Proceedings of CONF-APMM 2025 Symposium: Applied Mathematics and Quantitative Modeling
DOI: 10.54254/2753-8818/2026.PU35202

Comparative Performance Analysis of BFS and Left-Hand Rule Algorithms for Autonomous Maze Navigation in OCaml

Qile He

Shanghai Pinghe School, Shanghai, China
le66880205@gmail.com

Abstract. Compare the Performance of Global and Local Navigation Paradigms in Autonomous Spatial Path Planning in this Study. The first purpose is to explore the efficiency of algorithms, memory scaling and structure robustness for discrete maze-solving agents built in functional programming languages. Mechanistically, in this way, binary weights are used to convert a continuous environment into a structured two-dimensional grid matrix, the starting coordinate of the entrance is fixed at the lower-right corner, and the target destination is placed on the upper-left boundary of the exit. The two frameworks for modelling path planning are as follows: First, a state-exhaustive Breadth-First Search (BFS) tracking model is employed to use a first-in, first-out queue and a boolean tracking matrix for uniform radial, layer-by-layer exploration; second, a localized Left-Hand Rule algorithm is used, which is structured as a memoryless Finite State Machine that executes a deterministic relative turning priority queue based on real-time directional vectors, prioritising relative left turns and straight movements over relative right turns and backward turnarounds. Benchmarking in OCaml shows that BFS ensures the shortest path by exploring uniformly outwards but is space-intensive; the Left-Hand Rule is very memory-efficient but may lead to non-optimal routing or cyclic deadlocks in multiply connected topologies. BFS is very suitable for resource-intensive applications that require high precision in the end, and the Left-Hand Rule provides an efficient and light-weight paradigm only for simply connected geometric boundaries.

Keywords: Autonomous Path Planning, Breadth-First Search, Left-Hand Rule, OCaml

1. Introduction

Maze-solving is a representative mathematical model of path planning in the study of discrete mathematics and modern operations research. From the standpoint of mathematics, a two-dimensional maze can be rigorously mapped into a planar rectangular coordinate system and represented as a set of matrices containing a finite number of discrete points [1-3]. At each of these coordinate points in the set, the state is either that of a passable road or an impassable obstacle, and this value is binary. The goal of the mathematics is to find an optimal sequence of continuous coordinates (that is, the shortest path) from the start coordinate to the end coordinate under the given step size rule, namely, moving only one step at a time to the adjacent coordinates up, down, left, and right. This class of problems in combinatorics, which is based on a grid coordinate system, has many

DOI: 10.54254/2753-8818/2026.PU35202

practical applications in real life, such as optimizing navigation routes for vehicles and allocating resources in logistics networks [3].

Two types of mathematical methods with different degrees of difficulty are used for solving the path-finding problem of such coordinate graphs. The first strategy is blind search based on global state traversal, and Breadth-First Search (BFS) is the most typical. The mathematical process of BFS is similar to the spread of water waves; from a source node, it spreads out in successive layers based on distance to construct a loop-free tree-like hierarchical structure, also known as a tree diagram, within a mathematical system [4]. Since this way will cover all possible path branches systematically, it can be shown that the globally optimal shortest path for an unweighted grid will be found. However, the cost of this strategy is a large space overhead due to the need to store a considerable number of explored coordinate states in a queue [2, 4]. On the other hand, the second option is a reactive rule based on local geometric constraints, often called the "Left-hand Rule". Geometrically, it is strictly necessary that the maze walls have no internal closed loops and all walls are connected to the outermost boundary for this rule to apply. Given the above geometric constraints, the algorithm does not need to know the entire map information and only needs to follow the one-dimensional boundary line for trajectory tracking; thus, it has a very small space overhead [5]. However, a prominent defect of the Left-Hand Rule is that it fails to guarantee a shortest path, and moreover, when there are isolated closed-loop walls in the maze, the rule will lead to infinite looping and thus a mathematical dead end.

To compare the two mathematical strategies with considerable differences in space overhead and path optimality systematically, this paper introduces a general comparison of the BFS algorithm and the Left-Hand Rule. The two algorithms are both implemented in the functional programming language OCaml, and to conduct a quantitative efficiency test under the same conditions, a control variable method (Fair Testing Conditions) is used; that is to say, the same maze matrix, a starting point of the bottom-right corner, and an end point of the top-left corner are selected [6]. The following chapters of this paper are organized as follows: Chapter 2 introduces the method, presents the transformation of a two-dimensional maze into a mathematical discrete matrix model, analyses the construction logic of tree branches in the BFS algorithm and the local turning priority decision-making of the Left-Hand rule, and provides their rigorous code implementations in OCaml. Chapter 3 provides and analyses the quantitative experimental data under fair evaluation conditions, and compares the path length and the number of exploration nodes. Finally, Chapter 4 will present the main results of this study and put forward some practical suggestions that are easy to understand for high school students in breaking the cycle of deadlocks in the left-hand rule.

