STATISTICS, OPTIMIZATION AND INFORMATION COMPUTING
Stat., Optim. Inf. Comput., Vol. **12**, May 2024, pp 761–781.
Published online in International Academic Press (www.IAPress.org)

Exploring Maze Navigation: A Comparative Study of DFS, BFS, and A* Search Algorithms

ELKARI Badr [1],OURABAH Loubna [1],SEKKAT Hiba [1],HSAINE Ayoub [1] ESSAIOUAD Chama [1], BOUARGANE Yassine [1],EL MOUTAOUAKIL Karim [2,*]

[1]*EIDIA, Euromed Research Center, Euro-Med University (UEMF), Fez, Morocco*
[2]*Engineering Science Laboratory, Polydisciplinary Faculty of Taza, Sidi Mohamed Ben Abdellah University of Fez, Morocco*

Abstract A comparative study was conducted using Python and Pmaze to evaluate the performance of DFS, BFS, and A* search algorithms in path planning for a maze. The main objective of the project was to compare the efficiency of these algorithms in terms of path cost and algorithmic complexity. No physical robot was used in this study as it focused solely on the analysis of search algorithms. This study contributes to the advancement of knowledge in the field of path planning by exploring different approaches for mazes using Python and Pmaze.

Keywords Path Planning, Navigation Pymaze, DFS, BFS, A*, Search Algorithm Robotics

DOI: 10.19139/soic-2310-5070-1939

1. Introduction

Path planning plays a crucial role in a wide range of applications, including robotics, gaming, logistics, and autonomous systems. The ability to efficiently navigate from a starting point to a target location is essential for achieving optimal outcomes in these domains. As such, the development and evaluation of search algorithms for path planning have garnered significant attention.[1]

In this study, we conducted a comparative analysis to evaluate the performance of three popular search algorithms: Depth-First Search (DFS), Breadth-First Search (BFS), and A* (A-star). These algorithms have been widely studied and employed in various fields due to their effectiveness in finding optimal paths.

Our primary focus was on their application in maze path planning. Mazes, with their intricate layouts and complex configurations, serve as ideal testbeds for evaluating the efficiency and effectiveness of search algorithms. By comparing the performance of DFS, BFS, and A* in maze path planning, we aimed to gain insights into their strengths and limitations, providing valuable knowledge for practical implementation.[2]

To conduct the study, we utilized the Python programming language, renowned for its versatility and extensive libraries. Additionally, we leveraged the power of Pmaze, a specialized Python library designed for maze generation and path planning. The combination of Python and Pmaze provided a robust and flexible platform for implementing the search algorithms and evaluating their performance.

By investigating the performance of DFS, BFS, and A* in maze path planning using Python and Pmaze, our study contributes to the advancement of knowledge in the field. The findings and insights gained from this research hold significant implications for real-world applications, such as robotics navigation and game AI. Through our

*Correspondence to: ELKARI Bar (Email: b.elkari@ueuromed.org). EIDIA, Euromed Research Center, Euro-Med University (UEMF), Fez, Morocco.

ISSN 2310-5070 (online) ISSN 2311-004X (print)

comparative analysis, we shed light on the efficiency, effectiveness, and trade-offs associated with these search algorithms, paving the way for further enhancements and advancements in the field of path planning.[3, 4, 5, 6]

In the fascinating world of robotics and maze exploration, various techniques and algorithms come together to navigate through complex environments.

A maze was chosen as an example of a robot environment, where a camera is used to capture an image of the maze. The parameters of the maze are then extracted from the snapshot picture using image processing techniques. These parameters are utilized to draw the maze using MATLAB. Image processing techniques play a crucial role in the accuracy of finding the target within the maze.

To enhance the image, image enhancement techniques are employed to adjust the brightness and contrast values. Additionally, morphological algorithms are utilized to eliminate any unwanted components in the image.

Two searching methods, namely Depth-First Search (DFS) and Breadth-First Search (BFS), are employed to evaluate all possible paths to the target within the maze. There are differences in terms of performance, such as complexity, completeness, optimality, and time required to obtain the best solution.[7]

In the given scenario, a maze is chosen as an example of a robot environment. A camera is used to capture an image of the robot environment, and image processing techniques are employed to extract parameters from the snapshot picture. These parameters are then used to create a maze representation using MATLAB.

To improve the accuracy of finding the target within the maze, image enhancement techniques are applied. These techniques modify the brightness and contrast values of the image. Additionally, morphological algorithms are used to eliminate any unwanted elements in the image.

