Efficiency Analysis of Three Maze-Solving Algorithms

Zhiyue Xi

Faculty of Science and Engineering, University of Nottingham Ningbo China, Zhejiang, China

Ssyzx12@nottingham.edu.cn

Abstract. Maze generation and solving have long been studied as fundamental problems in computer science and artificial intelligence, not only because of their recreational and educational value but also due to their relevance in real-world applications such as robotics navigation, game design, and optimization tasks. This study reviews real-world application scenarios of maze generation and solving algorithms and summarizes selected research advances in these areas over the past two decades. The research also evaluates the time efficiency of three classic maze-solving algorithms, which are Greedy search, Dijkstra's algorithm, and Floyd-Warshall algorithm across different data scales, revealing insights into their performance and applicability in various contexts. The algorithmic time complexity and empirical runtime are analyzed and fitted to investigate the alignment between theoretical complexity and practical performance. Overall, this work highlights the importance of connecting theoretical advances with empirical evaluation, providing guidance for both academic research and industrial practice in algorithm design.

Keywords: Maze, solving-algorithm, algorithm evaluation.

1. Introduction

Maze generation and solving techniques have a wide range of real-world applications. Maze generation algorithms could be utilized in game-based mobile learning applications to teach fundamental programming concepts. For instance, the binary tree algorithm can be used to generate mazes of varying complexity to cater to students at different learning stages, thereby aiding them in gradually mastering programming skills [1]. In game development, maze generation technology is employed to create game levels and enhance player experience. A study has proposed an automatic maze generation method, which is based on human player behavior tendencies. By simulating human players' behavior, the difficulty of the maze could be assessed and adjusted accordingly in order to match the players' skill levels and progress, which could make the game more challenging and engaging [2]. Maze generation techniques also play a significant role in Virtual Reality (VR). A maze terrain creation system has been developed, who automatically generates various complex maze patterns and converts them into 3D maze terrains, enhancing visual realism in immersive VR environments. This system is not only adaptable for traditional grid maze generation, but also enables the creation of circular mazes. It could also transform the hand-drawn maze sketches into 3D maze terrains.

During the past 20 years, both maze generation and maze solving have made notable progress. In 2011, an automatic maze generation method based on human player tendencies is proposed by Foltin [3], which employs supervised learning to study player behavior and creates test players that simulate human behavior to assess maze difficulty, then generating mazes that match the perceived difficulty of human players. In 2023, Ivanitskiy et.al explored the possibility of application of Transformer models in maze solving [4]. They found that these models can form structured internal representations of maze topology and optimal paths, and that residual streams of individual tokens can be decoded linearly to reconstruct the entire maze. In 2024, Mia et.al. introduced a novel maze solving method based on spectral graph theory [5]. By mapping the maze to a finite element grid and using the eigenvalues of the graph's Laplacian matrix to constrain connectivity between the input and output, the method minimizes the length of the optimal path.

These advancements in maze generation and solving over the past two decades demonstrate the continuous development of these techniques in both theoretical and applied aspects, providing valuable references for research and practice in related fields. However, in this experiment,

researchers mainly focus on three classic algorithms that solve given maze, and investigate the time efficiency by measuring the runtime of algorithms under certain data structures. More details related to these research methods and data structures are listed in the following section.

2. Methods

2.1. Maze Design

In this study, the maze is formally modeled as a simple directed graph $G = (V, E)$ with $|V| = n$ vertices. The algorithms under evaluation are required to compute, for any given instantiation of G, the path from vertex v_1 to vertex v_n whose cumulative edge-weight is minimal. This graph-based representation is ubiquitous in real-world applications such as navigation-system route planning and autonomous agent pathfinding; consequently, a rigorous analysis of the efficiency and relative merits of shortest-path algorithms in this context carries substantial practical significance.

2.2. Maze Generation Workflow

The maze generator deployed in this study implements the following steps.First, a graph with n vertices would be created. Then, the main path, would be created, with the starting node as v_1 and the ending node as v_n. The length of main path l_{main}, representing the number of edges included in the path, is limited as $l_{main} \in [\frac{1}{3}n, \frac{2}{3}n]$.Several side branches, would then be summoned, with the starting node as v_a and the ending node as v_b, where:

v_a is included in the main branch or the side branch that have been created and all nodes except v_a and v_b is not included in the main branch or the side branch that have been created. In this study, the total node number n and side branch number s could be inputted. Fig. 1 shows a sample output of the algorithm receiving $n = 10$, $s = 3$.