2. Method

2.1. Transformation of two-dimensional maze

To perform computational path-planning algorithms, a continuous physical or graphical maze must first be discretized into a formal structured environment that a program can process. In this study, the spatial topology of a two-dimensional maze is formalized into a discrete $M \times N$ matrix, representing a grid-coordinate mathematical system where M denotes the total number of rows and N represents the total number of columns. This structural abstraction translates geometric routing constraints into a pure data science problem, enabling discrete graph traversal operations.

To rigorously evaluate the efficiency and scalability of the path-planning algorithms under different workloads, various environmental scales are configured within the evaluation system. These dimensions range from small-scale setups (such as a 10×10 matrix used for baseline logical

DOI: 10.54254/2753-8818/2026.PU35202

validation) to larger, high-density configurations (such as a 20×20 matrix or higher). Documenting these scaling dimensions is critical because as the values of M and N increase, the total state space expands quadratically ($O(M \times N)$), which drastically increases the execution overhead and tree depth for global search methods.

The transformation workflow relies on a binary encoding scheme where each coordinate position (i, j) in the matrix is assigned a specific boolean or integer weight. Unobstructed paths and passable corridors are uniformly mapped to the value 0 (or a boolean false), indicating that an algorithm can legally transition into this state. Conversely, static obstacles, walls, and dead ends are strictly encoded as the value 1 (or a boolean true), marking them as impassable nodes.

Finally, strict boundary constraints are established within the matrix coordinate logic. Any computational move resulting in an index out of bounds (where $i < 0$, $i \geq M$, $j < 0$, or $j \geq N$) is structurally classified as an impassable element equivalent to a wall, preventing the program from throwing out-of-bounds exceptions during execution. The explicit coordinate mappings for the unique starting node (located at the bottom-right corner, $(M-1, N-1)$) and the destination target node (located at the top-left corner, $(0, 0)$) are definitively locked into the matrix configuration, ensuring identical evaluation environments for both evaluated algorithms.

2.2. Breadth-first search algorithm

2.2.1. Core concept and mathematical foundation

BFS algorithm is a foundational framework in graph theory that is widely utilized for the systematic exploration of tree or graph structures [7-9]. In maze-solving scenarios, the discrete matrix generated during the transformation stage is mathematically abstracted as an undirected graph, denoted as $G = (V, E)$, where V represents the set of passable vertices (coordinate states with a value of 0) and E represents the effective edge set connecting adjacent passable coordinates. The core concept of BFS relies on a "radial" or "layer-by-layer" traversal strategy. Rather than exploring a single path branch in depth, the algorithm originates from the source node and extends evenly outward, thereby systematically scanning the entire spatial search domain. From a mechanical perspective, this propagation process resembles the physical movement of water waves or ripples on a surface. Since the edges in a standard grid maze possess identical mathematical weights (meaning the cost of each movement step is equal to 1), this layer-by-level progression provides a rigorous mathematical guarantee for global optimality: the first time the destination target node is discovered, the generated coordinate sequence is structurally guaranteed to be the shortest path.

2.2.2. Algorithm principle and specific execution tasks

The following are the specific computational steps for conducting this detailed exploration, using an explicit first-in, first-out (FIFO) queue for data storage and maintaining a state. The sequence of functions in the workflow is as follows:

Initialization Task: The algorithm initializes an empty FIFO queue and enqueues the starting coordinates $(M-1, N-1)$ as the root element. Concurrently, a boolean matrix of identical dimensions (designated as the "Visited Matrix") is established to record the explored states, with the position corresponding to the starting coordinates initialized to "true". To reconstruct the final path upon termination, it is also necessary to instantiate a reference structure or parent-node pointer mapping table to store the predecessor of each discovered node.

DOI: 10.54254/2753-8818/2026.PU35202

State Exploration Loop: The system enters a continuous loop that executes as long as the FIFO queue is not empty. In each iteration, a coordinate is removed (de-queued) from the front of the queue and designated as the current active node. This active node is immediately compared and evaluated against the target destination coordinates $(0, 0)$. Once the target is identified, the exploration stage terminates, and the control flow shifts to the backtracking routine.

