Proceedings of CONF-APMM 2026 Symposium: Applied Mathematics and Quantitative Modeling
DOI: 10.54254/2753-8818/2026.PU35188

Maze Generation and Structural Simplification for Maze Solving: A Graph Theoretical Viewpoint

Bolin Lyu

College of Information Engineering, Changchun College of Electronic Technology, Changchun, China
abc18704493001@gmail.com

Abstract. Maze generation and maze solving are two well-known issues in computer science. At first glance they seem to be easy puzzles, but in fact, they involve many concepts from graph theory and algorithm design. This paper introduces some common methods for maze generation, such as Binary Tree algorithm, Recursive Backtracking and the Aldous–Broder algorithm. These methods generate mazes in various ways, thus the final structures of the mazes differ in terms of randomness, bias and complexity. Concerning maze solving, traditional search methods like Breadth-First Search (BFS) and Depth-First Search (DFS) are briefly mentioned. However, the main attention is paid to the Dead-End Filling algorithm. Unlike BFS or DFS, the Dead-End Filling does not search for a direct path. It eliminates the dead ends step by step until the remaining part shows the solution path. From a graph viewpoint, this procedure can be considered as simplifying the graph structure instead of exploring possible routes. The discussion indicates that the method used for generating a maze has an important effect on its solving way. This is particularly obvious for the reduction-based methods like Dead-End Filling, since their performance depends largely on the distribution of dead ends in the maze. In perfect mazes, where the maze structure is equivalent to a spanning tree, the Dead-End Filling works most effectively. In summary, this paper reviews several typical generation and solving methods and expounds their connections from a graph-theoretical perspective.

Keywords: Maze solving, maze generation, BFS, DFS

1. Introduction

Maze generation and solving have been researched for a long time in computer science. Though mazes are usually considered as games or puzzles, the related algorithms are closely related to graph theory, combinatorial optimization and general algorithm design. In actual applications, maze-like structures also occur in robot navigation, game maps, artificial intelligence pathfinding and even network routing problems. Therefore, the maze algorithms are not only interesting theoretically, but also applicable in many fields. Previous approaches to solving mazes were quite rudimentary. For example, following the walls might be suitable for some easy mazes, especially when the maze is single-connected. But in cases involving loops or complicated structures, these methods may not be effective or do not satisfy the requirements. Hence, ordered graph traversal algorithms like Breadth-

© 2026 The Authors. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (https://creativecommons.org/licenses/by/4.0/).

DOI: 10.54254/2753-8818/2026.PU35188

First Search (BFS) and Depth-First Search (DFS) were frequently used. BFS can guarantee the shortest path in an unweighted graph, while DFS usually needs less memory. However, both algorithms have the same disadvantage: they find the solution by examining all possible paths systematically. Maze generation has been changed in some aspects. Some techniques are easy and fast but lead to uneven distribution of mazes, whereas others create less biased or more regularly arranged mazes with larger computation times. Binary Tree method, Recursive Backtracking and Aldous–Broder are typical cases. The mazes generated by these methods show different appearances as well as structures. Such structural differences might influence the difficulty in solving the maze and the running time of a solving algorithm. Moreover, there is an alternative method for solving mazes. The Dead-End Filling algorithm is an example. It does not solve the maze by searching from the entrance to the exit as usual. It deletes the paths which cannot be part of the final solution. In a perfect maze, all the dead-end branches can be removed gradually and the leftover structure will be the only path from beginning to end. Therefore, this paper discusses some basic maze generation and solving algorithms, particularly Dead-End Filling. The aim is not to describe these algorithms in depth, but to clarify their operations, the structural properties they possess and how they can be analyzed in a graph-theoretical context.

2. Methodology

2.1. Maze generation methods

Maze generation is generally regarded as a graph problem. In a common grid maze, each cell can be treated as a vertex and the possible connection between two neighboring cells can be regarded as an edge [1, 2]. Therefore, a maze becomes a graph located on a two-dimensional grid [3].

A usual kind of maze is the perfect maze. In a perfect maze, there is only one way between any two cells, which indicates that the maze has no loops and no separated areas. From the viewpoint of graph theory, it is similar to a spanning tree, since all vertices are connected and there are no cycles [4, 5].

Several generation algorithms create various kinds of maze structures. Some methods are fast but may have some influences, while others give more random results but require more computation time [1, 2, 6]. In this paper, three common techniques are discussed: Binary Tree, Recursive Backtracking and Aldous–Broder.

