Comparison between A*, BFS and DFS algorithms in 3D grids

Arya Poonamalli

Abstract

Pathfinding algorithms are computer algorithms that aim to find a path through a grid with obstacles. A pathfinding algorithm works by taking a search grid with obstacle cells, a source cell and a destination cell, and searching outward from the source cell until it finds the destination cell. This research paper compares three pathfinding algorithms when they are made to search a 3D grid instead of a 2D grid. The algorithms tested are the A* search, breadth-first search and depth-first search. Each of the three algorithms is used to search a series of 3D grids with increasing obstacle density to determine which performs the best in terms of time taken to find the destination, length of the resulting path, and the number of cells searched to find the destination. These three metrics are graphed for each algorithm against the obstacle density of the grids to compare them. The results of the experiment prove decisively that in a 3D graph, the A* search algorithm performs the best across all three metrics due to its nature as an informed search algorithm.

Introduction

Pathfinding algorithms are computer algorithms that aim to find a path through a graph with obstacles. The algorithm explores the graph by starting at a source cell and checking different cells of the graph until it finds its destination. The cell that the algorithm checks next depends on the type of algorithm.

Pathfinding algorithms have a variety of applications in fields such as GPS navigation, game development, robotics [1], and space exploration [2]. In navigation, pathfinding algorithms are used to find the fastest route from one place to another. In game development, pathfinding algorithms are used for non-player characters (NPCs) to find a path through their environment [3]. In space exploration, pathfinding algorithms are used to help rovers navigate the terrain on the moon and other planets [4].

In navigation, a map-matching process is used, which integrates digital map data with data from a positioning system such as GPS [5]. This is required because a pathfinding algorithm needs to know where the user is before it can start searching outward from that position. Then an algorithm, such as the A* search algorithm or Dijkstra's search algorithm, finds the most efficient path through the map of streets [6]. These algorithms take into account factors like the amount of traffic on each street, which is found using data collected from cellular devices located on those streets [6].

In game development, pathfinding algorithms are used to find the shortest path for non-player characters to take from one point to another. This requires a graph of the NPC's surroundings, taken from already-existing data from within the game's code. This can take the form of a 2D square grid, hexagonal grid or triangular grid, or a 3D cubic grid [1]. It can also take the form of an irregular grid, such as a visibility graph [1]. A pathfinding algorithm is used, usually the A* algorithm [7], due to its reliability and guarantee that it always finds the shortest path.

In robotics, for a pathfinding algorithm to be applicable, a graph first has to be generated from the surroundings. This graph is created using data from the robot's sensors, and can take the form of a 2D square grid, hexagonal grid or triangular grid, a 3D cubic grid, or a visibility graph, similar to in game development [9]. Another pathfinding algorithm is often used here, called the Jump Point Search (JPS). The Jump Point Search optimises the A* algorithm by being able to make jumps across straight lines instead of checking every grid cell along that line [8].

In space exploration, pathfinding algorithms are used to guide robots on the Moon or other planets [6] [10]. This works on the same principle as when pathfinding algorithms are used by any other robot - a graph is created of the surroundings using sensor data, and a pathfinding algorithm is used to find the best path through the graph.

A* algorithm: The A* algorithm aims to find the shortest path through a grid by calculating for each cell the cost of reaching that cell from the source (g value) and the distance from that cell to the destination (h value). The algorithm then adds those 2 values to create for the cell an f value, and chooses to visit the cell with the lowest f value. Running the calculations for each cell the algorithm visits may take some time, but in the end it is guaranteed to find the shortest possible path from the source to the destination. The A* algorithm is used in a wide range of scenarios, because it's reliable and always guaranteed to find the shortest possible path. For example, it is used in GPS navigation [6] and game development [7]. The A* search algorithm is an example of an informed search algorithm. This means it knows which direction the destination cell is, so it can optimise itself accordingly and only search grid cells that take it closer to the destination [11].