Transfer and Verification of Neighboring Nodes: If the current node is not the target, the algorithm performs local exploration to calculate the coordinates of its four direct neighbors based on legal directional displacement vectors: upper $(-1, 0)$, lower $(+1, 0)$, left $(0, -1)$, and right $(0, +1)$. Each candidate neighbor node must pass three strict verification checks: it must be located within the grid boundaries, its matrix encoding value must equal 0 (passable), and its corresponding entry in the "Visited Matrix" must be marked as "false".

En-queuing and Path Mapping: Neighbouring nodes that have successfully passed all verification conditions are immediately marked as "true" in the Visited Matrix to avoid processing them multiple times. Subsequently, these nodes are appended to the end of the queue (en-queued), and their parent-child relationships are logged in the pointer mapping table.

This detailed state-recording method comes with significant spatial costs. Because the algorithm must simultaneously store a large volume of frontier coordinate branches and historical tracking data within the queue and the visited matrix, its space complexity can reach $O(M \times N)$ in the worst-case scenario.

2.2.3. Practical application scenarios

Besides discrete mathematical models and maze-solving simulations, the basic ideas of the BFS model are used widely in current computer science and network engineering. A typical application scenario is the routing protocol for a decentralized network, such as the OSPF mechanism for network broadcasting and packet routing, which needs to find a path with the minimum number of hops in a complex topology. BFS is also a typical algorithm used by web crawlers for index construction; it repeatedly retrieves links at different depths via this algorithm to build a complete map of the network. The two fields where this algorithm is frequently used in data science and social network analysis (SNA) are to determine the "degree of separation" among individuals and to construct maps of user interaction paths and structural connection clusters based on the theoretical foundation of the "Six Degrees of Separation".

2.3. The Left-Hand Rule algorithm

2.3.1. Geometric principle and reactive mechanism

BFS and other global- and state-exhaustive search methods are not used; instead, the Left-Hand Rule is a local geometric-constraint-based reactive heuristic wayfinding method. The essential mathematics for this law does not use a global map of the world; instead, it adopts a local perception method to follow the one-dimensional boundary line of a two-dimensional manifold. The algorithm is based on the geometric assumption that if the explorer stays in contact with the left wall of the maze at all times and the topology of the environment satisfies a certain geometric condition, its path will eventually reach the exit boundary for the traveler. The maze needs to be "simply connected", that is, all internal wall segments must be connected continuously to the outside edge of the matrix, and there should be no isolated, floating structural loop.

DOI: 10.54254/2753-8818/2026.PU35202

Since the Left-Hand Rule makes decisions entirely according to the local environment near the current location, it functions as a memoryless system [10]. It does not require a complex visited matrix or historical tracking structures to store past paths. Therefore, the space complexity of this strategy is fixed at $O(1)$ in the worst-case scenario, representing an extremely low memory overhead compared with global traversal methods. However, this local response characteristic also brings an obvious trade-off: because the law lacks a global perspective on the network layout, it cannot guarantee path optimality and frequently generates suboptimal and tortuous paths.

2.3.2. Finite State Machine, specific tasks, and hyperparameter configuration

To formally execute this tracking logic within the digital grid matrix, the algorithm is modeled as a Finite State Machine (FSM). The internal state of the system at any given step is defined by the coordinate pair (i, j) and the orientation vector indicating the current heading direction. The structure of its sequential tasks is defined as follows:

Initialization Task and Initial Hyperparameter: The system sets the active coordinates to the starting position $(M-1, N-1)$. At this stage, the initial orientation vector must be clearly designated as a foundational configuration hyperparameter (for example, the orientation must be initialized as "North" or "West"). As a baseline reference variable, the initial orientation directly affects the initial turning behavior and the generation of subsequent paths.

Directional Priority Evaluation: At all active cells, a fixed, hard-coded direction priority queue is employed for evaluation according to the current direction vector. To simulate the continuous behaviour of "left-hand wall-following", the relative steering priorities are as follows: [Relative Left Turn, Straight, Relative Right Turn, Back]. Calculate the change in absolute matrix coordinates for the above relative directions according to the current orientation state via an algorithm.