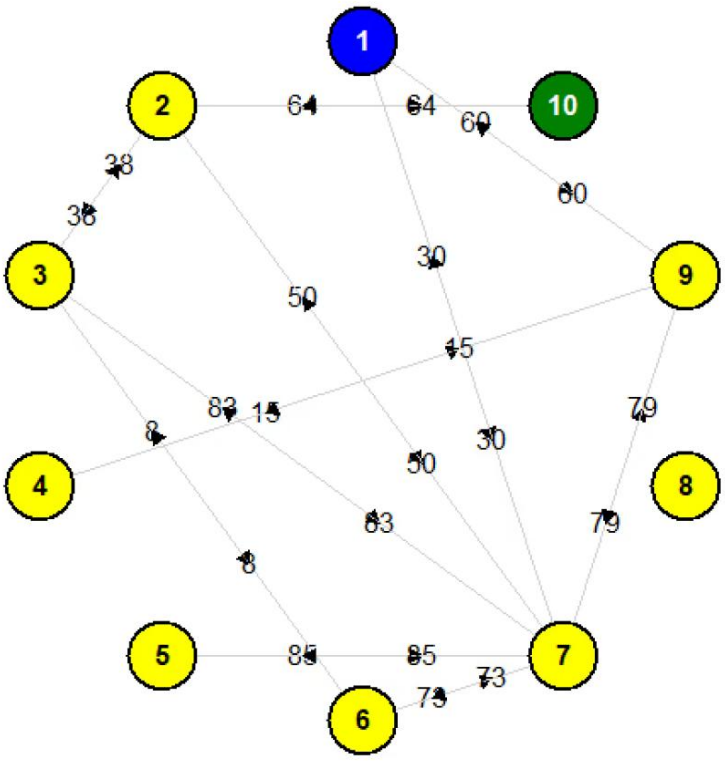


Fig. 1 Maze Visualization (Picture credit: Original)

2.3. Maze Solving Algorithms

The study implemented and tested three algorithms, which are Greedy search, Dijkstra and Floyd-Warshall algorithm. These three algorithms are classic and have been proven to be effective, and are supposed to search for a shortest path from node v_1 to node v_n

Greedy method implemented for the experimental setting is the Best-First Search that always extends the frontier vertex with the smallest current tentative cost from the source. Its merit lies in extreme simplicity—only $O(m \log n)$ with a binary heap—and often empirically fast discovery of the destination when edge weights are mildly correlated with Euclidean distance. Yet the algorithm is not optimal; it may commit irrevocably to locally promising yet globally sub-optimal arcs, yielding incorrect shortest-path costs in graphs with arbitrary or adversarial weights.

Dijkstra's algorithm is the classical label-setting method for the single-source shortest-path problem on graphs with non-negative edge weights, which is appropriate to calculate the shortest path required for this experimental scenario. In the experiment setting, it maintains a min-priority queue keyed by tentative distances from vertex v_1 and iteratively extracts the vertex of globally smallest label, relaxing all outgoing arcs. This greedy yet exhaustive expansion guarantees exact optimality, regardless of weight distribution, and returns the true minimum-cost path to vertex v_n. The algorithm is simple to implement and exhibits predictable behaviors. Its principal limitation is its blindness to the target's location, so it often explores regions irrelevant to the desired $c1$ to n path. With a binary heap the time complexity is $O((m + n) \log n)$ and the space requirement $O(n)$.

Floyd–Warshall is an all-pairs shortest-path algorithm that, in the present experimental framework, constructs an n × n distance matrix by dynamic programming. Successive iterations $k = 1 \ldots n$ allow every vertex pair (i, j) to be updated via the relaxation rule $dij = min(dij, dik + dkj)$. Once the matrix is complete, the minimum-cost path from vertex 1 to vertex n is read directly. The method is notable for its conceptual simplicity and its ability to furnish every pairwise distance in a single run, making it attractive for batch or preprocessing tasks. Conversely, its $O(n^3)$ time and $O(n^2)$ space requirements render it impractical for large graphs, and its dense matrix

3. Results and Discussion

The experiment tested the time efficiency of the three algorithms in various data scales multiple times, recording the runtime. The outcomes are shown in Table 1.