Breadth-First Search (BFS): The BFS works by first checking the closest cells to the source, then checking the closest cells to those cells, and so on. This does not require any calculations like for the A* algorithm, but the algorithm needs to visit every cell adjacent to the cells it's visited before it can move on, and the path found may not be the fastest. This makes the BFS less efficient than the A* algorithm. It is used when the source is close enough to the destination that an in-depth exploration is not required [12]. The breadth-first search is an example of an uninformed search algorithm, also called a blind search algorithm. This means it does not know which direction the destination cell is, so it continues exploring outward in every direction until it finds its destination [11].

Depth-First Search (DFS): The DFS works by going down one path until its end before switching to a different path. This algorithm also requires little to no calculations. This algorithm sometimes gets lucky and immediately goes down the correct path, in which case it finds the destination extremely fast. However, it usually goes the wrong way and gets an unnecessarily long path. This algorithm is used when an in-depth exploration of the graph is necessary to find the destination [13]. The depth-first search is another example of an uninformed search algorithm [11].

Methodology

Independent variables:

- Algorithm: The 3 algorithms being used are the A* search algorithm, breadth-first search, and depth-first search. Each of them performs differently in grids of different obstacle densities.

- Density: The 3 algorithms will be tested using 10x10x10 grids of different densities, ranging from 0.0 (no blocked cells at all) to 0.8 (4 out of every 5 cells are blocked). Different algorithms will perform differently in higher or lower densities.

Dependent variables:

- Length of path found: The length of the path found by the algorithm depends on which algorithm is being used and the number of obstacles in the grid. It will be measured in cell units. It is calculated by adding the distances between each pair of consecutive cells in the path. The distance between each pair of consecutive cells is calculated by using the formula $\sqrt{((\Delta x)^2 + (\Delta y)^2 + (\Delta z)^2)}$, where Δx is the difference between the x-coordinates of the two cells, Δy is the difference between the y-coordinates of the two cells, and Δz is the difference between the z-coordinates of the two cells. Because the different length values are sometimes orders of magnitude apart, the dependent variable on the graph will be log(l) where l is the path length.

- Time taken: The time taken by the algorithm to find a path through the grid is measured in seconds.The time starts when the algorithm is given the grid, source cell coordinates and destinations cell coordinates, and ends when the algorithm outputs the path, its length, and the other cells it visited. Because the different time values are sometimes orders of magnitude apart, the dependent variable on the graph will be log(t) where t is the time taken.

- Cells visited: While finding the path, the algorithm checks other cells in the process. The number of cells an algorithm visits, including the cells that are part of the final path found, is counted.

Control variables:

- Grid size: All the algorithms are tested in grids that are 10 cells long, 10 cells high and 10 cells deep. This means each grid has 1000 cells.

- Position of destination relative to source: The destination is always on the opposite corner of the grid from the source. For example, if the source cell has the coordinates 0,0,0 then the destination cell will have the coordinates 9,9,9.

The implementation of the 3 pathfinding algorithms was done in Python. The visualisation of the 3D graph search was done using Plotly, a library of Python. Different colours were assigned to cells that were blocked, visited by the algorithm, and part of the path found. Plotly was also used to graph the time taken, length of the path found and number of cells searched against obstacle density for each search algorithm.