Two searching methods, namely Depth-First Search (DFS) and Breadth-First Search (BFS), are employed to evaluate all possible paths to the target within the maze. It is evident that both methods yield the same results in terms of finding the target, but they differ in terms of performance metrics such as complexity, completeness, optimality, and time required to obtain the best solution.[8]

This paper presents a comprehensive study on the Breadth-First Search (BFS) algorithm by conducting various simulation experiments. The efficiency of the BFS algorithm is evaluated using a Java-based simulation tool called RoboSim. The experimental results demonstrate that the BFS algorithm can effectively perform path planning in dynamic environments, successfully avoiding obstacles and reaching the target point. However, when different heuristic functions are applied, variations in the results are observed. As a result, further investigation of this algorithm is warranted, and it will be deployed in real-time Autonomous Mobile Robotic Applications (AMRA) for mission-critical applications.[10]

This article discusses two searching methods, namely Depth-First Search (DFS) and Breadth-First Search (BFS), which have been employed to evaluate all potential paths to a target. While both methods yield the same results, there are disparities in terms of performance, including complexity, completeness, optimality, and time required for finding the best solution.

In terms of space efficiency, DFS outperforms BFS since it only needs to store information about the currently examined path. On the other hand, BFS is not time efficient when compared to DFS because it may end up exploring deep branches of the search tree, which can be computationally expensive. Additionally, some branches in the tree may be infinite, rendering DFS incomplete. However, BFS is considered an optimal search method, ensuring that it always finds the shortest path. In contrast, DFS does not guarantee optimality as it may find a solution that is not necessarily the best.

Therefore, considering the differences in complexity, completeness, optimality, and time efficiency, one can conclude that DFS and BFS offer distinct advantages and disadvantages depending on the specific requirements of the search problem at hand.[11]

The objective of this research was to create an information map for mobile robot path planning in the context of small transportation. The study focused on the quantitative and qualitative analysis and implementation of the A* algorithm for mobile robot path planning. The key findings indicate that accuracy and computation time for calculating the shortest path are crucial factors to consider in path planning.

The A* algorithm was chosen to generate the information map for mobile robot path planning. The algorithm was applied to the robot's working environment, which included various starting and ending points with associated weighted values. The main goal was to create a map that highlights the areas most frequently traversed by the robot

during pathfinding. The results showed that the robot tended to utilize areas without obstacles and sharp turns more frequently.

It should be noted that the heuristic component of the A* algorithm provides an estimated value from the current node to the final node, which may not always be entirely accurate. Alternatively, path planning algorithms like RRT or RRT* explore nearby nodes extensively and offer highly accurate results, although they may not be as fast as the A* algorithm.

Overall, the research highlights the significance of accuracy and computation time in mobile robot path planning. The A* algorithm proved effective in generating the information map, while considering factors such as obstacle-free areas and smooth turns. However, other algorithms like RRT or RRT* can provide more precise results by exploring a larger number of nearby nodes, albeit at a potential cost of increased computation time.[12]

Our findings align with existing research and related studies on maze path planning. The superiority of A* in terms of path cost and the trade-off between path optimality and computational complexity are widely acknowledged in the field.

Similar to our study, other comparative studies have also compared the performance of DFS, BFS, and A* in maze path planning. While the specific results may vary depending on the maze configurations and evaluation metrics, the overall trends and trade-offs observed in our study are consistent with previous research.

These similarities further validate the significance and reliability of our findings, supporting the broader understanding of the strengths and limitations of different search algorithms for maze path planning.

2. Methodology

2.1. Experimental Setup:

To conduct our comparative study of DFS, BFS, and A* search algorithms in maze path planning, we established a well-defined experimental setup. Firstly, we implemented a maze generation process to create diverse maze configurations for testing. The maze generation algorithm ensured the creation of mazes with varying complexities, including dead-ends, loops, and multiple paths. This approach allowed us to evaluate the algorithms' performance across a range of maze types.

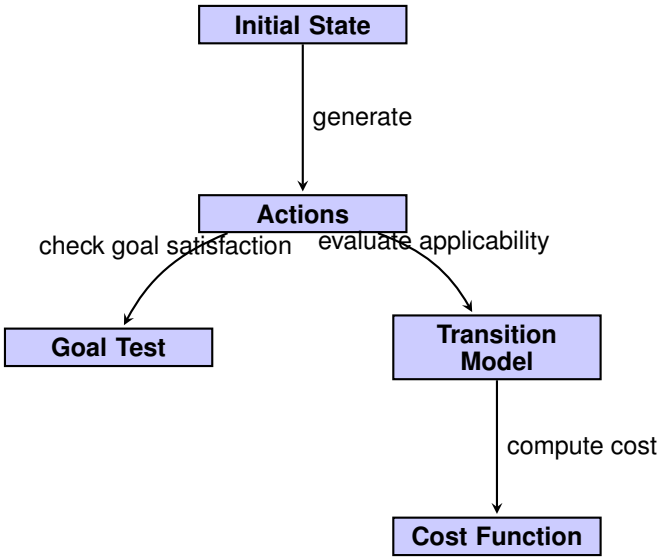
Next, we represented the generated mazes in the programming environment using Pmaze, a specialized Python library for maze manipulation and analysis. Pmaze provided a convenient and efficient way to interact with the maze structures, enabling the algorithms to navigate through the maze and find paths.

2.2. Problem Formulation:

All the DFS, BFS, and A* algorithms are based on a common problem, which can be formulated using the graph above. The nodes in the graph represent different elements of the problem, including the Initial State, Actions, Transition Model, Goal Test, and Path Cost Function. The edges between the nodes illustrate the relationships and dependencies between these elements.