The Binary Tree algorithm is a basic method for generating mazes. For each cell, it deletes either the northern wall or the eastern wall randomly. Due to its simple rule, the algorithm is convenient to implement and works in linear time. Nevertheless, this simplicity leads to an obvious disadvantage. As the algorithm only selects from a few directions, the produced maze frequently exhibits a strong directional tendency. Consequently, the result may include numerous regular corridor-like structures [1, 6].

Recursive Backtracking is connected with depth-first search. It begins from a cell, chooses an unvisited neighbor and removes the wall between them. If there are no unvisited neighbors, it moves to an adjacent cell and selects another one [2, 7]. Usually, this approach generates complex and branched paths, whereas simpler and less biased mazes are constructed by Binary Tree [1, 2].

The Aldous–Broder algorithm employs a unique technique. It explores the graph randomly. A cell which is unvisited and linked by an unused edge is added into the maze [4, 5]. This process can generate uniform spanning trees, so the obtained result is more theoretically fair. However, the algorithm may visit some cells several times before visiting all the cells of the graph. Because of this repeated inspection, it is usually less efficient than simple generation methods [4, 8].

DOI: 10.54254/2753-8818/2026.PU35188

Binary Tree is advantageous but biased. Recursive Backtracking has more complicated structures with reasonable efficiency. Aldous–Broder is more random and theoretically uniform, but it takes longer time [1, 6, 8].

2.2. Maze solving approaches

Maze solving can be regarded as a graph problem. Generally, solving a maze involves finding a route from one vertex to another: the entrance and the exit. Therefore, maze-solving algorithms are usually connected with graph traversal, path planning and optimization [9, 10].

Traditional methods such as BFS and DFS solve the problem by examining possible paths. BFS investigates level by level, while DFS searches deeply into one branch before checking other branches [9, 11]. These techniques are feasible and general, but they require searching the graph.

Dead-End Filling works in another manner. It does not find the correct route directly, but excludes those sections of the maze which certainly cannot be the solution. In graph theory, it removes some branches from a tree-like structure [3, 5]. Hence, this approach is especially suitable for perfect mazes, because a perfect maze has no loops and only one path exists between the beginning and the end [4, 5].

2.2.1. Dead-end filling: structural reduction perspective

The main idea of Dead-End Filling is straightforward. In a perfect maze, if a path terminates at a dead end, this path cannot be included in the final route from the entrance to the exit. Thus, its removal will not affect the correct solution. When the maze is regarded as a tree, a dead end is equivalent to a leaf node. Removing the leaf nodes which are not the starting or ending point does not destroy the unique path between these two fixed points [3, 5].

A maze can be depicted by an undirected graph $G = (V, E)$. In this depiction, each cell is a vertex and each open passage between cells is an edge. A dead end is a vertex with degree one, which only has one connection to the rest of the maze [3, 5].

The algorithm examines and eliminates these isolated vertices. After the removal of a vertex, it might happen that another cell turns into an isolated vertex. This procedure is carried out until no further deletions can be done. In a completely connected maze, the leftover network is the solution backbone, indicating the single path from the beginning to the end [4, 5].

This is why Dead-End Filling ought not to be considered as a usual search method. It does not "search for" the path like BFS or DFS does. On the contrary, it deletes all the incorrect paths. The desired path will be revealed after eliminating the unnecessary branches.

2.2.2. Algorithmic characteristics

Dead-End Filling has some characteristics which distinguish it from the classical graph search algorithms. BFS and DFS often have to choose the next node to be inspected, thus their efficiency is related to the traversal sequence. Dead-End Filling is less influenced by the traversal sequence. The configuration of the maze, particularly the positions of the dead ends, is more significant [9, 11].

First, Dead-End Filling does not carry out direct route searching. It does not begin from the entrance and try various paths until it reaches the exit. Conversely, it operates from the outer unnecessary branches towards the inside.

Moreover, this method depends on repeated pruning. After eliminating an unneeded section, new opportunities might appear, so the algorithm gradually simplifies the graph until it is completed.

DOI: 10.54254/2753-8818/2026.PU35188

Third, the algorithm is founded on topology. Its performance is influenced by the graph structure instead of a particular search method.

Finally, in perfect mazes, the intersection is determinate. As the maze is similar to a spanning tree, deleting all the unnecessary branches will ultimately remain the unique route from the starting point to the end point [4, 5].

This method is attractive because it alters the nature of the problem. The maze solving is no longer a pathfinding task but turns into a graph simplification task. It enhances the comprehension. In instructional or demonstrative systems, the removal of wrong paths may be more understandable than the search algorithm examining many branches [12].

2.2.3. Comparison with BFS and DFS

BFS, DFS and Dead-End Filling can be applied to solve mazes, but their thinking methods are quite distinct.

