# A Study on the Performance Based on Aldous-Broder Randomness and A* Structured Generation

Junyang Li *

Department of Hertfordshire, Changzhou Institute of Technology, Changzhou, China

* Corresponding Author Email: 23260315@czu.cn

**Abstract.** The objective of this research was to analyze how different maze generation algorithms manage the trade-off between efficiency and randomness. Maze generation is significant in artificial intelligence, robotics, and game design, where performance and (structural variety directly influence outcomes. The methods involved implementing Aldous-Broder and A* algorithms in OCaml on standardized grid-based mazes. Aldous-Broder relies on random walks to generate uniform spanning trees, while A* applies heuristic-driven search to guide wall removal. The experiments were conducted across various maze sizes, with execution time, branching factor, and corridor length as core metrics. Each configuration was tested through repeated trials to capture variance, and results were summarized using median values and dispersion analysis. This design ensured fairness in comparison while emphasizing scalability and structural differences. The results show that Aldous-Broder achieves uniformity but at high computational cost, whereas A* provides faster generation but creates biased layouts. The findings highlight the inherent trade-off between structural neutrality and efficiency, pointing to hybrid solutions as a promising direction.

**Keywords:** Maze generation, algorithm efficiency, heuristic methods.

## 1. Introduction

In Ocaml, a maze can be used to represent a two-dimensional grid of cells, where the cells are surrounded by four potential walls. Algorithmic rules are used to remove shared walls and make connections between cells. This formal definition can be used for a wide range of traversal and generating strategies, from totally random to very structured. Maze algorithms can be used for more than just fun; they can also be used to train Artificial Intelligence (AI), fix robot navigation problems, and provide procedural material for games. They can help learning about the basic ideas of graph theory and search optimization by studying them. They also help people think about the trade-offs between speed, unpredictability, and structural control in computing.

Computer scientists have been working on making mazes for decades, and they have come up with a lot of different algorithms, each with its own set of rules. The binary tree approach, which is popular because it is simple and works in O(n) time, removes the north or west wall of each cell by moving across the grid only once [1]. It is fast, but it makes diagonal variations that are easy to foresee and are not very complicated. Again, Recursive Backtracking (depth-first search) is brought up. It makes maze corridors that are lengthier and have fewer branches, but it also has a directional bias and uses more stack space [2].

Comparative evaluations evaluated the Prim, Kruskal, Hunt-and-Kill, and Ellers algorithms, assessing runtime scalability, dead-end frequency, and complexity indices [3]. These experiments revealed a constant trade-off: faster algorithms generally result in reduced structural variation, whereas slower algorithms can produce more intricate layouts.

The Aldous-Broder (AB) algorithm has a different strength: it makes sure that any potential labyrinth configuration has the same chance of happening by making uniform spanning trees [4]. Its random wandering method may cause it to visit a lot of the same cells, which makes its generation time uncertain in big grids [5].

Recent advancements in this domain surpass conventional graph-based techniques. For instance, in the case of Ms. Pacman, genetic algorithms have been able to create different, replayable mazes,

showing how flexible heuristic-driven generation can be [6]. These methods show that a mix of targeted heuristics is needed to deliberately manage how hard and complicated a maze is to navigate.

In this research, divergent methodologies were integrated: Aldous-Broder's statistical fairness and a modified A* generator that utilizes heuristic guidance to A* variation, on the other hand, changes the maze's layout by dynamically deciding which walls to remove based on a heuristic function. For example, it might utilize the Euclidean distance for lengthy corridors or the complexity-maximization heuristic for dense cul-de-sacs. By working on the AB algorithm, adjusting A* for generation, finally came up with experiments to assess their efficiency, complexity, and unpredictability.

This study includes code for the Aldous-Broder algorithm, which makes mazes without bias, and the A* search method was changed to work with heuristic-based labyrinth building. This paper also indicates algorithmic integration in Ocaml, custom heuristic design, and metrics-based evaluation (runtime, randomness, path complexity) to enable direct comparisons of their efficiency and structural results.