Table 1. Algorithms runtime over data scale

Data scale(node)	10	20	50	100	200	500	1000
Greedy	0.30s	1.05s	8.05s	44.90s	196.90s	1242.6s	4299.6s
Dijkstra	0.35s	1.31s	9.43s	46.21s	217.14s	1524.7s	7229.5s
Floyd-Warshall	0.14s	0.25s	12.60s	92.72s	158.58s	1310.78s	9526.9s

The runtime vs data scale plot of the three algorithms is shown in Fig. 2, Fig. 3 and Fig. 4.

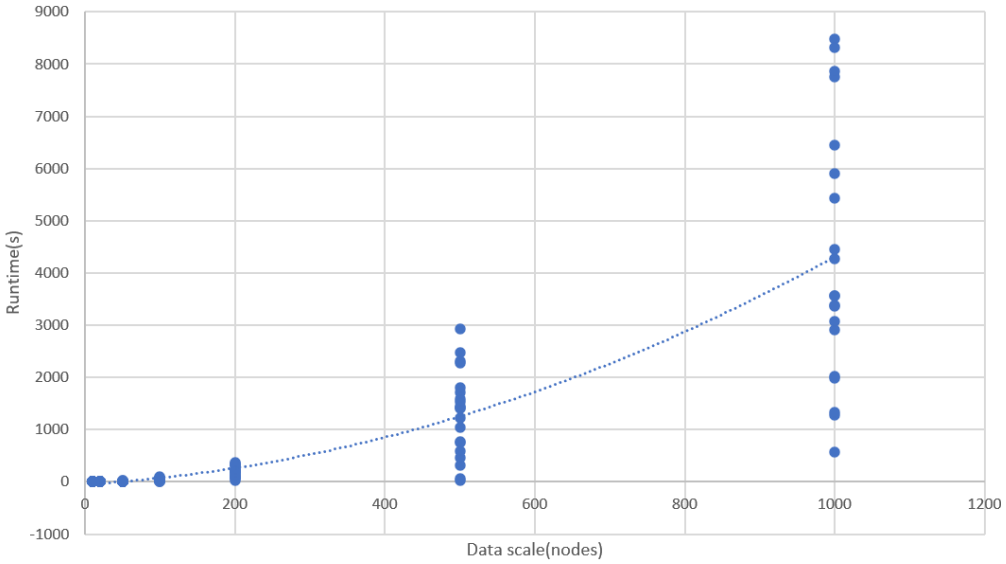


Fig. 2 Greedy algorithm runtime vs Data scale plot (Picture credit: Original)

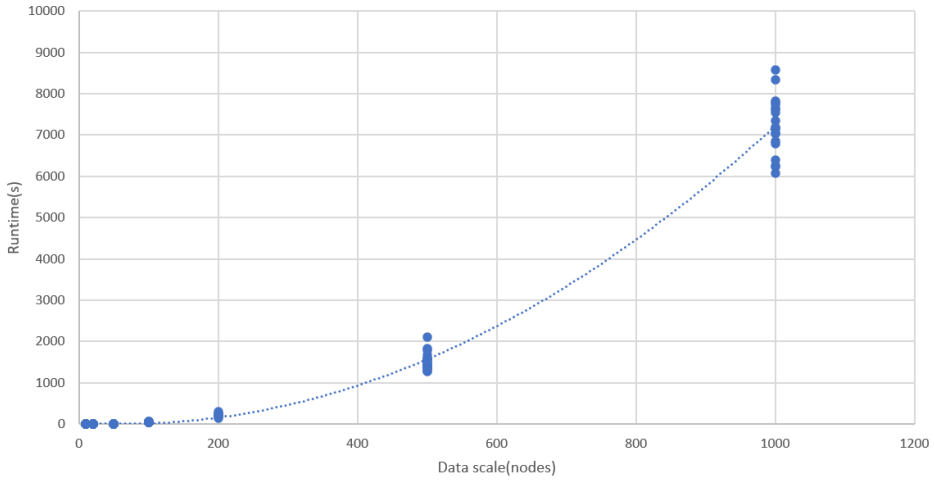


Fig. 3 Dijkstra algorithm runtime vs Data scale plot (Picture credit: Original)

