A Comparative Study of Maze-Solving Algorithms: Breadth-First Search, A and Binary Tree with Depth-First Search

Youye Wang *

Shanghai Qibao Dwight High School, Shanghai, China

* Corresponding Author Email: yywang_jerry@qibaodwight.org

Abstract. Maze solving has been an important problem in computer science for a long time, which is used as a model for searching, optimization, and path finding. This study compares three representative algorithms: Breadth-First Search (BFS), A*, and Binary Tree combined with Depth-First Search (DFS). And the study tests the time they cost of different sizes of mazes. Experimental results show that all methods perform similarly in small size mazes with the execution times under 0.1 seconds. However, as maze size increases, BFS and A* are able to maintain efficiency and scalability while the method of Binary Tree + DFS becomes significantly slower and impractical for large size mazes. BFS ensures the shortest solution path but it requires higher memory at the same time. A* achieves a balance between optimality and efficiency through heuristic functions. Binary Tree & DFS provides educational value in demonstrating recursive searching though less scalable, Overall, the study concludes that BFS and A* are more applicable to the real-world path finding problems while Binary Tree & DFS is more suitable for teaching structural search processes than the other 2 methods.

Keywords: BFS, A*, binary tree.

1. Introduction

Mazes solving is a traditional problem in computer science field since it provides an efficient and useful model of methods of searching, optimizing and choosing. By optimizing the methods, some basic algorithms for further research can be further strengthened. This is related to many industries such as robotics navigation, computer games, network routing, and artificial intelligence planning. In general, mazes are presented by a 2-D rectangle which is divided into lots of single units.

In the previous study, many methods have been created to solve the mazes [1]. Wall Follower, a relatively simple method to solve maze. It is very easy to understand but has a low efficiency. Breadth-First search, first created by Edward F. Moore, is one of the famous methods that can generate the shortest path naturally [2]. Although it has high memory cost, it still provided lots of ideas based on BFS. The A* algorithm explored by Peter Hart, Nils Nilsson and Bertram Raphael in 1968, which is introduced as a heuristic search method, combines the path cost and heuristic estimation together, which makes the method have both efficiency applicability [3]. What' more, Binary tree has also been explored by Cayley. These methods are mainly used in maze generation and relative structure representation, which provides a simple model for searching process [4]. In addition, there are some more difficult methods to solve the mazes. For example, Dijkstra's Algorithm developed by Edsger W. Dijkstra in 1956 is used to solve mazes which have different weight in different sizes [5].

This study will mainly focus on the three algorithms—Breadth-First Search, A*, and Binary Tree and try to evaluate their effectiveness in solving mazes. This paper will implement these algorithms in Ocaml and design some mazes for experiments to compare their accuracy and efficiency. By testing the experiment, the study aims to identify not only the theoretical advantages of each method but also the limitations of each algorithm.

2. Methods

2.1. Breadth First Search

BFS is the most common graph traversal algorithms [6]. Its basic logic is to search the whole maze level by level. In this way, any time the program goes to the next level, it must have completed all the units in the previous level so that the result is naturally the quickest path among several solutions once the solution is first found.

The principle of BFS relies on a queue data structure, which is used to take and store the units which are planned to be searched. The logic of queue that always takes the first and add at the end support the ability of naturally generating best solution [1].

The key feature of BFS is its guarantee of always finding the shortest path in 'unweighted' mazes, which makes it highly accessible in many scenarios. However, its drawbacks are that it makes high memory costs since it needs to store huge number of units in the queue. But even so, BFS is still a famous algorithm of solving mazes and provides a base for other algorithms like A* [7].

2.2. A*

The A* algorithm is a classical heuristic search method [3]. It is widely used to solve maze and plan the route. Its basic idea is to combine the actual path cost with an estimated cost towards the goal, thus making a balance between optimality and efficiency. For each unit, A* defines a cost function:

$f(n)=g(n)+h(n)$,

where $g(n)$ represents the actual cost from the start node to the current node, and $h(n)$ represents an estimate of the cost from the current node to the goal [8].

