Complexity Comparison and Analysis of Maze Algorithms

Yifan Zhao *

School of Advanced Technology, Xi'an Jiaotong-Liverpool University, Suzhou, China

* Corresponding Author Email: Yifan.Zhao2302@student.xjtlu.edu.cn

Abstract. Maze generation and solving algorithms are widely used in areas such as path planning and game design. However, existing research has primarily focused on optimizing the performance of individual algorithms. To address this limitation, this study analyzes four maze generation algorithms and four maze-solving algorithms. The article firstly introduces the core principles, implementation and key characteristics of each algorithm. It then systematically evaluates their time complexity and space complexity. The results show that among the generation algorithms, the binary tree algorithm is the most efficient but produces biased maze structures. The Prim's algorithm generates highly complex mazes at the expense of substantial space and time requirements. Among solving algorithms, A-star algorithm achieves optimal ideal efficiency Ω(n) when equipped with a well-designed heuristic function. BFS and DFS are highly efficient, but they can only fit unweighted paths. This study provides a theoretical basis for algorithm selection. Future work will involve experimental measurements of the actual runtime performance of these algorithms.

Keywords: Algorithms, complexity, maze generation, maze solving.

1. Introduction

Before the formalization of computer science, mathematicians had conducted elementary explorations into theoretical problems related to mazes. The development of graph theory and topology had jointly laid the theoretical foundation for maze problems. With the development of computer science, researchers have progressively established a series of algorithms for constructing maze structures or solving maze-related problems. In the field of maze generation, the Prim algorithm proposed in 1957 is a classical approach [1]. It can generate mazes with an acyclic tree-like structure by employing a greedy strategy to progressively expand random walls. The recursive division algorithm, which formed in the 21st century, offers another approach to maze construction. It recursively bisects maze space and randomly creates holes on the dividing lines, eventually generating a maze with hierarchical structure. In the field of maze solution, Dijkstra's algorithm introduced in 1959 first provided a solution to single-source shortest path problems [2]. In 1968, Hart et al. enhanced this algorithm by adding a heuristic function [3]. From this, they developed the more efficient A* algorithm [3].

Although these algorithms have been comprehensively verified in theory, their performance differences in practical applications still require systematic comparison and analysis. The current research still has two main limitations. Primarily, existing research mainly concentrates on performance analysis and optimization of an individual algorithm. For instance, the research has evaluated optimization methods for Dijkstra's algorithm, including heuristic-guided search and hierarchical preprocessing [4]. These studies rarely conduct complexity analyses across multiple algorithms. Secondarily, current research primarily focuses on theoretical time complexity analysis. Although different algorithms may have similar theoretical time complexity, practical applications often take into account the influence of factors such as occupied space and data distribution.

To address these limitations, this study reviewed and analyzed the existing literature. Through evaluating the complexity differences among various maze algorithms, the study provides scientific evidence for algorithm selection in practical applications such as path planning and game development.

The remainder of this article is structured into three main parts. The second section discusses the fundamental principles and features of the maze algorithms. It includes four maze generation algorithms: binary tree algorithm, Aldous Broder algorithm, recursive backtracking algorithm and the

Prim's algorithm. It studies also employs four maze solving algorithms: the foundational Depth First Search (DFS) and Breadth First Search (BFS), along with the more complex Dijkstra's Algorithm and A-star algorithm. Section 3 is devoted to a summary of the results, followed by a detailed discussion. It analyzes the performance differences among various algorithms and suggests future research directions for optimization. The fourth section summarizes the whole article. The author anticipates that this work will provide both theoretical foundations and data support for future research on related algorithms.

2. Method

This section presents eight classical maze algorithms. These algorithms are divided into two major categories: maze generation algorithms and maze solving algorithms. The article will explain their core concepts, principles and characteristics respectively.

2.1. Maze Generation Algorithms

To facilitate the understanding of algorithms, this section employs a grid model to represent the maze. The maze consists of an M × N rectangular grid, where each cell represents an independent node. Walls exist between all adjacent cells. Passages are formed by removing these walls between cells.

2.1.1. Binary Tree Algorithms

The core concept of the binary tree algorithm is to simulate the structural characteristics of a binary tree [5]. Starting from the origin, the algorithm processes each cell in sequence. For each cell, it randomly selects one wall in the direction toward the goal and removes it.

This algorithm is highly efficient and could be easily implemented. However, since the connectivity direction of each cell is fixed, the resulting maze exhibits strong directional bias and low complexity.

2.1.2. Aldous Broder Algorithm

The algorithm constructs the maze through the principle of "random access" [6]. It begins by randomly selecting a starting node. Then, it proceeds to randomly visit an adjacent node and removes the wall if the target node remains unvisited. This iterative procedure continues until all nodes have been accessed. The algorithm is easy to implement and generates mazes without specific biases. However, it may visit areas that have been already visited multiple times, resulting in a low time efficiency.