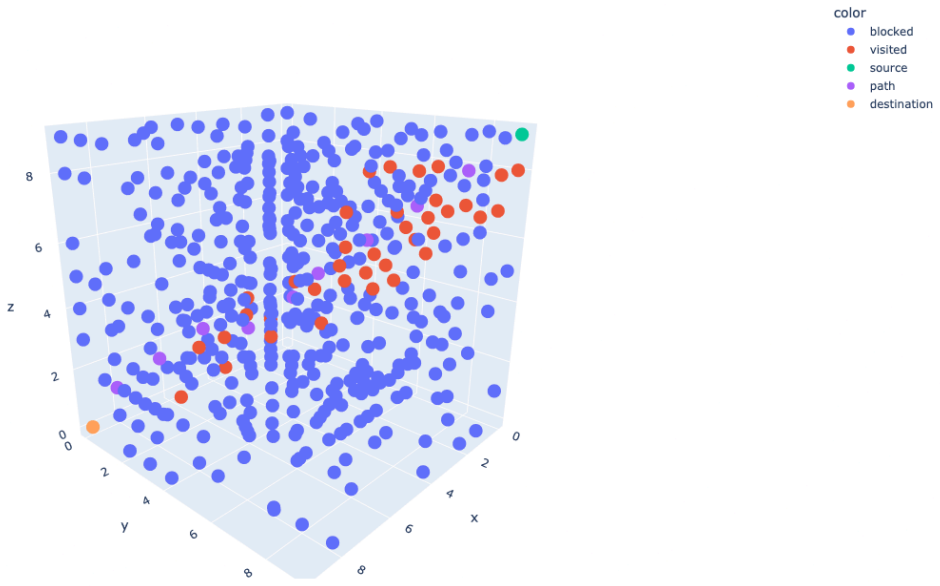

Fig 1: This is the visualisation of when the A* algorithm was used on a search grid with obstacle density 0.4.

Hypothesis: The hypothesis of the research is that the A* algorithm will take an average amount of time to find the fastest path, while the breadth-first and depth-first searches will take longer and visit more cells on the way.

Research question: To what extent does the obstacle density of a 3d grid traversed by a pathfinding algorithm affect the average length of the path found, time taken to find the path, and the number of cells visited by the algorithm?

This research question is aimed at finding out how different pathfinding algorithms perform in 3 dimensions instead of 2. This can be measured by finding out the time taken by the algorithm, number of cells visited by the algorithm, and the length of the path found. A better-performing algorithm would take less time, visit fewer cells, and find a shorter path than a more poorly performing algorithm.

Methodology: In total, 100 trials were conducted for each level of obstacle density. For each trial, a random 3D 10x10x10 grid was generated with a source cell at one corner, a destination cell at the opposite corner, and obstacle cells in between. This grid was given to each algorithm, and three variables were measured each time. One was the time taken by the algorithm to find a path from the source to the destination. The second was the length of the path from the source to the destination found by the algorithm. The third was the number of cells searched by the algorithm on the way from the source to the destination. After the trials were complete, the average value for each of these variables was calculated for each algorithm at each level of obstacle density. Three graphs were then made - one of average time taken against density for each algorithm, one of average path length against density for each algorithm, and one of the average number of cells searched against density for each algorithm.

Results

These results show that the A* search algorithm consistently outperforms both the breadth-first search and depth-first search. However, the breadth-first search and depth-first search do come close to it sometimes.