By using these elements, DFS, BFS, and A* algorithms search for optimal or satisfactory paths to reach the destination by exploring the space of possible states. Each of these algorithms uses different strategies to select actions and perform the search.

It is worth noting that these algorithms are not limited to robot navigation but can also be applied to other pathfinding problems, such as trajectory planning in virtual environments, puzzle solving, and many more.[13]



2.3. General principle of the project:

The graph represents the general principle of the project. It starts with the ”Start” node and then follows the ”Search Algorithm” process. The process checks for obstacles in the path. If there is an obstacle, the project returns to an explored point to find an alternative path. If there is no obstacle, it checks if the destination has been reached. If the destination is reached, the project stops. If not, it chooses another path and continues the search algorithm. This loop continues until the destination is reached or there are no more available paths to explore.

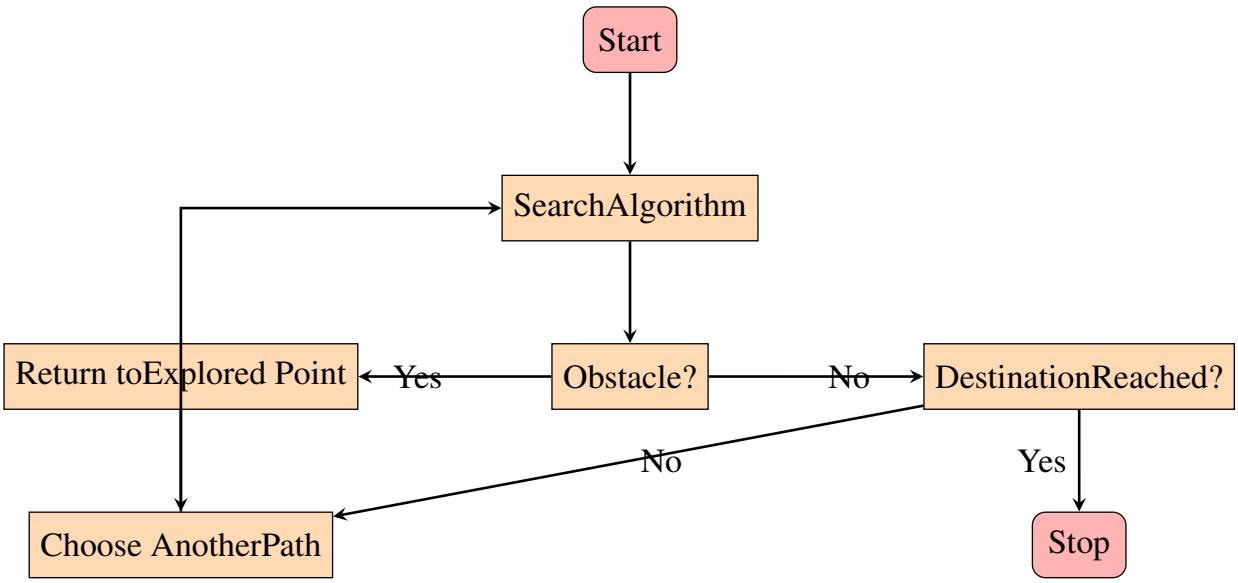
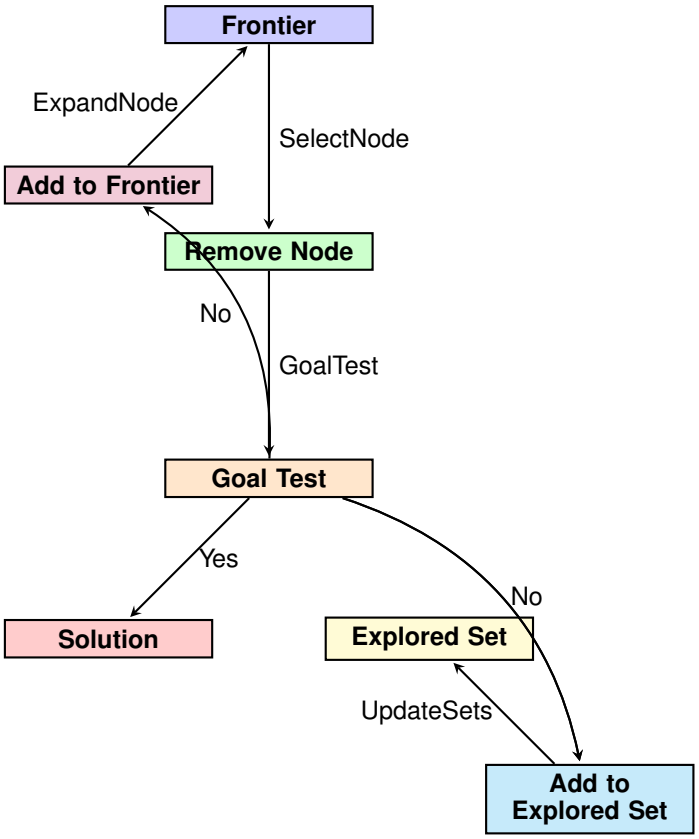


Figure 1. General principle of the project

2.4. General Approach:

The general approach for the DFS, BFS, and A* algorithms can be described by the graph above. The nodes in the graph represent different components of the approach, including the Frontier, Explored Set, Solution, and Node.

