{E: } (x, y) &\to (x, y+1) \end{aligned}$$

2.3 Transition Function

$$T(s, a) = \begin{cases} s' & \text{if no wall in direction } a \\ s & \text{otherwise (blocked movement)} \end{cases}$$

2.4 Problem Complexity

- **State space**: 10,000 states
- **Maximum transitions**: 40,000 (4 per state)
- **Actual transitions**: Reduced by wall configurations
- **Reachable states**: ∼1,000–10,000 (maze-dependent)
- Key complexity factors:
  - Blocked paths creating decision branches
  - Possible cyclic loops
  - Multiple alternative routes

3 Algorithm Specifications

3.1 Traditional Search Methods

- **Breadth-First Search (BFS)**:
  - Queue-based frontier management
  - Guarantees shortest path (step count)
  - Completeness: Yes
- **Depth-First Search (DFS)**:
  - Stack-based (LIFO) exploration
  - Risk of infinite loops without cycle detection
  - Non-optimal solutions
- **Uniform Cost Search (UCS)**:

  - Priority queue ordered by path cost
  - Optimal for weighted graphs
  - Dijkstra's algorithm variant

- **Iterative Deepening Search (IDS)**:
  - Depth-limited DFS with incremental depth
  - Combines BFS completeness with DFS memory efficiency

- **A* Search**:
  - Heuristic function: $f(n) = g(n) + h(n)$
  - $g(n)$: Path cost from start
  - $h(n)$: Heuristic estimate to goal (Manhattan distance)
  - Optimal if heuristic admissible/consistent

3.2 Heuristic Methods

- **Greedy Best-First Search**:
  - Expands nodes with minimal $h(n)$
  - Fast but non-optimal
  - Prone to local minima

3.3 Local Search Algorithms

- **Hill Climbing**:
  - Gradient ascent toward goal
  - Terminates at local maxima
  - No backtracking

- **Simulated Annealing**:
  - Probabilistic acceptance of worse moves
  - Temperature schedule: $T_k = T_0 / \log(k + 1)$
  - Escapes local optima via exploration

3.4 Machine Learning Approaches

- **Q-Learning**:
  - Update rule: $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
  - $\alpha = 0.1$, $\gamma = 0.99$, ϵ-greedy strategy ($\epsilon = 0.2$)
  - Q-table implementation

- **Genetic Algorithm**:
  - Chromosome: Move sequence encoding
  - Operators:
    * Selection: Tournament (size=5)
    * Crossover: Single-point (rate=0.8)
    * Mutation: Random move change (rate=0.2)
  - Population: 500, Generations: 500
  - Fitness: Path length + goal achievement

4 Performance Analysis

4.1 Theoretical Comparison

Table 1: Algorithm Characteristics Comparison

Algorithm	**Complete**	**Optimal**	**Time Complexity**	**Space Complexity**
BFS	Yes	Yes (steps)	$O(b^d)$	$O(b^d)$
DFS	Yes (finite)	No	$O(b^m)$	$O(bm)$
UCS	Yes	Yes (cost)	$O(b^{1+c^*/\epsilon})$	$O(b^{1+c^*/\epsilon})$
IDS	Yes	Yes	$O(b^d)$	$O(bd)$
A* Search	Yes	Yes*	$O(b^d)$	$O(bd)$
Greedy Best-First	Yes	No	$O(b^m)$	$O(b^m)$
Hill Climbing	Sometimes	No	$O(b)$	$O(1)$
Simulated Annealing	Probabilistic	No	$O(1)$	$O(1)$
Genetic Algorithm	Probabilistic	No	$O(G \cdot P \cdot E)$	$O(P \cdot S)$
Q-Learning	Probabilistic	Asymptotic	$O(\lvert\mathcal{S}\rvert \cdot \lvert\mathcal{A}\rvert)$	$O(\lvert\mathcal{S}\rvert \cdot \lvert\mathcal{A}\rvert)$

*With admissible heuristic; b: branching factor, d: solution depth, m: max depth, c^*: optimal cost, ϵ: cost step, G: generations, P: population size, E: evaluation cost, S: state space

4.2 Empirical Metrics

Evaluation criteria applied to 100 maze instances:

- **Success rate**: Percentage reaching (100,100)
- **Path length**: Steps in solution path
- **Computational time**: Wall-clock execution
- **Memory usage**: Peak memory consumption
- **Training time** (ML approaches): Time to convergence

4.3 Visualization

- Real-time path animation
- Frontier expansion heatmaps
- Cost function landscapes
- Q-table value distributions

5 Conclusion

- **Optimal paths**: BFS, UCS, and A* consistently find minimal-cost solutions
- **Memory efficiency**: IDS and Hill Climbing minimize space requirements
- **Large mazes**: A* outperforms others when using admissible heuristics
- **ML approaches**: Q-Learning adapts best to dynamic environments
- **Heuristic methods**: Greedy Best-First provides fastest suboptimal solutions
- **Stochastic methods**: Genetic Algorithms handle multimodal solution spaces effectively