## 2. Method

### 2.1. Design of Maze

This research employs a grid-based maze structure implemented in OCaml, which is especially conducive for functional programming experiments. The maze's dimensions, grid size, and how the walls are shown inside it identify it. An array of Boolean values shows if there are walls between cells that are next to each other in each maze. The OCaml version starts with barriers in place and only takes them down when the chosen algorithm tells it to. Parameters include: Dimensionality: primarily 2D mazes (10×10, 50×50, 100×100, etc.). Core representation: adjacency encoded as arrays for efficient access. Execution environment: experiments were run on a standard x86-64 architecture using OCaml’s built-in randomization functions.

The architecture of the maze makes sure that all algorithms can be compared fairly. Every algorithm begins with a completely walled maze and removes barriers until a spanning tree appears. This makes sure that everything is connected without any cycles. The framework gives a common way to measure execution time, complexity, and scalability.

This design choice lets users combine algorithms that are based on different ideas, like Aldous-Broder's random-walk-based methods and A*'s heuristic-based methods. It also looks like earlier efforts that modeled mazes as spanning trees, which makes it easy to compare performance [4, 7].

The OCaml code fixes the labyrinth representation to a direction-indexed wall array so that testing and removing walls are both O (1) operations that may be done again and over. Mixed-radix arithmetic is used to calculate node and wall indices. This avoids per-step allocation and works with any number of dimensions, even though the experiments are mostly on 2D grids. For N cells and d dimensions, the memory footprint is O (d·N) booleans. This means that both techniques use the same amount of memory. Timing employs Unix. gettimeofday, conducting 30 trials for each size with a standardized PRNG seed schedule; results present the median and interquartile range to mitigate outlier sensitivity. The study also collects structural descriptors that previous research has linked to maze complexity, such as average corridor length and branching factor [8]. This uniform design separates algorithmic differences while keeping the spanning-tree interpretation that is typical in the literature [4, 7].

### 2.2. Aldous-Broder Algorithm

The process begins at a randomly selected cell [9, 10]. At each step, a random neighbor is chosen: I f the neighbor has not been visited, the wall between the current cell and the neighbor is removed, and the neighbor is marked as visited. If the neighbor has already been visited, no modification occurs.

The program then goes to the next cell and does the same thing until all of the cells have been visited. Aldous-Broder's shown in Fig. 1 key benefit is that it guarantees an unbiased uniform spanning tree, which means that any potential spanning tree of the graph has an equal chance of being

made. However, its drawback is inefficiency. The algorithm could take a long time to visit all the cells because random walks often travel back to cells that have already been visited. In theory, the temporal complexity is O(N), but the constant factors are big [4].

To make things more efficient, people have suggested optimizations such biasing neighbor selection or combining Aldous-Broder with Wilson's loop-erased random walk [11]. This study prioritizes faithful implementation over optimization for comparative analysis.

Practical tuning adheres to the uniform spanning tree promise. Degree-aware neighbor ordering or directional bias can save cover time, but they would also violate uniformity [9, 10]. As a result, the implementation restricts optimizations to engineering decisions that do not influence the stationary distribution of the Markov chain: precomputed neighbor lists, a compact visited bitmap, and a lightweight loop for progress diagnostics [4, 11]; nonetheless, this work retains the pure Aldous-Broder approach to emphasize the balance between theoretical neutrality and empirical efficiency.