The principle of the A* is based on a priority queue. During the process of searching, the program will more likely to go towards the units which have less cost than others. As a result, A* can quickly approach the goal with relatively low memory cost. Meanwhile, it can ensure that the solution path is relatively optimal in some conditions.

The key feature of A* is obviously the balance of the optimal solution and efficiency (memory cost). Although the performance of A* largely depends on the heuristic function design, it is still one of the most important algorithms for maze solving and path finding research [9, 10].

2.3. Binary Tree

Binary tree is a basic hierarchical data structure in computer science [11]. It's widely used in the areas like representing ordered relationships, managing dynamic sets of data, and supporting efficient searching. Its main idea is based on the theory that each node can have at most two child nodes called the left child and the right child. As a result, it forms a branch structure [11].

In usual, the child nodes always obey some specific rules. For example, in a binary search tree, the left child is usually smaller than the parent node, and the right child contains larger values. Thus, this arrangement allows more efficient searching, insertion, and deletion, making the time complexity be often in logarithmic time.

The principle of binary tree is based on recursive decomposition. A binary tree can be defined as a root node with two subtrees, and each of the subtrees again is a root node with two subtrees. By using systematic strategies, all binary trees can be easily visited and searched.

Its key feature is that it can balance between simplicity and power. On one hand, it can provide strong and clear level structure to support many. On the other hand, it can guarantee the high efficiency of searching and updating the data. Therefore, the binary tree is one of the most important models in data structure and algorithm research.

Since binary tree itself is just a method to record the structure of the maze, the program needs a Depth first searching (DFS) to deal with the mazes in the binary tree form [8]. DFS can effectively search all the units and won't leave out any units.

3. Result and Discussion

This Table 1 shows the time taken by three maze-solving method: Breadth-First Search (BFS), A*, and Binary Tree + Depth-First Search (DFS)—across different maze sizes.

Table 1. Time consuming of the 3 methods

Size oft he maze	BFS(s)	A*(s)	Binary tree & DFS(s)
5*5	0.068	0.074	0.066
20*20	0.065	0.072	0.064
100*100	0.078	0.092	0.412
500*500	0.262	0.496	125
1500*1500	1.925	5.35	/

For small mazes (5×5, 20×20), all the three methods perform similarly, with the time under 0.1 seconds. In this condition, Binary Tree + DFS is a little faster than other methods. As maze size increases to 100×100, BFS (0.078s) and A* (0.092s) still remain efficient, while Binary Tree + DFS time rises significantly to 0.412s. In large mazes (500×500 and above), BFS and A* increases relatively, while Binary Tree + DFS becomes extremely slow (125s for 500×500) and cannot handle 1500×1500 mazes within practical time.

The difference in size is mainly because of the algorithm the program uses. In BFS, it goes through all the units in the mazes and ensures the output is the optimized path while the memory is relatively high [6]. Based on BFS, A* adds a heuristic function to decrease the units which need to be search [3]. However, it will slightly increase the work of calculation on every single unit. This makes it perform well in larger size mazes. The methods of binary tree plus DFS search every path in the maze and recall if the path is the solution. This method doesn't perform well in the mazes of this study [12].

Binary tree + DFS is not good at the scalability and is relatively inefficiency because binary tree is mainly used to solve the problem which fits a binary choice. BFS and A*, meanwhile, are relatively better in this study's mazes although the memory cost is larger than the binary tree + DFS.

To further improve, it is possible to switch to graph representations with adjacency lists instead of strict binary trees. And also, the study can develop a more complex heuristic function for A* to improve its performance in choosing the better direction.

In conclusion, binary tree and DFS are more suitable for educational computer science demonstrations while BFS and A* can be actually used in solving mazes in larger size due to their scalability and efficiency.