Wall Detection and State Transition: The algorithm checks the matrix values of candidate adjacent cells sequentially according to the priority list. The system selects the first direction within the legal boundaries where the matrix value equals 0 (i.e., a passable channel). Subsequently, the active coordinates are updated to the new cell, and the internal direction vector rotates to match the selected absolute direction. If the test results of all three forward options (left, straight, right) equal 1 (i.e., walls), the system switches its direction state to backward, thus performing a 180° dead-end U-turn.

Termination Condition Check: The aforementioned cyclic process continues until the active coordinates coincide with the target endpoint $(0, 0)$.

2.3.3. Environmental constraints, task failure, and deadlock

Although the omission of the historical memory saves space, if this cannot meet the fundamental geometric requirements, it will be seriously impacted in operation. If the maze matrix is a "multiply connected" topology and contains isolated walls that are not connected to the boundary (commonly referred to as "islands"), this local tracking mechanism will be completely invalid.

When the starting or ending point is located in an area surrounded by an isolated island-like wall structure, the direction priority cycle mechanism will inevitably cause the state machine to keep circulating in an internal loop indefinitely. Since the algorithm does not save the history of access, it cannot recognise whether it has returned to a state it was in before. This deficiency results in a catastrophic failure state, namely continuous cycling; consequently, a permanent mathematical deadlock arises from which the termination conditions can never be met. To process these

DOI: 10.54254/2753-8818/2026.PU35202

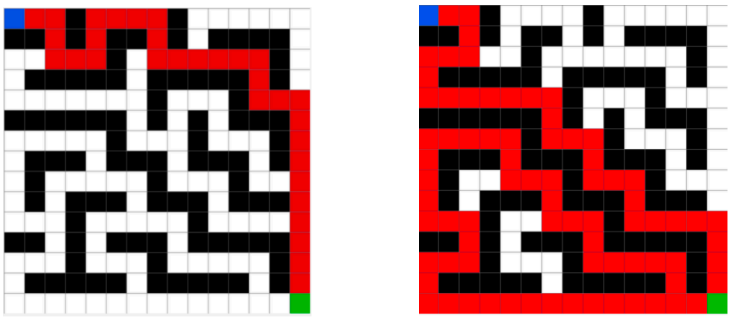
topological boundaries, auxiliary rules or heuristic enhancement means need to be introduced to detect and break cyclical deadlocks, as will be detailed in the following sections.

3. Results and discussion

To evaluate the operational efficiency and pathfinding behaviors of the BFS and Left-Hand Rule algorithms under control variable principles, both strategies were executed within identical two-dimensional matrix environments rendered via the RGB color model. Quantitative evaluation metrics focused on path length, node exploration density, execution time, and algorithmic stability across different topological layouts. The benchmarked experimental data and performance comparison under a standard 10 × 10 dimension maze are summarized in Table 1.

Table 1. Quantitative performance metrics of BFS and Left-Hand Rule in a maze

Evaluation Performance Criteria	Breadth-First Search (BFS)	The Left-Hand Rule
Exploration Strategy	Level-by-level Search (Queue)	Reactive wall-following
Final Path Length (Steps)	14	32
Total Explored Nodes	35	15
Approximate Execution Time (ms)	12.4	3.1
Guarantees Shortest Path	Yes	No
Memory Usage	Higher	Lower
Risk of Looping	No	Possible (requires step limit)
Algorithmic Success Rate (%)	100%	100% (Simply Connected Only)



(a) (b)

Figure 1. Comparison of pathfinding trajectories within a standard 10 × 10 maze environment: (a) globally optimal shortest path generated by BFS; (b) suboptimal and redundant wall-following trajectory generated by the Left-Hand Rule (picture credit: original)

As shown in the quantitative parameters in Table 1, under identical testing conditions, these two computational strategies produced completely divergent path trajectories (Figure 1). In the standard 10 × 10 matrix structure, the Breadth-First Search (BFS) algorithm systematically determined the absolute shortest path, terminating the entire process in exactly 14 steps. In sharp contrast, due to its lack of global spatial perception capabilities, the Left-Hand Rule produced a highly redundant and tortuous trajectory requiring 32 steps, representing a path length expansion of approximately 128.6% compared to the optimal path. However, this optimization was premised on a heavy cost regarding resource consumption: BFS had to evaluate a total of 35 state nodes within its first-in,

DOI: 10.54254/2753-8818/2026.PU35202

first-out (FIFO) queue mechanism, whereas the strategy based on localized wall-following only needed to explore 15 nodes to successfully complete the path solution. This empirical validation demonstrates that while BFS secures structural optimality, the Left-Hand Rule maintains a clear advantage with lower resource occupancy in terms of immediate node memory overhead.