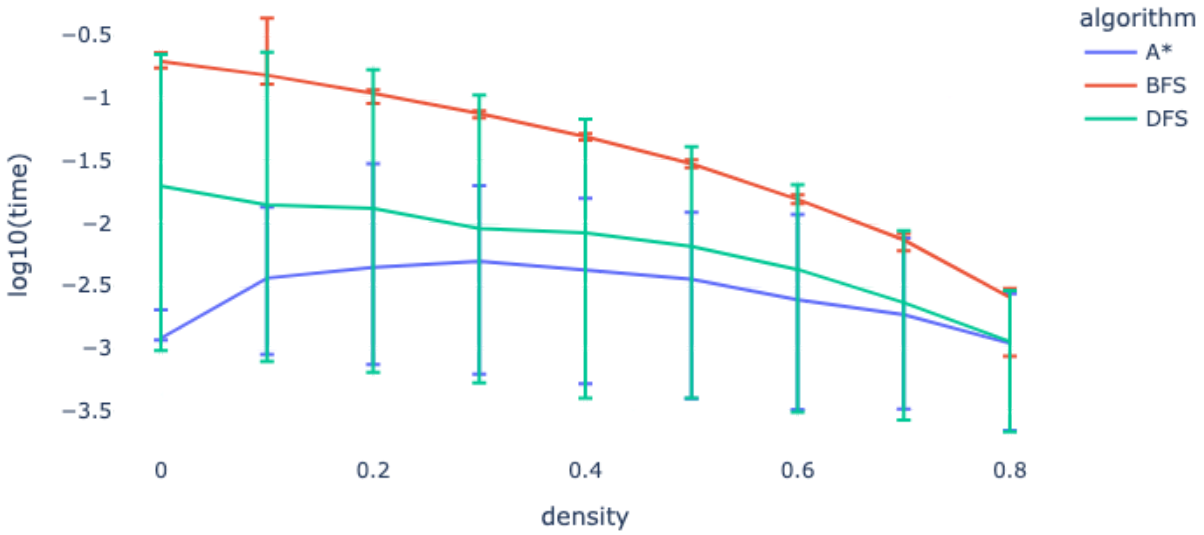

Fig 2: This is a graph of log10 of time taken against the graph density. The error values represent the highest calculated value and the lowest calculated value. As we see, the A* algorithm does best on average across all densities, though the depth-first search comes close at density 0.8. The depth-first search also has a wider range of values, meaning it sometimes does extremely well and sometimes does extremely badly.

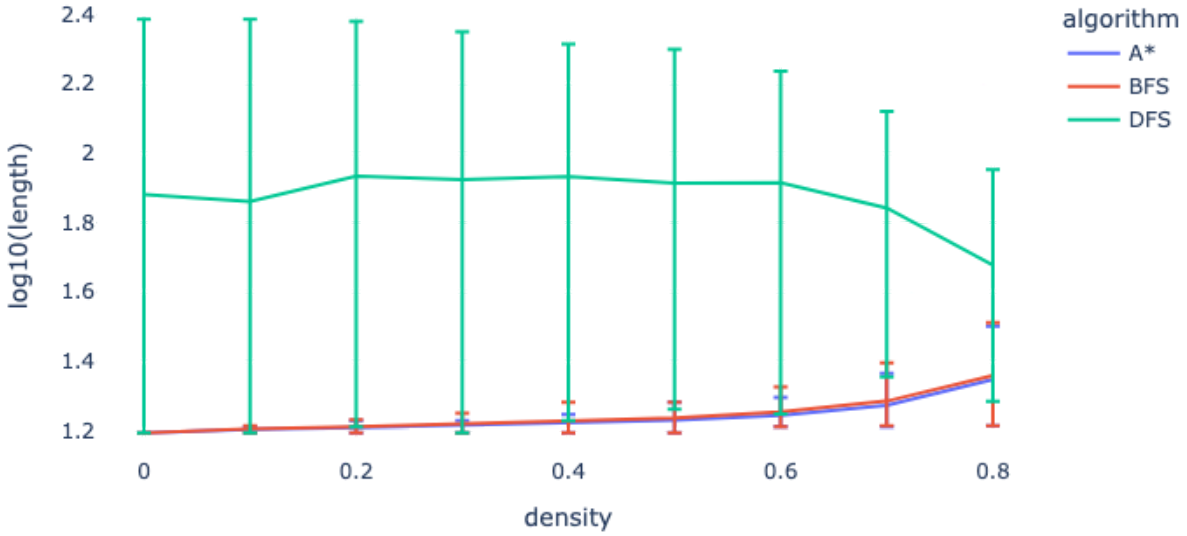

Fig 3: This is a graph of log10 of the length of the path found against the graph density. The error values represent the highest calculated value and the lowest calculated value. The A* algorithm is still the best across almost all densities, but the breadth-first search comes equal to it at lower densities. Meanwhile the depth-first search has an extremely high average, but

sometimes does better than the breadth-first search, but never better than the A*.

