Exploring Pathfinding Efficiency in Grid-Based Mazes Using DFS, BFS, and A*

Abstract

In this project, I explored pathfinding in grid-based environments, a common problem in robotics, games, and navigation systems, where an agent must find a route from a start position to a target while avoiding obstacles. I implemented three core graph search algorithms: depth-first search (DFS), breadth-first search (BFS), and A*, using fundamental data structures such as stacks, queues, and priority queues, and applied a heuristic distance measure in A* to guide the search. By running large batches of simulations on random 20*20 mazes with varying obstacle densities, I measured reachability, path length, and the number of cells explored for each algorithm. The results show that all three algorithms are equally successful at finding a path whenever one exists, but BFS and A* consistently find near-shortest paths, while A* explores substantially fewer cells than BFS, and DFS tends to produce much longer, less efficient routes; additionally, the probability of reaching the target drops sharply once obstacle density exceeds roughly 50–60%, indicating a structural connectivity limit in the maze.

Results

- **Experiment**

I ran a series of simulations comparing depth-first search (DFS), breadth-first search (BFS), and A* search on randomly generated 20*20 mazes. For each trial, I fixed the start cell at (1, 1) and the target cell at (18,18) and generated obstacles independently with a given obstacle density d between 0.0 and 1.0 in steps of 0.1. At each density, I ran 100 trials. For every algorithm and trial, I recorded (1) whether the target was reachable (reached or not), (2) the length of the path found in number of cells (for successful trials), and (3) the total number of cells visited during the search. I then averaged these quantities over the 100 trials at each density to estimate the probability of reaching the target, the average path length, and the average number of explored cells.

- **Hypothesis**

Before running the experiments, I expected obstacle density to have a phase transition effect on reachability: with few obstacles, the start and target should almost always be connected, but above some critical density the maze should almost always be disconnected and the probability of reaching the target should drop to (essentially) zero. For the algorithms, I hypothesized that BFS and A* would find similarly short paths (because both are optimal in an unweighted grid), while DFS would often find significantly longer, more wandering paths. I also predicted that A* would explore fewer cells than BFS by using the heuristic distance to the target, and that both BFS and A* would typically visit more cells than DFS in very open mazes, where DFS can commit to one long corridor.

- **Table of Results**

Table 1. Effect of obstacle density on reachability, path length, and explored cells for DFS, BFS, and A* (subset of densities shown).

Each entry is averaged over 100 random 20*20 mazes at the given obstacle density. “Reached prob” is the fraction of trials where the target was reached; “Avg path len” is averaged over successful trials only; “Avg explored cells” is averaged over all trials.

Density	Algorithm	Reached prob	Avg path len (cells)	Avg explored cells
0.00	DFS	1.00	153.00	309.00
0.00	BFS	1.00	35.00	396.00
0.00	A*	1.00	35.00	297.00
0.20	DFS	0.97	76.03	203.84
0.20	BFS	0.97	35.21	308.49
0.20	A*	0.97	35.21	177.09
0.30	DFS	0.63	63.19	154.41
0.30	BFS	0.63	36.94	198.87
0.30	A*	0.63	36.94	133.58
0.50	DFS	0.01	37.00	17.93
0.50	BFS	0.01	37.00	18.49
0.50	A*	0.01	37.00	18.13
0.60	DFS	0.00	0.00	6.70
0.60	BFS	0.00	0.00	6.70
0.60	A*	0.00	0.00	6.70

As density increases from 0.0 to 0.5, the probability that the start and target are connected drops from 1.0 at low densities to 0.01 at density 0.5, and becomes exactly 0.0 by density 0.6. At low densities, all three algorithms almost always succeed, but DFS produces much longer paths (e.g., 153 cells vs. 35 cells at density 0.0) and explores a similar number of cells to BFS and A*. As density increases into the 0.2–0.3 range, DFS paths get shorter because many side corridors are blocked, but DFS still finds noticeably longer routes than BFS and A*, which consistently return near-shortest paths. A* systematically explores fewer cells than BFS at the same density (for example, 177 vs. 308 explored cells at density 0.2), showing the benefit of using a heuristic to guide the search toward the target. The sharp drop in reachability between densities 0.4 and 0.6 indicates that beyond roughly 50–60% obstacles, the maze effectively breaks apart into disconnected regions, meaning that in a heavily cluttered environment, no algorithm can reliably reach the goal because most starts and targets are not connected at all. The original data is stored in search_experiment_results.txt under the src directory.

- **Synthesis**

Across all densities from 0.0 to 0.5, the three algorithms have essentially identical reach probabilities at a given obstacle density, which matches the theoretical expectation that they should find a path whenever one exists. The main differences lie in path quality and search cost. BFS and A* consistently find paths of length about 35–40 cells whenever the maze is connected, which is close to the shortest possible path in a 20*20 grid. DFS, in contrast, often returns paths that are two to four times longer in open mazes because it pursues one direction deeply and backtracks only when forced. As obstacles are added, DFS's average path length decreases somewhat (many potential detours are blocked), but it still remains longer than BFS and A* until reachability collapses around density 0.5–0.6.