4. Conclusion

In conclusion, this study compares the three mazes solving methods, which is Breadth-First Search (BFS), A*, and Binary Tree & Depth-First Search (DFS). After testing the time they take for different sizes of mazes, the results indicate that all the three algorithms perform efficiently on small mazes, but their efficiency differs significantly when the size of the maze increases. BFS consistently ensures that the output is always the shortest path and can maintain a stable performance while BFS still requires high memory usage. A*, which is another maze solving method, balances efficiency and optimality by integrating heuristic functions, which enable it to become more adaptable to larger mazes than BFS. In contrast, Binary Tree & DFS perform well in simple and effective for small mazes, but they demonstrate poor efficiency and are time-consuming as maze size grows, becoming impractical for large-size maze solving problems.

Overall, the finding highlights that BFS and A* are more suitable for real-world applications such as robotics navigation and AI path planning since they have great balance of efficiency and reliability. Binary Tree + DFS, however, just remains valuable in educational contexts as it can clearly demonstrate recursive searching processes and binary structural modeling. For future improvements, the study may include optimizing heuristic functions for A* and trying to explore alternative graph representations to further enhance and develop the performance of maze solving.

Moreover, it is important to recognize that maze-solving algorithms are not only theoretical exercises but also play a crucial role in practical applications. In autonomous systems such as drones, self-driving cars, and robotic exploration in hazardous environments, pathfinding efficiency and reliability are essential. The adaptability of algorithms like A makes them particularly attractive for real-time problem solving where quick decision-making is required under limited computational resources. On the other hand, BFS provides a theoretical guarantee of optimality, which makes it valuable in scenarios where accuracy is prioritized over speed, such as network routing or mapping unknown environments. Furthermore, incorporating parallel computing and hardware acceleration could significantly enhance algorithm performance when handling large-scale mazes or real-world navigation grids. Future research could also explore hybrid approaches that combine the advantages of different algorithms, leveraging BFS's reliability, A's flexibility, and even the conceptual clarity of Binary Tree + DFS for specialized cases. This integrative perspective not only deepens the understanding of algorithmic design but also points toward practical frameworks that align with the rapid growth of artificial intelligence and robotics technologies. Ultimately, continuous optimization and cross-disciplinary exploration will ensure that maze-solving methods remain relevant and impactful across both educational and industrial domains.

References

[1] Cormen T H, Leiserson C E, Rivest R L, Stein C. Introduction to Algorithms. 3rd ed. Cambridge: MIT Press, 2009.

[2] Moore E F. The shortest path through a maze. In: Proceedings of the International Symposium on the Theory of Switching. Cambridge: Harvard University Press, 1959.

[3] Hart P E, Nilsson N J, Raphael B. A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics, 1968, 4 (2): 100 – 107.

[4] Cayley A. On the theory of the analytical forms called trees. Philosophical Magazine, 1857, 13 (85): 172 – 176.

[5] Dijkstra E W. A note on two problems in connection with graphs. Numerische Mathematik, 1959, 1: 269 – 271.

[6] Wikipedia. Breadth-first search [EB/OL]. https://en.wikipedia.org/wiki/Breadth-first_search, 2025 - 09 - 12.

[7] GeeksforGeeks. Time and space complexity of DFS and BFS algorithm [EB/OL]. https://www.geeksforgeeks.org/dsa/time-and-space-complexity-of-dfs-and-bfs-algorithm/, 2025 - 09 - 12.

[8] Dinh H Q, Dinh H T. Inconsistency and accuracy of heuristics with A* search. arXiv preprint arXiv: 1307.2200, 2013.

[9] Agostinelli F, McAleer S, Shmakov A, Baldi P. Obtaining approximately admissible heuristic functions. arXiv preprint arXiv:2106.01473, 2021.

[10] Prieditis A E. Machine discovery of effective admissible heuristics. Machine Learning, 1993, 12 (1 – 3): 117 – 141.

[11] Knuth D E. Fundamental Algorithms. The Art of Computer Programming, Vol. 1. 3rd ed. Boston: Addison-Wesley, 1997.

[12] Wikipedia. Depth-first search [EB/OL]. https://en.wikipedia.org/wiki/Depth-first_search, 2025-09 - 12.