The edges between the nodes illustrate the actions taken during the algorithm execution,such as removing a node from the Frontier, performing a Goal Test, adding a node to the Explored Set, and adding a node to the Frontier. By following this approach, the DFS, BFS, and A* algorithms explore the space of possible states by adding new nodes to the Frontier, examining them, and continuing the search until a solution is found or all options have been explored.[14]



2.5. DFS algorithm:

Depth-first search (DFS) is a fundamental algorithm used to explore graph and tree structures. It starts from the root node and explores as far as possible along a specific branch before backtracking and exploring other unexplored branches. This process is repeated until all nodes in the graph or tree have been visited. Depth-first search is widely used in various fields of computer science, such as network analysis, route planning, path optimization, and solving tree traversal problems. Its ability to deeply explore complex data structures makes it a valuable tool for solving various algorithmic problems.[15]

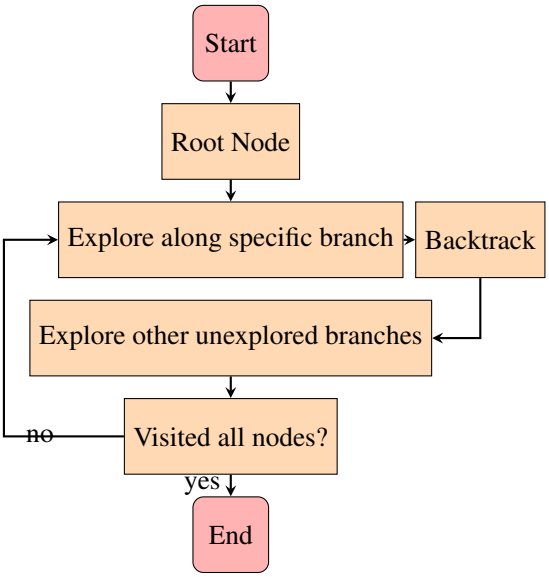


Figure 2. Graph Representation of the DFS Algorithm

2.6. BFS algorithm:

The breadth-first search algorithm (BFS) is an essential method for exploring graphs and solving various problems, such as finding the shortest path in a graph or solving puzzle games like the Rubik's Cube. Many computer science problems can be formulated in terms of graphs, such as network analysis, route planning, or mapping routes. Graph search algorithms, like breadth-first search, are valuable for analyzing and efficiently solving these problems.

Breadth-first search (BFS) starts by exploring a starting node, followed by its adjacent nodes, and then all nodes accessible through a path of two edges, three edges, and so on. The algorithm visits all vertices in a graph G that are at a distance of k edges from the source vertex s before visiting vertices at a distance of k+1 edges. This step is repeated until there are no more vertices accessible from start.[16]

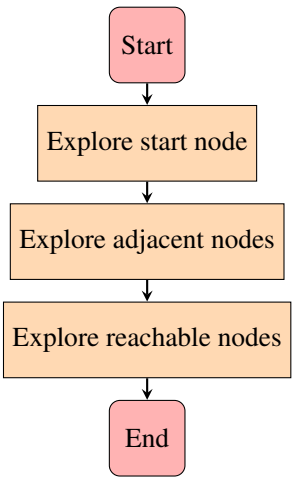


Figure 3. Graph Representation of the BFS Algorithm

2.7. A algorithm:*

A* (pronounced ”A star”) is a widely used computer algorithm for pathfinding and graph traversal. It efficiently finds a feasible path between multiple nodes or points on the graph. When it comes to finding a path on a map with numerous obstacles, as illustrated in the left pathfinding example below, it can be challenging to navigate from A to B. Without further guidance, a robot would simply keep moving forward until it encounters an obstacle.[17]

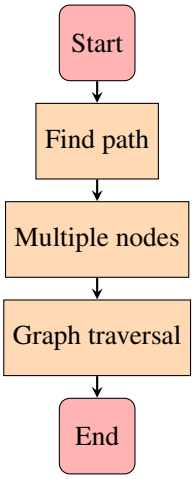


Figure 4. Graph Representation of the A* Algorithm

2.8. Evaluation Metrics:

In order to comprehensively assess and compare the performance of the search algorithms in maze path planning, we employed multiple evaluation metrics. These metrics were carefully selected to provide insights into various aspects of algorithm performance and efficiency.

Firstly, we considered the path cost as the primary metric. Path cost measures the total length or cost of the path found by each algorithm. By analyzing path cost, we gained a clear understanding of the efficiency and optimality of the algorithms in finding the shortest or most optimal paths within the maze. A lower path cost indicated a more efficient and favorable solution.