The exploration statistics highlight the practical trade-offs between the algorithms. In an open grid (density 0.0), BFS explores the most cells (about 396 on average), because its wavefront expands uniformly in all directions. DFS and A* explore fewer cells (309 and 297, respectively) because they focus more on one direction. As density increases, all three algorithms explore fewer cells overall, partly because many paths terminate quickly at obstacles and partly because many mazes are disconnected. A* consistently explores fewer cells than BFS at the same density (e.g., 211 vs. 356 explored cells at density 0.1 and 133 vs. 199 at density 0.3) while still matching BFS's path length, confirming that A* successfully cuts down the search space using its heuristic. These results show three distinct regimes. In sparse environments (low obstacle density), BFS and A* are preferable when we care about finding the shortest path, with A* offering shorter run times in terms of explored cells. DFS is less attractive because it generates much longer paths with no gain in reachability. As the environment becomes more cluttered, all methods start to fail for the same structural reason: the grid becomes disconnected, and the target is simply unreachable. This suggests that in real-world applications like robotics or game pathfinding, the choice of algorithm matters a lot in moderately cluttered but still navigable environments (where A* can give big efficiency gains), while beyond a certain obstacle density, the bottleneck is not the search strategy but the geometry of the space itself.

- **Acknowledgements**

I would like to thank Professor Harper and Professor Al Madi for providing the project template and guidance site, as well as the TAs for guiding me through the debugging process. I also appreciate that my classmates help me debug. Thanks to Oracle and W3Schools for the detailed Java documentation and Wikipedia pages (Depth-first search, Breadth-first search, and A* search algorithm), which helped me grasp the search algorithms quickly.

Extension:

- **What**

For my extension, I focused on making the search algorithms more realistic for a physical robot, similar to the Micro Mouse competition. In the basic project, DFS, BFS, and A* teleport between frontier cells for free, but a real robot has to physically move through the maze to visit each explored cell. To capture this, I measured not just the logical path from start to target, but also the total number of grid steps required to walk between consecutive visited cells in the order each algorithm explores them. This lets us ask a different question: which algorithm is fastest in terms of physical motion, under the assumption that moving through the grid is costly and turning/accelerating are nontrivial?

Thus, I made some implementation changes: In AbstractMazeSearch.java, I added a visitedOrder list to record the order in which cells are removed from the frontier, exposed this list via a getVisitedOrder() accessor, cleared the list at the beginning of search, and appended each cell immediately after it is removed from the frontier in findNextCell(). In SearchWalkingExperiment.java, I ran this experiment on the same setup as before: random 20*20 mazes with obstacle densities from 0.0 to 1.0 in steps of 0.1, with 100 trials per density. For each density and algorithm, I recorded (1) the probability of reaching the target, (2) the average path length, (3) the average number of explored cells, and now (4) the average number of walking steps based on the exploration order.

- **Outcome**

Table 2. Walking cost vs. path quality for DFS, BFS, and A* in an open 20*20 maze (100 trials per density).
“Reached prob” is the fraction of trials where the target was reached; “Avg path length” is averaged over successful trials only; “Avg explored cells” is averaged over all trials.

Density	Algorithm	Reached prob	Avg. path length (cells)	Avg. explored cells	Avg. walking steps
0.00	DFS	1.00	153.00	309.00	151.00
0.00	BFS	1.00	35.00	396.00	1393.00
0.00	A*	1.00	35.00	297.00	1115.00
0.10	DFS	1.00	101.38	248.18	146.80
0.10	BFS	1.00	35.02	355.06	1517.32
0.10	A*	1.00	35.02	215.26	763.26
0.20	DFS	0.95	77.82	196.66	159.28
0.20	BFS	0.95	35.27	303.25	1549.92
0.20	A*	0.95	35.27	172.07	739.18
0.30	DFS	0.68	60.41	156.34	185.67
0.30	BFS	0.68	36.12	218.74	1381.04
0.30	A*	0.68	36.12	141.10	941.14
0.40	DFS	0.18	47.33	79.34	121.32
0.40	BFS	0.18	39.44	86.87	549.40

Density	Algorithm	Reached prob	Avg. path length (cells)	Avg. explored cells	Avg. walking steps
0.40	A*	0.18	39.44	76.98	578.20

Across obstacle densities from 0.0 to 0.4, all three algorithms have essentially the same reach probability at each density, and BFS/A* consistently find short paths of about 35–40 cells when a path exists, while DFS paths shrink from 153 cells down to about 47 cells as obstacles block detours. However, the average walking steps tell a very different story: DFS stays in the ~120–190 range, while BFS is consistently above ~1350 steps for densities 0.0–0.3 and A* stays around ~740–1100 steps. This shows that BFS and A* pay a huge cost in sweeping large regions of the maze while DFS tends to move locally along one corridor, making it far cheaper for a real robot to walk the algorithm, even though its final logical path is worse. At density 0.4, reachability already drops to 0.18 for all three algorithms, foreshadowing the sharp collapse to zero reach probability around densities 0.5–0.6 where the grid becomes effectively disconnected. Overall, the extension shows that the "best" algorithm depends heavily on what is being optimized: if the goal is a short final path, BFS and A* are preferable, but if the goal is to minimize physical motion during search, DFS can actually be the most attractive choice. The original data is stored in search_walking_results.txt under the ext directory.

- **How to run**

Compile and Run SearchWalkingExperiment.java. This class will write the results to search_walking_results.txt.