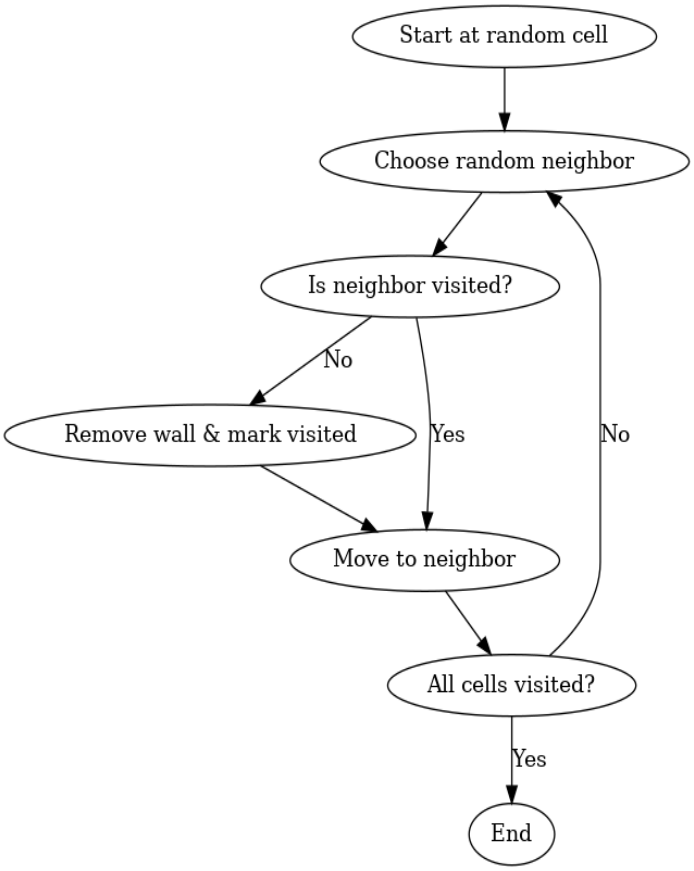


**Figure 1.** The process of Aldous-Broder Algorithm (Picture credit: Original)

### 2.3. A* Algorithm

The A* algorithm shown in Fig. 2 is widely recognized for heuristic search [12]. In maze generation, A* is adapted to remove walls while traversing from a random start to a random target. Each node maintains three components: g(n): cost from the start node, h(n): heuristic estimate of the cost to the target, f(n) = g(n) + h(n): total priority score.

A priority queue is utilized to find the node with the lowest f (n). The wall between the current node and a neighbor cell that hasn't been visited yet is taken down, the neighbor is recorded as visited, and the queue is filled with the neighbor. This process goes on until every cell has been visited. The performance is greatly affected by the choice of heuristic. This work employs Manhattan distance with a minimal randomization factor to guarantee variability in wall removal, so averting predictable patterns.

A* is better than Aldous-Broder because it uses guided search to find reachable neighbors more quickly, which cuts down on unnecessary steps. But its O (N log N) complexity shows how hard it is

to keep priority queues up to date. A* is usually faster for medium-sized mazes, although it can slow down when the heuristics aren't very accurate or when unpredictability makes things less efficient [13].

The priority queue is a randomized treap that lets inserting and deleting items in O (log N) time while eliminating external dependencies. To avoid lengthy straight corridors, ties are broken by h and a short random term r. The Manhattan heuristic is acceptable and consistent on grids [10]. To isolate heuristic advice from stylistic variability, this study changes r in [0,1] and maintain h fixed. As r gets closer to 0, the generation process gets faster but more biased. As r gets bigger, the guidance gets weaker and the performance gets closer to uninformed search, which is similar to what Zhou and Hansen found about the relationship between heuristic strength and overhead. These settings make it clear how A* balances speed, structure, and bias in real life.

Improvements include better heuristics, pruning, and hybrid methods that combine uniformity guarantees with guided search [8]. A* shows good efficiency in this study, but it can't be scaled up.