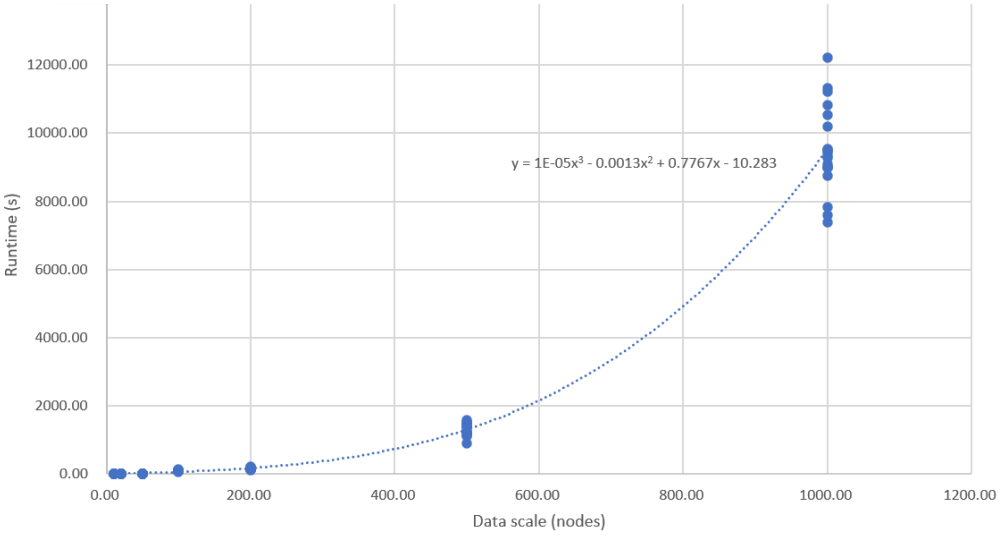


Fig. 4 Floyd-Warshall algorithm runtime vs Data scale (Picture credit: Original)

Analyzing the outcomes of experiment, it is obvious that:

Under large data scale circumstances, Floyd-Warshall algorithm is notably slower than the other two algorithms, which is as predicted. This is due to that the algorithm is designed for multi-source graphics solving, which suggests that it calculates more than necessary in the experiment environment.

In certain maze scenarios, the greedy algorithm has been observed to consume significantly less time than Dijkstra's algorithm, despite both having the same theoretical time complexity. This discrepancy may be attributed to the fact that, in specific mazes, the greedy algorithm happens to reach the endpoint with only a few steps of trial. It is important to note that the greedy algorithm does not guarantee an optimal solution but merely ensures a suboptimal one. This means that, although the greedy algorithm may sometimes be more time-efficient, Dijkstra's algorithm can be regarded as an effective alternative to the greedy algorithm across all environments.

By comparing the data points with the trend line, it can be observed that the time range for solving mazes with the same algorithm under the same data scale is relatively large. This makes it difficult to determine whether the time function fit is good. Meanwhile, attempts to predict the average running time for higher data scales based on the running time results for lower data scales have been largely unsuccessful, resulting in significant errors

4. Conclusion

The study concludes that maze generation and solving techniques have made significant strides over the past 20 years, with notable advancements in both theoretical and applied aspects. The binary tree algorithm's ability to create mazes of varying complexity has proven effective in educational settings, 9*facilitating the gradual acquisition of programming skills. The experimental evaluation of three classic algorithms—Greedy search, Dijkstra's algorithm, and Floyd-Warshall algorithm—revealed that while the Greedy algorithm can be faster in certain scenarios, Dijkstra's algorithm offers a more reliable solution across all environments. Floyd-Warshall algorithm, though powerful for multi-source problems, is less efficient for large-scale instances due to its cubic time complexity. Future research could focus on investigating the performances of other algorithms over a larger data scale. Possible optimizations on algorithms are also suggested.

References

[1] Pochtar E, Milanowski M. Comparative maze study. 2024 May 5. Available from: https://evan-pochtar.github.io/static/papers/ComparativeMazeStudy.pdf

[2] Ishikawa Y, Yoshihara T, Okamura K, Ohzeki M. Individual subject evaluated difficulty of adjustable mazes generated using quantum annealing. 2023 Nov 10. Available from: https://arxiv.org/pdf/2309.04792

[3] Foltin M. Automated maze generation and human interaction. Brno: Masaryk University Faculty of Informatics; 2011. p. 10-24.

[4] Ivanitskiy IM, et al. Structured world representations. 2023 Dec 5. Available from: https://www.kimi.com/chat/d2ohkqls8fb4o03j87f0

[5] Čarapina M, Staničić O, Dodig I, Cafuta D. A comparative study of maze generation algorithms in a game-based mobile learning application for learning basic programming concepts. Algorithms. 2024 Nov 10;17(9):404. Available from: https://www.mdpi.com/1999-4893/17/9/404 (Accessed 2025 Aug 31)