BFS examines the graph step by step. It can determine the shortest path in an unweighted maze graph. This is an advantage, especially when the path optimality is important. However, BFS has to save a great amount of frontier nodes at the same time, which needs much memory in larger mazes [9, 10].

Depth First Search (DFS) searches along one branch till the end and then returns. It usually consumes less memory than Breadth First Search (BFS). However, DFS may waste much time in investigating long and wrong branches before locating the solution. Moreover, DFS cannot ensure the shortest path [9, 11]. Recursive Backtracking is closely connected to DFS, thus the mazes produced by Recursive Backtracking usually have several long and complex paths [2, 7].

Dead-End Filling does not fall into the same category. It does not develop a new direction, nor does it select a route to investigate. It only eliminates the branches without solutions. Consequently, it is more appropriately termed as an elimination or reduction method.

Another difference is the visual interpretability. BFS and DFS present the way a path is explored. While Dead-End Filling presents the correct path by removing the incorrect parts, which is beneficial in visualization and educational fields, where it is necessary to understand the solving procedure [1, 2].

2.2.4. Theoretical advantages and limitations

In regard to its efficiency, Dead-End Filling is very simple. Each node can be deleted at most once. Thus, the entire process needs a linear time, usually denoted as $O(|V|)$, where $|V|$ indicates the number of cells or vertices in the maze. This characteristic is advantageous for huge perfect mazes.

Moreover, this method has some restrictions. It functions most effectively in a perfect maze. When the maze includes loops or several valid paths, eliminating the dead ends may not be sufficient to determine a unique solution. In some cases with loops, there might be a small number of dead ends or several remaining paths after pruning.

This algorithm is applicable to different kinds of graph search methods. BFS, DFS, Dijkstra's algorithm and A* can be utilized in various graph forms, like cyclic graphs and weighted environments [9, 10, 13]. Nevertheless, Dead-End Filling is more specialized. It is effective and easy to understand, but it is not so general as these searching approaches.

Although there is this restriction, Dead-End Filling is still beneficial. It offers a definite method to link maze solving with graph structure. It is also helpful in demonstrating the importance of the

DOI: 10.54254/2753-8818/2026.PU35188

uniqueness of paths in perfect mazes. Some recent investigations have explored maze solving by means of other graph-based concepts, for example, spectral graph theory and topological optimization [14].

2.2.5. Summary of methodological significance

From the above discussion, Dead-End Filling is considered a graph reduction method instead of a conventional searching approach. Its benefit is not only in dealing with perfect mazes, but also in providing another way to solve mazes.

It transforms the question of "how to find the correct route?" into "which portions of the graph cannot be a part of the correct route?" This new viewpoint is significant. It also simplifies the observation of the solving process, since the solution becomes evident as the incorrect branches disappear.

The method fits well with spanning-tree based maze structures also. As perfect mazes are connected to spanning trees, Dead-End Filling works particularly suitable for this kind of maze [4, 5, 8]. Therefore, it can be considered as a connection between practical maze solving and fundamental concepts in graph theory.

3. Results and discussion

3.1. Comparative characteristics of maze generation methods

This investigation does not rely on experimental evidence or benchmark results. Nevertheless, the maze generation techniques can be assessed according to their structural features. According to previous works, various maze generating methods produce different maze structures, which influence both the maze complexity and solving behavior [1, 2, 6].

The Binary Tree algorithm mainly constructs mazes with certain directionality. Since it deletes walls only in some directions, the maze has many regular corridors and thus a simpler and more regular structure. Although these mazes may be easier to explore, their randomness is limited [1, 6].

Recursive Backtracking generates more intricate and separated patterns. When the algorithm arrives at the situation requiring a backtrack, the maze usually has long corridors and a branched structure. Therefore, the maze is more complicated than a Binary Tree maze. There is a relation between Recursive Backtracking and DFS, which frequently forms long passages before returning to the previous position.

Aldous–Broder produces a special type of structure. It is based on random walks and can generate uniform spanning trees, thus the generated maze is usually less predictable. The connections in this maze are more randomly arranged than in Binary Tree and Recursive Backtracking mazes [4, 5]. However, this randomness has a disadvantage, since the algorithm may require many steps by retracing its paths through the previously visited cells [4, 8].

These differences are significant as maze generation and solving are related. An algorithm for solving cannot operate in an empty area; it functions on the structure generated by the generation algorithm. Consequently, the method used for generation may influence the difficulty and behavior of the solving process [1, 6].