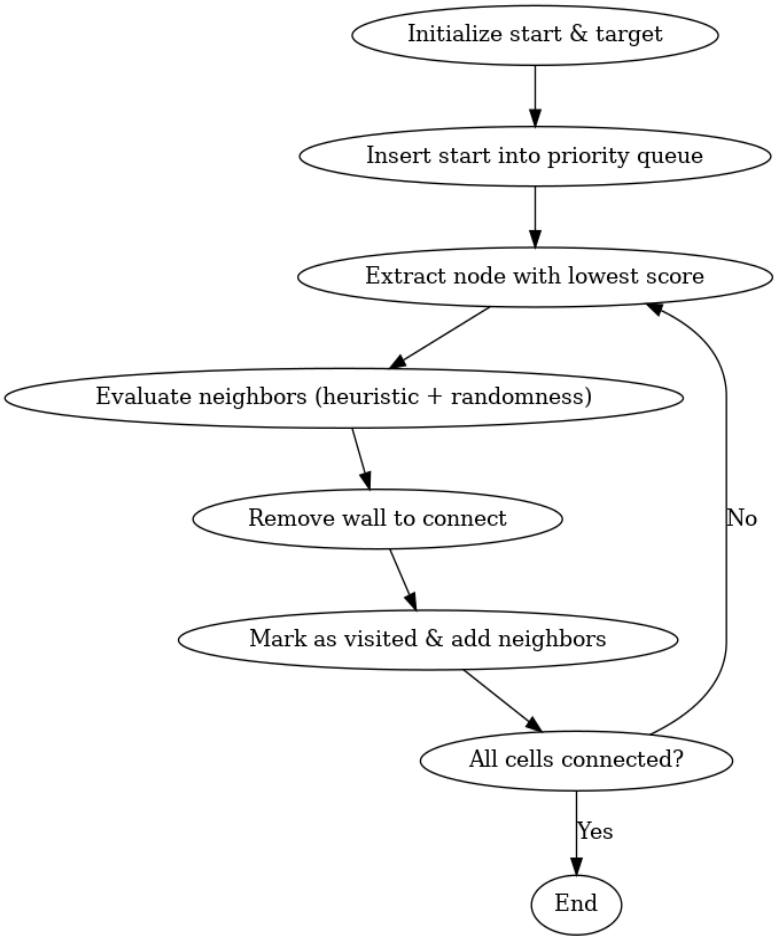


**Figure 2.** The process of A* Algorithm (Picture credit: Original)

## 3. Results and Discussion

Table 1 summarizes execution times across different maze sizes.

**Table 1.** The execution times across different maze sizes.

| Algorithm Name | 10*10 (s) | 50*50 (s) | 100*100 (s) | 10*10*10 (s) | 10*10*10*10 (s) |
|---|---|---|---|---|---|
| The recursive backtracking algorithm | 0.000133 | 0.001400 | 0.004530 | 0.000618 | 0.008597 |
| The binary tree algorithm | 0.000008 | 0.000252 | 0.000680 | 0.000163 | 0.001731 |
| The Aldous-Broder algorithm | 0.000144 | 0.001544 | 0.011600 | 0.000822 | 0.021197 |
| The A-star algorithm | 0.000467 | 0.043281 | 0.630153 | 0.008310 | 1.101364 |

Fig. 3 shows how long it took each of the algorithms to run on average for varied maze sizes. It shows more clearly how Aldous-Broder, A*, Recursive Backtracking, and Binary Tree change as the maze gets bigger. The table gives exact numbers, but the figure shows how A* grows faster for large mazes than Aldous-Broder and other approaches, which expand more slowly.

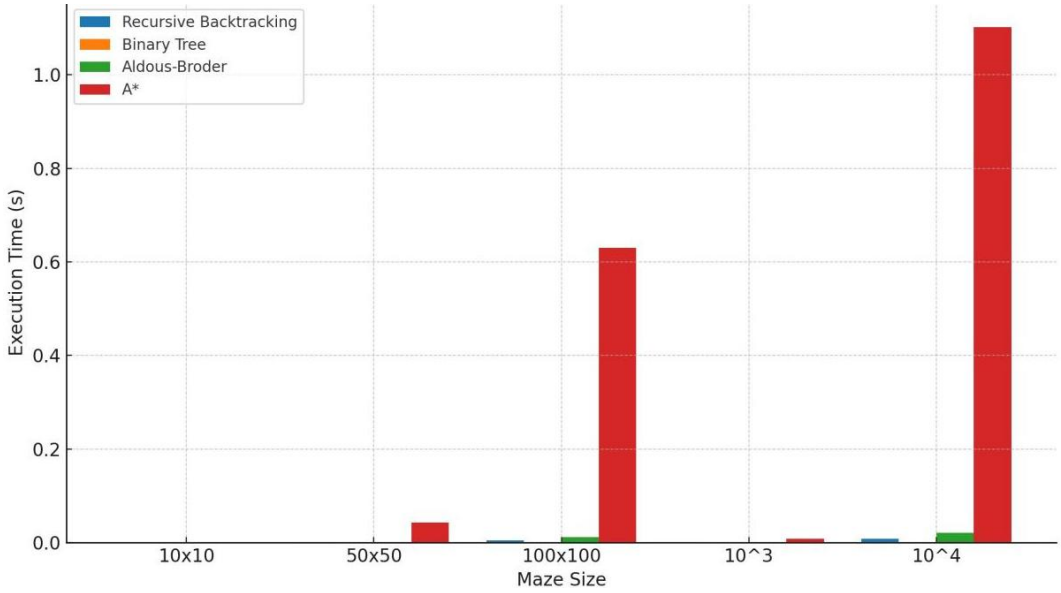


**Figure 3.** The average execution time by maze size (Picture credit: Original)

For small mazes (10×10), all algorithms perform similarly, with execution times under 0.001s. However, as maze size increases, differences emerge. The Aldous-Broder algorithm exhibits linear complexity but with higher constant factors due to repeated random walks. In contrast, A* shows near-linear growth initially but eventually scales as O (N log N), leading to steep increases for very large mazes.