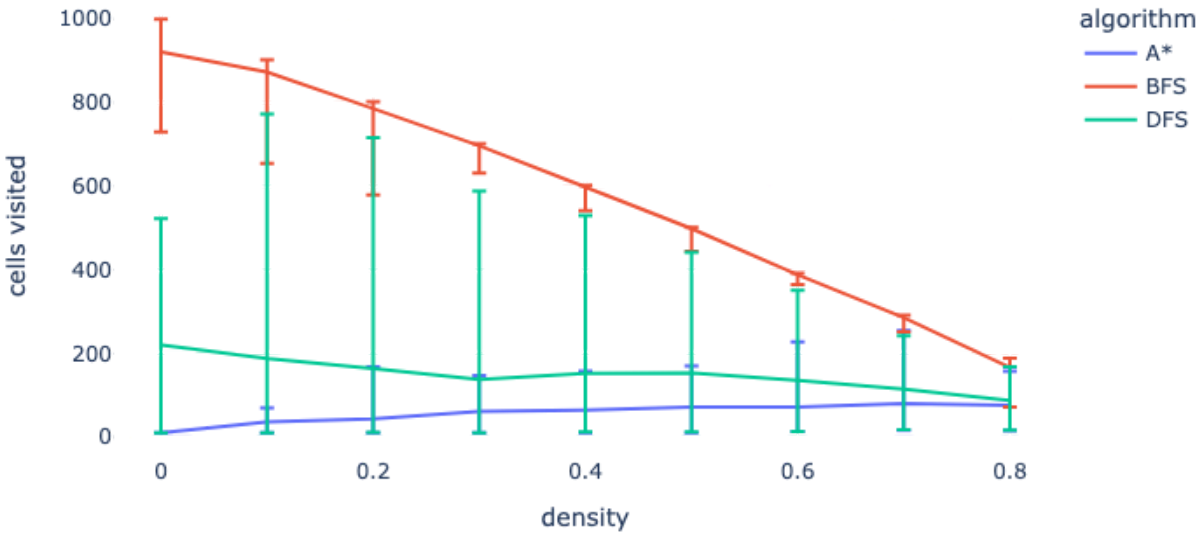

Fig 4: This is a graph of the number of cells visited by the algorithm against the graph density. The error values represent the highest calculated value and the lowest calculated value. Here, A* is still the best on average across all densities, with the depth-first search coming close at density 0.8. Even with a high upper bound to its range, the depth-first search still never does worse than the breadth-first search's average.

Ranking by time taken:

1. In first place is the A* algorithm. On average, it is faster than both the depth-first and breadth-first searches, except at density 0.8, when the depth-first search is on average slightly faster.
2. In second place is the depth-first search. It is always faster on average than the breadth-first search. However, it is slower on average than the A* search, except at density 0.8, when it becomes slightly faster. Also, even at lower densities, it may get lucky and get to its destination faster than the A* algorithm.
3. In third place is the breadth-first search. It is slower on average across all densities than the depth-first search and A* search. This is because it usually visits almost every cell possible before finding its destination.

Ranking by length of path found:

1. In first place is the A* algorithm. This is because it always runs the necessary calculations to find the shortest possible path through the grid.
2. In second place is the breadth-first search. At lower densities, it may find the same path as the A* algorithm, resulting in the lengths of the paths found to be the same. However,

at higher densities, the shortest possible path is not as obvious and can only be found by the A* algorithm, whereas the breadth-first search finds a slightly longer path.
3. In third place is the depth-first search. It often takes an extremely long, complicated path to reach the destination. However, sometimes it gets lucky and finds an extremely short path, which may be shorter than that found by the breadth-first search.

Ranking by number of cells visited:
1. In first place is the A* algorithm. It visits the fewest number of cells because each new cell it visits is constantly guiding it closer towards its destination. This helps it reach its destination by visiting very few cells.
2. In second place is the depth-first search. On average it visits more cells than the A* algorithm, but sometimes it gets lucky and visits fewer.
3. In third place is the breadth-first search. It usually visits almost every cell it can before it finds its destination.

Overall ranking:
1. In first place is the A* algorithm. On average it has the fastest time, finds the shortest possible path every time, and visits the least number of cells. It can be used in a wide variety of applications due to its reliability and high performance.
2. In second place is the depth-first search. Sometimes it takes extremely long, finds an extremely long path, and visits a large number of cells, and sometimes it gets lucky, goes straight for the destination, and reaches it in no time flat. However, it usually performs averagely in terms of time taken and cells visited, even if it finds very long paths. It can be used in high-density grids when time is of the essence and path length doesn't really matter.
3. In third place is the breadth-first search. It is consistently the slowest algorithm and it visits almost every cell it can. However, the paths it finds are only slightly longer than those found by the A* algorithm. It can be used when there is a need for a simple algorithm and the amount of time it takes doesn't matter.