DOI: 10.54254/2753-8818/2026.PU35188

3.2. Structural behavior of dead-end filling

Dead-end filling is closely related to the topology of the maze. When the algorithm removes the dead ends, it is affected by the positions and branch lengths of the dead ends. From a graph perspective, finally all the leaf nodes are removed until the solution backbone remains [3, 5].

In the Binary Tree mazes, dead ends usually occur in certain parts due to the overall directional bias of the maze. Therefore, the pruning process may proceed rapidly towards the main remaining path. In most situations, the solution structure appears relatively quickly [1, 6].

In the Recursive Backtracking mazes, the situation is likewise different. Dead ends may occur at the end of lengthy and profound branches. Thus, the filling procedure may be carried out in several stages. Some short branches vanish earlier, while the deep branches need more steps to be completely eliminated. This is similar to the depth-first feature of the generating process [2, 7].

In the Aldous–Broder mazes, the dead ends are distributed more randomly. Since the maze is constructed by random walks and uniform spanning-tree method, the pruning procedure is less localized. Dead ends may appear in different parts of the maze, therefore the simplification process may require more steps until the solution backbone is clear [4, 5, 8].

Though the visual process varies among different maze types, a node is deleted at most once, which maintains the algorithm as linear with respect to the number of cells. It is more computationally advantageous than some other algorithms which may require keeping a large search frontier or exploring numerous unnecessary branches, such as in perfect mazes [9, 10].

3.3. Algorithmic interpretation and comparative insights

Dead-End Filling plays a significant role in the classification of maze-solving algorithms. It is not a less efficient variation of BFS or DFS. It bases on a different viewpoint of the problem.

BFS and DFS solve a maze through exploration. BFS spreads out from the starting point and can get the shortest path in an unweighted graph, but requires more memory [9, 10]. On the other hand, DFS examines the deeper part of one branch and usually saves memory, but may waste time in unnecessary branches and cannot always find the shortest path [9, 11].

Dead-End Filling searches for the right path in the maze by eliminating the incorrect parts until only the correct part is left. This indicates that maze solving can be viewed as either the discovery of the path or the reduction of the graph.

This method is suitable for perfect mazes. As a spanning tree, there is only one route connecting the starting point with the ending point. After eliminating all the dead-end branches, this route is retained [4, 5]. Nevertheless, it also indicates the disadvantage of Dead-End Filling. If the maze is not perfect and has cycles, multiple feasible routes may still exist and removing the dead-ends may not be sufficient.

Unlike Dijkstra's algorithm or A*, Dead-End Filling is not very flexible. Dijkstra's algorithm and A* are more suitable for weighted graphs, dynamic environments or cases where the path cost is significant [13]. Dead-End Filling is less adaptable, but it is also easier to understand both in its structure and visualization. It is mainly applied to static perfect mazes, to seek the solution by elimination [13, 14].

3.4. Key observations

Several observations can be drawn from the comparison.

DOI: 10.54254/2753-8818/2026.PU35188

First, the method of generating the maze has a significant influence on the solving process. Various algorithms for generating the maze produce different distributions of dead ends, branches and corridors, therefore the behavior of Dead-End Filling also varies [1, 2, 6].

Second, Dead-End Filling is principally based on the structure rather than the visiting order. This is opposite to BFS and DFS, in which the node visiting sequence is important [9, 11].

Ancient mazes may function under certain circumstances. A perfect maze is equivalent to a spanning tree and a spanning tree ensures that any two vertices are connected by exactly one path. Therefore, after removing the unnecessary branches, the solution path is still preserved as [4, 5].

Furthermore, Dead-End Filling has some disadvantages. It is easy to use, clear and fast for solving perfect mazes, but it is not as universal as BFS, DFS, Dijkstra's algorithm or A* when there are cycles or several paths exist [9, 10, 13].

Finally, maze solving is not confined to the traditional traversal methods. Recent studies using spectral graph theory indicate that maze problems can also be investigated by means of graph Laplacians and topology-based methods [14].

3.5. Summary of discussion

The discussion shows that maze generation and maze solving should be regarded as a whole. The generation algorithm constructs the maze, whereas the solving algorithm deals with it. Therefore, the same solving method may give different results for mazes generated by different algorithms.

Dead-End Filling is an example of this kind of relationship. It indicates that it is not necessary to use a direct path search to solve a maze. By taking advantage of the graph's properties, the solution can be obtained through a sequential elimination process. This provides another feasible way to solve mazes.

There is no best algorithm which is appropriate for all situations. BFS is suitable for finding the shortest path. DFS is preferable when less memory is needed. Dead-End Filling is particularly advantageous for perfect mazes and when the clarity of vision or structure is important [9-11]. Hence, the selection of algorithm for solving the maze should be based on both the characteristics of the maze and the purposes of the application.