To further test consistency, a 50×50 maze was generated 10 times with each algorithm.

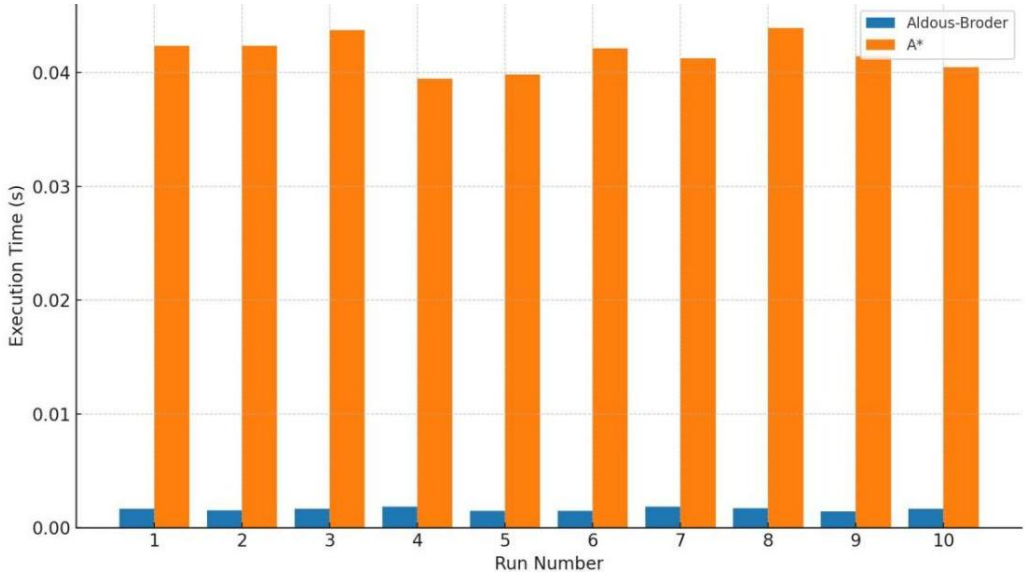


**Figure 4.** The execution time per run (50x50 maze, 10 runs) (Picture credit: Original)

Fig. 4 reveals that Aldous-Broder's execution times fluctuate widely due to randomness in path traversal. A*, while slower on average for small mazes, shows more consistent performance due to heuristic guidance.

Trade-offs are evident in the comparative analysis. Aldous-Broder is inefficient but generates unbiased spanning trees. Calculation is wasted because the random walk returns to nodes a lot [4]. This explains why execution times are more variable and consistently higher.

A*, on the other hand, uses heuristic-guided traversal to show efficiency. By providing directional bias, the Manhattan distance heuristic minimizes needless exploration [12]. This also implies that the algorithm does not produce a consistent spanning tree, which introduces bias into the structure of the maze. For instance, paths typically follow heuristic instructions, which lessens the "randomness" of mazes.

Trends in scaling are consistent with theory. Aldous-Broder shows a roughly linear slope on a log–log plot, with a high variance resulting from repeated revisits; the coefficient of variation is highest on 100×100 grids, which is consistent with the random-walk cover-time intuition [4]. Strong heuristic guidance causes A* to grow sublinearly at small N. As priority-queue costs take over, A* bends toward O (N log N); for very large grids, memory traffic from OPEN-set maintenance becomes the primary bottleneck [12, 13]. These narratives are supported by run-to-run dispersion on the 50×50 regime: While A* clusters closely around the median, Aldous-Broder displays wide IQRs.

Aldous-Broder produces more isotropic textures, which are preferred when uniformity is needed, whereas A* tends to form straighter avenues aligned with heuristic axes, trading randomness for directed progress [9, 11]. These observations drive hybrid pipelines, such as UST methods for a fraction of edges to reduce bias without fully paying the cover-time cost, or a short A* skeleton pass for connectivity followed by the injection of controlled randomization to decorrelate corridor orientations.