Conclusion

This leads to the conclusion that in a 3d search grid, the A* algorithm on average outperforms the breadth-first search and depth-first search in every metric across all densities. This means that in a 3d grid, the A* algorithm would be suitable for almost any purpose. However, in graphs of very high densities, the depth-first search can match the A* algorithm in cells visited and time taken. This makes the depth-first search more suitable for high-density graphs if the length of the path found is not of as much importance.
This does match the hypothesis that the 3d graphs wouldn't affect the A* algorithm's ability to always find the shortest path. However, it was not expected that it would outperform both the

breadth-first search and depth-first search by such a wide margin. This is probably due to the fact that the A* search is an informed search algorithm and the breadth-first search and depth-first search are not, which makes it more adaptable to different conditions such as 3d grids.

Future Work

Future work includes experimenting with even more search algorithms, such as Dijkstra's search algorithm. It would be interesting to see how it compares to the breadth-first search and depth-first search, but it is doubtful that it could match the A* algorithm. This would be done by writing a separate search function for Dijkstra's algorithm and making the main program run it alongside the other algorithms for each trial.
Also, one thing that was kept constant is the position of the destination cell relative to the source cell. It would be interesting to experiment with different destination cells in the grid to see if that affects the performance of any of the algorithms. One algorithm in particular, the breadth-first search, is said to perform best when the destination is close to the source and does not require an in-depth traversal to find, so it would be interesting to see how much better it could perform under such circumstances. This would be done by changing the function that creates a new graph, source and destination for each trial to have it put sources and destinations at random points instead of simply at opposite corners. Also, the distance between the source and destination would have to be brought in as a variable, and the time taken, length of path found, and number of cells visited could be graphed against it.

References

[1] Abd Algfoor, Zeyad, Mohd Shahrizal Sunar, and Hoshang Kolivand. "A comprehensive study on pathfinding techniques for robotics and video games." *International Journal of Computer Games Technology* 2015.1 (2015): 736138.
[2] Kjellberg, Henri C., and E. Glenn Lightsey. "Discretized constrained attitude pathfinding and control for satellites." *Journal of Guidance, Control, and Dynamics* 36.5 (2013): 1301-1309.
[3] Rafiq, A., Asmawaty Abdul Kadir, T., & Normaziah Ihsan, S. (2020, February). Pathfinding algorithms in game development. In *IOP Conference Series: Materials Science and Engineering* (Vol. 769, No. 1, p. 012021). IOP Publishing.
[4] Quddus, Mohammed, and Simon Washington. "Shortest path and vehicle trajectory aided map-matching for low frequency GPS data." *Transportation Research Part C: Emerging Technologies* 55 (2015): 328-339.
[5] Mehta, Heeket, Pratik Kanani, and Priya Lande. "Google maps." *International Journal of Computer Applications* 178.8 (2019): 41-46.
[6] Miao, Q., & Wei, G. (2025). A comprehensive review of path-planning algorithms for planetary rover exploration. *Remote Sensing*, *17*(11), 1924.

[7] Botea, Adi, et al. "Pathfinding in games." Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.
[8] Harabor, Daniel, and Alban Grastien. "Online graph pruning for pathfinding on grid maps." *Proceedings of the AAAI conference on artificial intelligence*. Vol. 25. No. 1. 2011.
[9] Lawande, Sharmad Rajnish, et al. "A systematic review and analysis of intelligence-based pathfinding algorithms in the field of video games." *Applied Sciences* 12.11 (2022): 5499.
[10] Tang, Laien. "Research on path planning of lunar exploration robot based on A* algorithm." *IET Conference Proceedings CP901*. Vol. 2024. No. 24. Stevenage, UK: The Institution of Engineering and Technology, 2024.
[11] Firmansyah, Boy. "A COMPARATIVE ANALYSIS OF INFORMED SEARCH AND UNINFORMED SEARCH ALGORITHMS IN THE EFFICIENCY OF AI PROBLEM MODELING." *Journal of Data Analytics, Information, and Computer Science* 3.3 (2026): 158-174.
[12] Zhou, Rong, and Eric A. Hansen. "Breadth-first heuristic search." *Artificial Intelligence* 170.4-5 (2006): 385-408.
[13] Tarjan, Robert. "Depth-first search and linear graph algorithms." *SIAM journal on computing* 1.2 (1972): 146-160.