Additionally, we evaluated the time execution of each algorithm in different maze configurations. This analysis involved measuring the computational efficiency and scalability of the algorithms. By examining the time execution, we could identify any significant variations in performance among the algorithms and determine their suitability for real-time or time-sensitive applications.

Furthermore, we assessed the search path generated by each algorithm. The search path refers to the sequence of nodes explored by the algorithm during the maze traversal. Analyzing the search path allowed us to gain insights into the exploration patterns and decision-making processes of the algorithms. This information helped identify any potential strengths or weaknesses in terms of exploration efficiency and thoroughness.

Finally, we considered the algorithmic complexity, including factors such as time complexity and space complexity. Time complexity indicated the computational resources required by the algorithms as the input size or maze complexity increased. Space complexity referred to the memory requirements during algorithm execution. Analyzing these complexities provided valuable insights into the scalability and resource utilization of the algorithms.

By utilizing these evaluation metrics, including time execution, path cost, and search path analysis, we aimed to gain a comprehensive understanding of the strengths and weaknesses of DFS, BFS, and A* in maze path planning. This holistic evaluation approach, combining maze generation, algorithm implementation, and metrics analysis, allowed us to conduct a rigorous and systematic comparison of the search algorithms in our study.

2.9. Mathematical function

Now, we will explain and present each mathematical function for each algorithm.

2.9.1. Depth-First Search (DFS) DFS is an algorithm for traversing or searching tree or graph data structures. It explores as far as possible along each branch before backtracking. Mathematically, you can represent DFS using recursion or a stack.

$$\text{DFS}(v) = 1 + \sum_{u \in \text{neighbors}(v)} \text{DFS}(u) \tag{1}$$

Here:

- v is the current vertex,
- neighbors(v) gives the neighbors of v,
- DFS(v) represents the order in which vertices are visited.

2.9.2. Breadth-First Search (BFS) BFS explores a graph level by level. It starts at the tree root (or some arbitrary node of a graph), and explores the neighbor nodes at the present depth before moving on to nodes at the next depth level.

$$\text{BFS}(v) = 1 + \sum_{u \in \text{neighbors}(v)} \text{BFS}(u) \tag{2}$$

Here:

- v is the current vertex,
- neighbors(v) gives the neighbors of v,
- BFS(v) represents the order in which vertices are visited.

*2.9.3. A** The A* algorithm uses two main functions: the cost function g(n) and the heuristic function h(n). These functions play crucial roles in determining the cost associated with reaching a particular node and estimating the cost from that node to the goal, respectively

$$f(n) = g(n) + h(n) \tag{3}$$

Here:

- **Cost Function** $g(n)$**:** Represents the actual cost of the path from the start node to the current node. It considers the cumulative cost of traversing the edges from the start node to the current node.
- **Heuristic Function** $h(n)$**:** Provides an estimate of the cost from the current node to the goal node. It's a heuristic, meaning it doesn't necessarily give the exact cost but should be admissible (never overestimates the true cost).
- The A* algorithm combines these two functions using the formula:

$$f(n) = g(n) + h(n) \tag{4}$$

So, in summary, A* uses two functions, $g(n)$ and $h(n)$, and the sum of these functions, $f(n)$, guides the algorithm to explore nodes in a way that balances the cost of the current path and the estimated cost to reach the goal.
While these are the core functions, the specific implementation might involve additional details, such as tie-breaking strategies, ways to handle ties in $f(n)$ values, and the data structures used to efficiently implement the algorithm, like open and closed sets.

And now we will explain The heuristic function $h(n)$ in the A* algorithm, is a domain-specific function that provides an estimate of the cost from the current node to the goal node. It is important that the heuristic function

is admissible, meaning it never overestimates the true cost to reach the goal. If the heuristic is admissible, A* is guaranteed to find the optimal path.

The choice of the heuristic function depends on the problem domain and the specific characteristics of the problem. Here are some common types of heuristics used in different scenarios:

1. Euclidean Distance (for geometric problems): The Euclidean distance between two points (x_1, y_1) and (x_2, y_2) in a two-dimensional space is given by:

$$\text{Euclidean Distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{5}$$

2. Manhattan Distance (for grid-based problems): The Manhattan distance between two points (x_1, y_1) and (x_2, y_2) in a two-dimensional space is given by:

$$\text{Manhattan Distance} = |x_2 - x_1| + |y_2 - y_1| \tag{6}$$

3. Zero Heuristic (for problems without additional information):

$$h(n) = 0 \tag{7}$$

4. Custom Heuristic (domain-specific):

In our case, we opt for the Euclidean distance as it allows us to assess the current state of the robot in relation to its previous state and determine the movement required to reach the target.

2.10. Maze generation and Experimental Parameters

Mazes were dynamically generated using Python, leveraging the capabilities of the Pmaze library. This library facilitates the creation of mazes with customizable parameters, enabling a systematic exploration of various maze configurations. The size of the maze, representing its dimensions, was a crucial parameter manipulated during the generation process. Additionally, the maze included walls, representing obstacles and barriers that challenge the traversal of the algorithms.