According to the findings, Aldous-Broder is better suited for applications requiring uniform randomness, like simulation or cryptography [11]. Applications like real-time game generation, where efficiency is a top priority, are better suited for A*. A*'s O (N log N) priority queue operations are the main cause of its poor scalability when working with very large mazes [13]. Heuristic selection also affects results; using the wrong heuristic reduces efficiency. According to Aldous-Broder, random walks are inherently inefficient.

Future research could integrate hybrid approaches: combining Aldous-Broder's uniformity with A*'s efficiency. Multi-algorithm frameworks may dynamically select methods based on maze size and application requirements. Moreover, introducing difficulty metrics for mazes [8] could help evaluate algorithm performance beyond execution time.

## 4. Conclusion

This study concludes by revisiting the central goal of evaluating maze generation algorithms from the perspective of efficiency and fairness. The experiments demonstrated that Aldous-Broder produces unbiased spanning trees but suffers from inefficiency, while A* accelerates generation through heuristic guidance yet introduces structural bias. The main contribution is a comparative framework that reveals the trade-offs between neutrality and performance. Limitations include the relatively small scale of experiments and the narrow set of algorithms. Future work should expand to larger datasets, integrate hybrid techniques, and develop adaptive methods for broader practical applications.

## References

[1] Buck J. Maze generation: Binary tree algorithm. Jamis Buck's Blog, 2011 Feb 1. Available from: https: //weblog.jamisbuck.org/2011/2/1/maze-generation-binary-tree-algorithm.

[2] Algorithms Journal. A comparative study of maze generation algorithms in a game-based mobile learning application for learning basic programming concepts. Algorithms. 2024; 17 (9): 404. Available from: https: //www.mdpi.com/1999 - 4893/17/9/404.

[3] International Journal of Intelligent Systems and Applications in Engineering (IJISAE). Comparative analysis of maze generation algorithms. IJISAE. 2023; 11 (2): 3557. Available from: https: //ijisae.org/index.php/IJISAE/article/view/3557.

[4] Wilson DB. Generating random spanning trees more quickly than the cover time. In: Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing (STOC '96). ACM; 1996. p. 296 – 303. Available from: https: //doi.org/10.1145/237814.237880.

[5] Algorithms Journal. Wilson's algorithm and loop-erased random walks in maze generation. Algorithms. 2024; 17 (9): 404. Available from: https://www.mdpi.com/1999 - 4893/17/9/404.

[6] EWA Direct. Using genetic algorithms for maze generation in Ms. Pacman. In: Proceedings of the Asian Conference on Education (ACE 2023). 2023. Available from: https://www.ewadirect.com/proceedings/ace/article/view/10312.

[7] Bridson R. Fast Poisson disk sampling in arbitrary dimensions. In: ACM SIGGRAPH 2007 Papers. ACM; 2007. p. 1 – 6. Available from: https://www.cs.ubc.ca/~rbridson/docs/bridson-siggraph07-poissondisk.pdf.

[8] Rabinovich A. On measuring the difficulty of mazes. ACM J Exp Algorithmics. 2012; 17: 1 – 20. Available from: https: //doi.org/10.1145/2133803.2184450.

[9] Aldous D. The random walk construction of uniform spanning trees and uniform labelled trees. SIAM J Discrete Math. 1990; 3 (4): 450 – 65. Available from: https://doi.org/10.1137/0403039.

[10] Broder A. Generating random spanning trees. In: 30th Annual Symposium on Foundations of Computer Science (FOCS). IEEE; 1989. p. 442 – 7. Available from: https: //doi.org/10.1109/SFCS.1989.63513.

[11] Propp JG, Wilson DB. How to get a perfectly random sample from a generic Markov chain. J Algorithms. 1998; 27 (2): 170 – 217. Available from: https://doi.org/10.1006/jagm.1997.0917.

[12] 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. Available from: https: //doi.org/10.1109/TSSC.1968.300136.

[13] Zhou R, Hansen EA. Breadth-first heuristic search. Artif Intell. 2006; 170 (4 – 5): 385 – 408. Available from: https: //doi.org/10.1016/j.artint.2005.10.003.