2.1.3. Recursive Backtracking Algorithm

The algorithm applies to the concept of depth-first search to maze generation [7]. It initiates from a randomly selected node. Then, it selects an unvisited adjacent node, removing the wall between the two. If the current node has no unvisited adjacent nodes, the process backtracks to the previously accessed node. This process continues until all nodes have been processed. While the algorithm is highly efficient, it requires additional memory to track visited cells. Due to its depth-first-search feature, the generated mazes tend to have fewer branching paths and more narrow and long passages.

2.1.4. Prim's Algorithm

The core concept of this algorithm lies in constructing a minimum spanning tree [1]. It begins by randomly selecting a node and adding all adjacent passages to a "candidate edge set". A passage randomly selected from this set is then removed, thereby connecting the nodes. The passages surrounding the newly connected cell are then added to the set. This process repeats until all cells are connected. The mazes generated by this algorithm has a large number of uniform branching paths, resulting in high complexity. However, the approach requires maintaining a dynamically growing list of candidate edges.

2.2. Maze Search Algorithms

2.2.1. Depth First Search

The algorithm will prioritize deep exploration along a certain path [8]. It backtracks to the most recent branching point when encountering a dead end. Its implementation takes advantage of the last-in-first-out (LIFO) feature of a stack. Firstly, the starting node is pushed onto the stack. The algorithm then iteratively pops the top element from the stack and checks whether it is the endpoint. If not, the unvisited adjacent nodes are pushed onto the stack; if yes, the process ends and returns the path. While the algorithm could find a feasible path, it cannot guarantee the shortest. Moreover, repeated exploration of incorrect branches may lead to low efficiency.

2.2.2. Breadth First Search

The algorithm explores uniformly in all directions to ensure the shortest path when reaching the destination for the first time [8]. Its implementation utilizes the first-in-first-out (FIFO) feature of a queue. Firstly, the starting node is placed into the queue. The algorithm then iteratively dequeues the element at the head of the queue and checks whether it is the endpoint. If not, the unvisited adjacent nodes are enqueued; if yes, the process ends and returns the path. This algorithm is easy to implement and guarantees finding the shortest path. However, it may require substantial memory to store the set of cells when dealing with large-scale mazes.

2.2.3. Dijkstra's Algorithm

It operates on a greedy strategy, systematically expanding paths from the starting node by always selecting the vertex with the minimal currently cumulative cost [9]. The algorithm is implemented through a priority queue. Firstly, the starting node's distance is set to 0 and all other nodes are set to infinity and enqueued. At each step, the node with the minimal cost is always chosen for evaluation to determine whether it is the target. If not, it updates the shortest distances of its neighboring nodes; if yes, the process ends and returns the path. The algorithm guarantees finding the shortest path but cannot handle graphs containing negative-weight edges.

2.2.4. A-star Algorithm

The fundamental principle behind the A-star algorithm involves guiding the search direction by combining the actual cost and the estimated cost [8]. At the heart of the algorithm is an evaluation function, $f(n) = g(n) + h(n)$. It integrates two distinct cost measures: an actual cost from the origin, $g(n)$, and an estimate of the remaining cost to the goal, $h(n)$. It employs a priority queue to progressively select nodes in order of increasing $f(n)$ value, continuing until the target node is attained. By effectively deleting irrelevant branches, the algorithm significantly improves search efficiency.

3. Results and Discussion

This section presents a systematic complexity analysis of the eight maze algorithms, with the results summarized in Table 1 and Table 2. Here, "n" represents the total count of nodes within the maze, while "m" represents the number of passages. For typical maze structures, "n" scales linearly with "m".

Table 1. Complexity analysis of maze generation algorithms

Maze generation algorithms	Space complexity	Time complexity (Average case)	Time complexity (Best case)	Time complexity (Worst case)
Binary tree algorithm	O (1)	Θ (n)	Ω (n)	O (n)
Aldous Broder algorithm	O (1)	Θ (n^2 log n)	Ω (n)	Unbounded
Recursive backtracking algorithm	O (n)	Θ (n+m)	Ω (n+m)	O (n+m)
Prim's algorithm	O (n+m)	Θ (mlogm)	Ω (mlogm)	O (mlogm)

Table 2. Complexity analysis of maze solving algorithms

Maze solving algorithms	Space complexity	Time complexity (Average case)	Time complexity (Best case)	Time complexity (Worst case)
DFS	O (n)	Θ (n+m)	Ω (n+m)	O (n+m)
BFS	O (n)	Θ (n+m)	Ω (n+m)	O (n+m)
Dijkstra‘s Algorithm	O (n+m)	Θ ((n+m)logn)	Ω ((n+m)logn)	O ((n+m)logn)
A-star algorithm	O (n+m)	Θ ((n+m)logn)	Ω (n)	O ((n+m)logn)

Table 1 summarizes the complexity of maze generation algorithms. Regarding memory usage, both the Binary Tree and Aldous-Broder algorithms require only constant space O (1). Both Recursive Backtracking and Prim's algorithms exhibit linear space requirements. In terms of time complexity, the Binary Tree algorithm maintains linear complexity O(n) in all cases, demonstrating the highest efficiency. In contrast, the Aldous-Broder algorithm demonstrates the weakest theoretical efficiency, and its worst-case time complexity is unbounded. Table 2 presents the complexity of maze-solving algorithms. All four algorithms exhibit linear space complexity. BFS and DFS require O (n) space, while Dijkstra's and A-star algorithms require O (n + m) space. In terms of time complexity, both BFS and DFS achieve O (n + m) in all cases. The A-star algorithm performs most efficiently in ideal case with a complexity of O (n). The Dijkstra's algorithm demonstrates the weakest theoretical performance, achieving O ((n + m) log n) under all conditions.