To ensure the reproducibility of the experiments, several parameters were carefully considered and standardized across all trials.

- The size of the maze played a pivotal role in assessing algorithmic performance, with variations in maze dimensions providing insights into scalability and adaptability.
- The placement and distribution of walls within the maze were defined to simulate realistic scenarios, contributing to a comprehensive evaluation of algorithmic efficiency in the presence of obstacles.

2.10.1. Algorithm Representation Each search algorithm—DFS, BFS, and A*—was represented as distinct points within the maze, symbolizing their starting positions. The algorithms then navigated through the maze, adapting their paths based on the encountered obstacles and the defined goal.

2.10.2. Maze Diversity

2.10.3. Comparison Scenarios In all comparison scenarios, the DFS and A* algorithms were represented by a green dot, while BFS was denoted by a green square. This standardized representation provided a clear visual distinction in the comparison figures, aiding in the interpretation of results. .

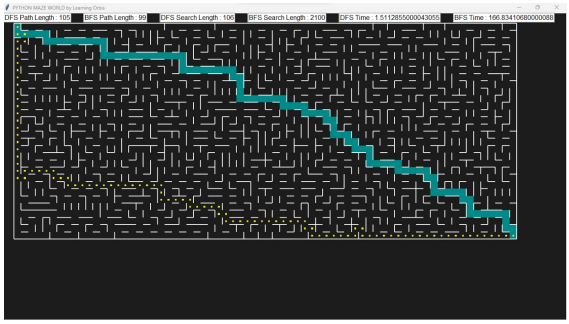


Figure 5. DFS VS BFS - Case 1

Figure 6. DFS VS BFS - Case 2

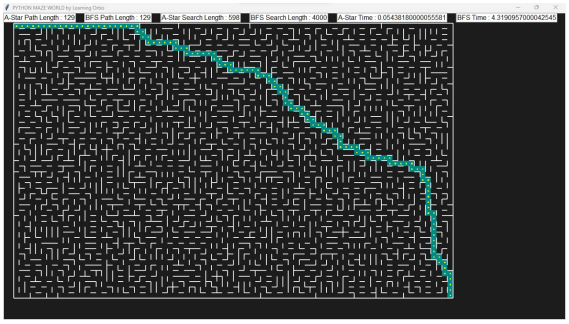


Figure 7. BFS VS A*

3. Comparative Study

In this section, we present the results of our comparative study, focusing on the performance of each search algorithm individually. We analyze their performance in terms of path cost, computational efficiency, and solution quality. To aid in the presentation of our analysis, we utilize tables and graphs to showcase the quantitative results obtained.

3.1. BFS vs. DFS (Start to Goal is Far):

We first compared the performance of BFS and DFS when the start and goal positions were relatively far apart in the maze. The objective was to evaluate how these algorithms fared in terms of path-finding efficiency over longer distances. Figures illustrates the comparison of path cost, and search path for BFS and DFS.

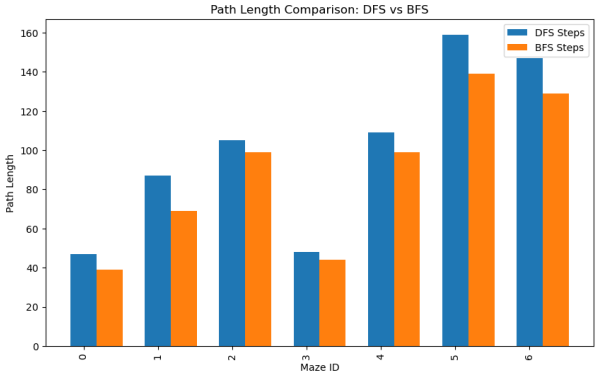


Figure 8. Path Length Comparison: DFS vs BFS

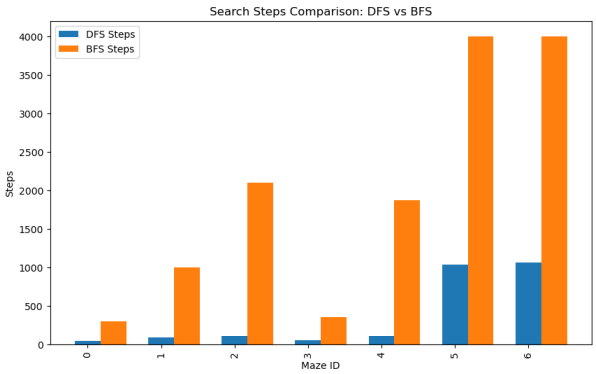


Figure 9. Search Path Comparison: DFS vs BFS

From the results, we observed that DFS generally exhibited lower Serach path costs compared to BFS in this scenario. This is because BFS systematically explores the neighboring nodes, ensuring that the shortest path is found. These findings highlight the trade-off between path optimality and computational efficiency when selecting between BFS and DFS for maze path planning.

In conclusion, based on our findings, it can be concluded that DFS outperforms BFS in terms of memory complexity. DFS requires less memory as it explores a single path deeply before backtracking, while BFS needs to store all visited nodes in memory. However, it is important to note that DFS does not guarantee the optimal path.