Therefore, according to the above experiments, the expense of obtaining the full World History in BFS is too high. BFS spreads outwards from the source node in the form of a wave. Based on the above-mentioned layer-by-layer mathematical tracking method, the shortest-path optimum can be obtained by taking all combinations of coordinates. However, maintaining the FIFO queue and the historical visited state matrix leads to a considerable memory overhead, and BFS is thus extremely memory-intensive for large-scale matrix environments, although it remains stable in operation. On the other hand, the reactive mechanism of the Left-Hand Rule does not use history and is therefore very spatially efficient. However, a lack of global vision is also not ideal. Since the first objective is to draw a short local curve, the autonomous agent frequently meets very long dead-end sections and needs to reverse multiple times before reaching an available exit. Moreover, in the case of multiply connected topologies with isolated island walls, the system will enter a catastrophic permanent cycle. To prevent an infinite loop in the evaluation process, a reasonable upper bound for the number of steps will be set to forcibly end the loop. This phenomenon is in line with the traditional maze-solving literature, and thus the reactive heuristic strategy is still very unreliable in non-simply connected geometric structures.

A primary limitation of the current research lies in the inherent rigidity of both traditional paradigms. While BFS faces immediate memory saturation during high-dimensional scaling, the Left-Hand Rule fails basic structural robustness checks. To mitigate these deficiencies, future research can integrate advanced Artificial Intelligence (AI) technologies and machine learning architectures into pathfinding systems. For example, by utilizing deep reinforcement learning frameworks such as Q-learning or Deep Q-Networks (DQN), autonomous agents can interact dynamically with complex maze topologies. By introducing a heuristic reward function reminiscent of the A^* search mechanism, the model could successfully strike a viable balance between the low-memory benefits of local boundary tracking and the global accuracy of state-exhaustive searches. Neural network predictors could intelligently identify structural dead ends and dynamically break cyclical deadlocks without maintaining an exhaustive tracking queue, thus pointing out a promising direction for the development of high-performance spatial navigation systems.

4. Conclusion

BFS and the Left-Hand Rule algorithms were implemented and tested in OCaml in this paper to analyze their path-planning performance for a discrete two-dimensional maze environment. Theoretically, the above algorithms have different advantages in terms of trade-offs among other characteristics. BFS can find the shortest path and is complete; however, it has a relatively large memory usage and explores a larger number of nodes. The Left-Hand Rule algorithm is an exceptionally lightweight, memoryless model; however, it is prone to suboptimal routing and permanent cyclic deadlocks in multiply connected topologies. Therefore, BFS is suitable for complex situations that require a very precise optimal path, and the Left-Hand Rule is suitable for navigation environments with limited memory and low overhead in simply connected geometric areas.

DOI: 10.54254/2753-8818/2026.PU35202

References

[1] Gross, J. L., Yellen, J., & Anderson, M. (2018). Graph theory and its applications. Chapman and Hall/CRC.
[2] Russell, S., & Norvig, P. (2021). Artificial Intelligence: A Modern Approach, 4/E.
[3] LaValle, S. (2006). Planning algorithms: Cambridge university press, 2006.
[4] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). Introduction to algorithms. MIT press.
[5] Niemczyk, R., & Zawiślak, S. (2019, April). Review of Maze Solving Algorithms. In Engineer of the XXI Century: Proceedings of the VIII International Conference of Students, PhD Students and Young Scientists (p. 239). Springer.
[6] Madhavapeddy, A., & Minsky, Y. (2022). Real World OCaml: Functional programming for the masses. Cambridge University Press.
[7] Banerjee, N., Chakraborty, S., Raman, V., & Satti, S. R. (2018). Space efficient linear time algorithms for BFS, DFS and applications. Theory of Computing Systems, 62(8), 1736-1762.
[8] Khuller, S., & Raghavachari, B. (2010). Basic graph algorithms. In Algorithms and theory of computation handbook: general concepts and techniques (pp. 7-7).
[9] Banerjee, N., Chakraborty, S., & Raman, V. (2016). Improved space efficient algorithms for BFS, DFS and applications. In International Computing and Combinatorics Conference (pp. 119-130). Cham: Springer International Publishing.
[10] Cai, J., Wan, X., Huo, M., & Wu, J. (2010). An algorithm of micromouse maze solving. In 2010 10th IEEE International Conference on Computer and Information Technology (pp. 1995-2000). IEEE.