The above results demonstrate the performance differences among various algorithms, providing a theoretical foundation for practical applications. The Binary Tree algorithm requires minimal space and time for maze construction. However, it can only generate mazes with low complexity and biases. In contrast, Prim's algorithm produces mazes with the highest complexity, making it suitable for generating large-scale mazes. The Aldous-Broder algorithm also requires little space, yet its huge time cost makes it impossible for large-scale maze generation. Recursive Backtracking strikes a balance, constructing relatively complex mazes with moderate time and space requirements. For maze-solving, the A-star algorithm demonstrates the highest ideal efficiency and is widely applied. While both BFS and DFS perform competently, they are only suitable for mazes with unweighted paths. Moreover, DFS is not applicable for identifying optimal solutions. It is noteworthy that the efficiency of A-star algorithm is completely influenced by its heuristic function [10]. When the heuristic function accurately estimates the actual distance from the current node to the goal, the performance of A-star approaches its lower bound Ω(n) [10]. When the heuristic function h(n) = 0, the A-star algorithm degenerates into Dijkstra's algorithm.

Furthermore, practical applications should consider not only theoretical complexity but also factors such as maze scale and constant multipliers. For instance, despite its prohibitively high theoretical time complexity, the Aldous-Broder algorithm is rarely considered for maze design. However, it is easy to implement and has a low space complexity. When dealing with extremely small-scale mazes, the actual performance of this algorithm may be acceptable.

This study provides a theoretical foundation for algorithm selection. Developers can make trade-offs between efficiency and complexity based on specific maze requirements. Building on the analytical results, future research may advance in two following directions. Firstly, design systematic experimental performance tests. It can not only provide data support for theoretical complexity analyses but also precisely quantify the impact of constant factors on practical performance. Consequently, it can offer more accurate guidance for real-world algorithm selection. Secondly, explore algorithm combinations and integrate the characteristics of different algorithms. The limitations of a single algorithm make it unable to cope with complex real-world demands. The combination of multiple algorithms may provide better solutions for complex problems.

4. Conclusion

This study selects four maze generation algorithms and four maze solving algorithms as the research objects. Firstly, the article introduces the basic principles and characteristics of various algorithms, and then systematically analyzes their performance differences. This article provides a solid theoretical basis for algorithm selection in different application scenarios through a systematic comparison of their time and space complexity. In the future, the author plans to test the actual running time of different algorithms for different data scales through experiments. The experimental data will provide data support for theoretical conclusions and promote the selection and application of these algorithms in practical scenarios.

References

[1] Prim RC. Shortest connection networks and some generalizations. Bell Syst Tech J. 1957; 36 (6): 1389 - 401.

[2] Dijkstra EW. A note on two problems in connexion with graphs. Numer Math. 1959; 1 (1): 269 - 71.

[3] Hart PE, Nilsson NJ, Raphael B. A formal basis for the heuristic determination of minimum cost paths. IEEE Trans Syst Sci Cybern. 1968; 4 (2): 100 - 7.

[4] Babior L, Sayyadzadeh I. Optimizing Dijkstra's algorithm: Enhancing pathfinding efficiency through heuristics and structural techniques. In: Proc IEEE Syst Inf Eng Design Symp (SIEDS). Charlottesville, VA, USA; 2025. p. 313 - 7.

[5] Kim PH. Intelligent maze generation [PhD dissertation]. Columbus (OH): The Ohio State University; 2019.

[6] Wilson DB. Generating random spanning trees more quickly than the cover time. In: Proc 28th Annu ACM Symp Theory Comput. Philadelphia, PA, USA; 1996. p. 296 - 303.

[7] Tarjan R. Depth-first search and linear graph algorithms. SIAM J Comput. 1972; 1 (2): 146 - 60.

[8] Kumar N, Kaur S. A review of various maze solving algorithms based on graph theory. IJSRD - Int J Sci Res Dev. 2019; 6 (12): 431. ISSN (online): 2321 - 0613.

[9] Wang SX. The improved Dijkstra's shortest path algorithm and its application. Procedia Eng. 2012; 29: 1186 - 90.

[10] Xie CL, Gao SH, Sun XZ. Path planning algorithm fusing improved A* algorithm and Bezier curve optimization. J Chongqing Univ Technol (Nat Sci). 2022; 36 (7).