3.2. DFS vs. BFS (Start to Goal is Near):

In the next comparison, we focused on the performance of DFS and BFS when the start and goal positions were in close proximity within the maze. This evaluation aimed to assess how these algorithms performed when the search space was limited. Figures presents the results of path cost, and search path for DFS and BFS.

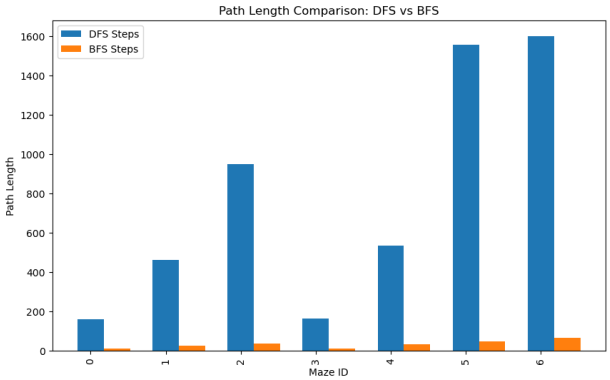


Figure 10. Path Length Comparison: DFS vs BFS

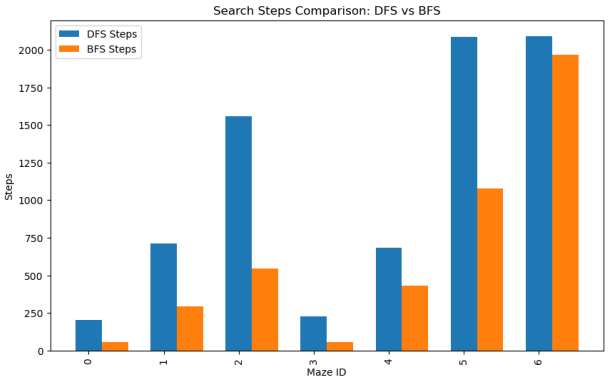


Figure 11. Search Path Comparison: DFS vs BFS

The results indicated that BFS generally achieved lower search path costs compared to DFS when the start and goal were near. This is because BFS explores the maze uniformly and is more likely to find a shorter path when the goal is in close proximity.These findings emphasize the importance of considering the proximity of the start and goal positions when selecting between DFS and BFS for maze path planning.

3.3. A vs. BFS:*

In the final comparison, we evaluated the performance of A* and BFS. A* is an informed search algorithm that utilizes heuristics to guide its search, while BFS is an uninformed algorithm that explores the maze uniformly. The comparison aimed to assess the effectiveness of A* in finding optimal paths compared to the more straightforward BFS. Figures showcases the comparison results of path cost, and search path for A* and BFS.

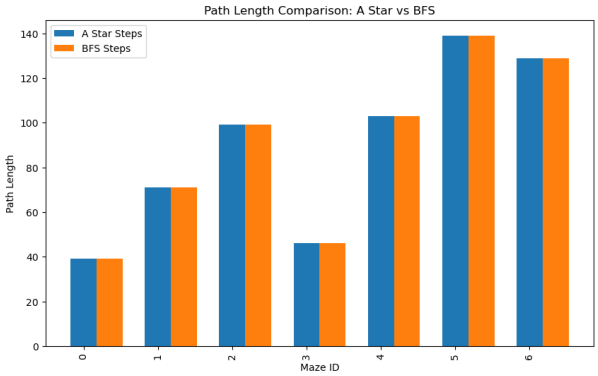


Figure 12. Path Length Comparison: A* vs BFS

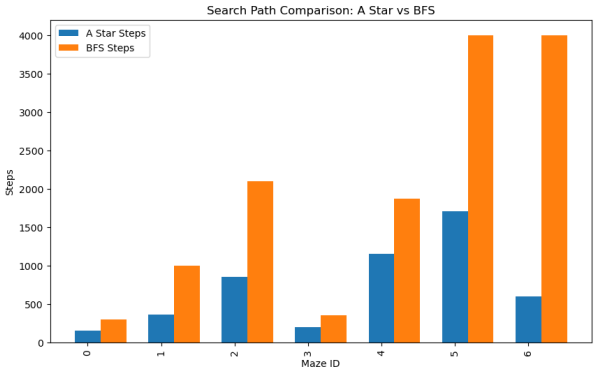


Figure 13. Search Path Comparison: A* vs BFS

The results indicate that A* and BFS have similar travel costs. However, A* integrates heuristics to guide research, allowing for more informed decisions. As a result, A* achieves a slightly better optimization than BFS. On the other hand. These results highlight the trade-off between trajectory optimization and the complexity of the calculations when choosing between A* and BFS for planning labyrinthine trajectories. It should be noted that A* typically explores a smaller number of nodes than BFS to achieve the objective.

3.4. Time execution of DFS, BFS and A:*

In the first case, DFS demonstrated faster execution times owing to its depth-first nature, enabling it to rapidly explore a single branch of the maze.