4. Conclusion

Maze construction and finding can also be investigated by using graph theory. In this paper, three kinds of generating methods have been introduced: Binary Tree, Recursive Backtracking and Aldous–Broder. They produce various kinds of maze structures. The Binary Tree is easy to implement, but may not be regular. The Recursive Backtracking generates more branching mazes, and Aldous–Broder makes mazes with greater randomness and theoretically uniform spanning trees theoretically.

BFS and DFS are used for traditional search methods in solving mazes. BFS is suitable for determining the shortest path, whereas DFS generally requires less memory. Nevertheless, both methods require the exploration of the maze graph actively.

Dead-End Filling offers an alternative solution to the problem. It does not trace the route directly. Instead, it eliminates the dead-end nodes repeatedly until only the solution backbone is left. This approach is particularly efficient in perfect mazes since their graph pattern is a spanning tree with a single path from the starting point to the end.

At the same time, Dead-End Filling is not applicable to all kinds of mazes. In cyclic or imperfect mazes, several possible paths may occur, thus eliminating dead ends may not give a unique clear

DOI: 10.54254/2753-8818/2026.PU35188

solution. Nevertheless, the approach is still useful as it is easy, effective and convenient to understand.

In general, this paper indicates that maze solving can be considered not only as a pathfinding issue, but also as a graph simplification problem. Dead-End Filling is an obvious example of this viewpoint, which facilitates the relationship between the practical methods for solving mazes and fundamental graph-theoretical concepts.

References

[1] Gabrovšek, P. (2019). Analysis of maze generating algorithms. IPSI Transactions on Internet Research, 15(1), 23-30.

[2] Li, J. (2025). Maze Generation Algorithms: A Comparative Research with a Focus on Recursive Backtracking in OCaml. In Proceedings of the 2nd International Conference on Engineering Management, Information Technology and Intelligence (EMITI 2025), 432-437.

[3] Even, S., & Tarjan, R. E. (1975). Network flow and testing graph connectivity. SIAM journal on computing, 4(4), 507-518.

[4] de Oliveira Nunes, I. (2020). *A Study of Random Walk Based Algorithms for Efficient Generation of Uniform Spanning Trees* (Doctoral dissertation, Universidade Federal do Rio de Janeiro).

[5] Kim, P. H. (2019). *Intelligent maze generation*. The Ohio State University.

[6] Čarapina, M., Staničić, O., Dodig, I., & Cafuta, D. (2024). A comparative study of maze generation algorithms in a game-based mobile learning application for learning basic programming concepts. *Algorithms*, *17*(9), 404.

[7] Wang, X. (2024). OCaml-based Recursive Backtracking and Aldous Broder algorithms in solving maze challenges. In *AIP Conference Proceedings* (Vol. 3194, No. 1, p. 050011). AIP Publishing LLC.

[8] Yang, K., Lin, S., Dai, Y., & Li, W. (2024). Optimization and comparative analysis of maze generation algorithm hybrid. *Applied and Computational Engineering*, *79*(1), 20-33.

[9] Shi, L. (2024). Research on path planning method based on graph search algorithm. *Highlights in Science, Engineering and Technology*, *97*, 267-274.

[10] Emelianov, S. G., Bobyr, M. V., & Kryukov, A. G. (2022). Research of the properties of the breadth-first search algorithm for finding the movement route of robots. Southwestern State University, 40.

[11] Cong, G., Kodali, S., Krishnamoorthy, S., Lea, D., Saraswat, V., & Wen, T. (2008). Solving large, irregular graph problems using adaptive work-stealing. In *2008 37th International Conference on Parallel Processing* (pp. 536-545). IEEE.

[12] Sharma, M., & Jindal, H. (2022). Pathfinding Visualizer. http: //www.ir.juit.ac.in: 8080/jspui/bitstream/123456789/3757/1/Pathfinding%20Visualizer.pdf

[13] Vijaya, J., Baghel, A. K., & Mishra, A. S. (2024). Dynamic Maze Solver: Integrating Advanced Algorithms for Optimal Path-finding in Varied Environments. In *2024 IEEE International Conference on Interdisciplinary Approaches in Technology and Management for Social Innovation (IATMSI)* (Vol. 2, pp. 1-6). IEEE.

[14] Martín-Nieto, M., Castaño, D., Horta Muñoz, S., & Ruiz, D. (2024). Solving Mazes: A New Approach Based on Spectral Graph Theory. *Mathematics*, *12*(15), 2305.