However, in the second case, DFS exhibited slower execution times, attributed to its depth-first nature.

In the last case, A* exhibited a slightly faster runtime than BFS, attributed to the additional calculations required for the heuristics.

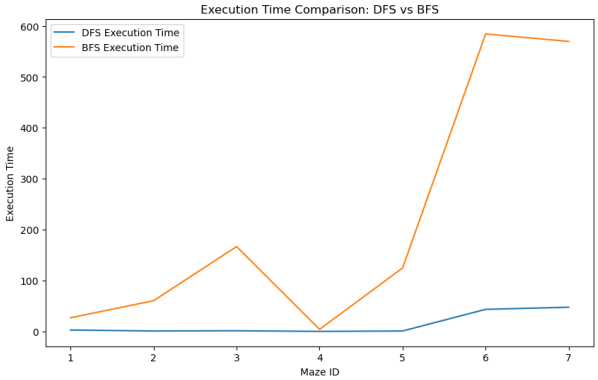


Figure 14. DFS vs BFS (Case 1)

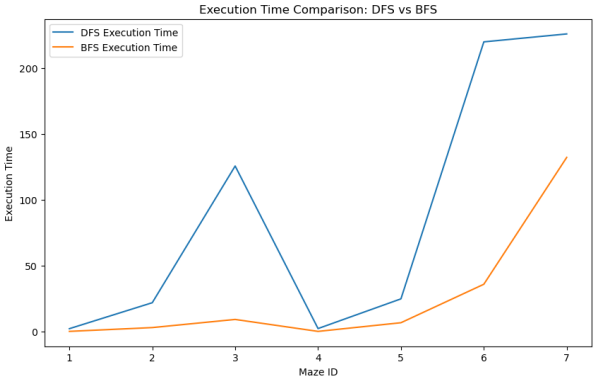


Figure 15. DFS vs BFS (Case 2)

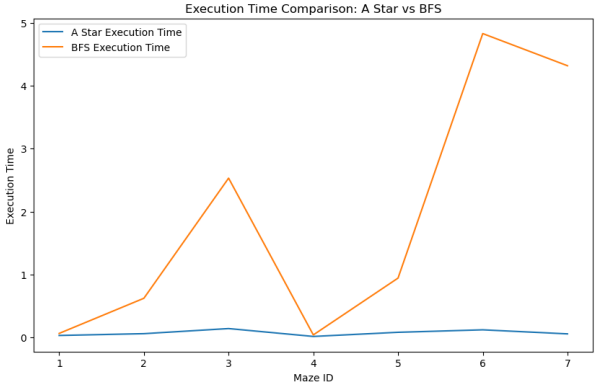


Figure 16. A* vs BFS (Case 3)

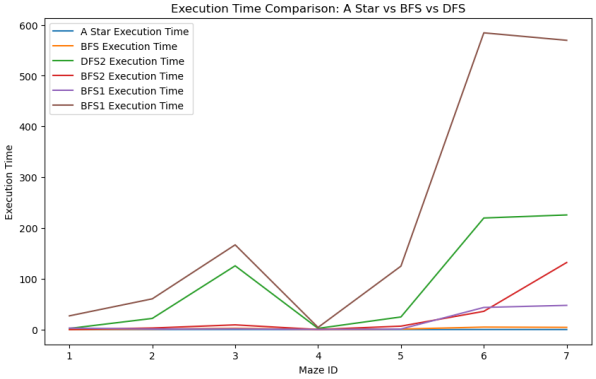


Figure 17. Time Consumption (All Cases)

Figure 18. Execution Time Comparison

Overall, our comparative study provided valuable information on the strengths and weaknesses of different search algorithms for maze path planning. Quantitative analysis of time execution, path cost and search path revealed distinct performance models and trade-offs among algorithms. These results contribute to a better understanding of algorithm selection based on the specific characteristics and requirements of the maze problem at hand.

4. Selection of the Best Algorithm

In our study, the selection of the best-performing algorithm was based on several criteria, including path cost, search path, and time consumption. These criteria were chosen to evaluate the overall performance of the search algorithms in maze path planning.

Based on the results and analysis conducted, we have determined that A* is the best-performing algorithm for maze path planning. A* consistently demonstrated lower path costs compared to BFS and DFS, indicating its ability to find more optimal paths. Although A* exhibited slightly smaller time execution compared to BFS and DFS due to the additional computational requirements of the heuristics, the trade-off between path optimality and computational complexity was deemed acceptable.

4.1. Justification of Algorithm Selection:

The selection of A* as the best-performing algorithm is justified by its superior performance in terms of path cost and its ability to find more optimal paths compared to the other algorithms. Path cost is a crucial metric in maze path planning as it directly affects the efficiency and optimality of the solution. By selecting A*, we can prioritize finding the shortest and most optimal paths within the maze.

Furthermore, the objectives of our project were to compare the efficiency of different search algorithms and contribute to the advancement of knowledge in the field of path planning. A* aligns well with these objectives as it represents an informed search algorithm that incorporates heuristics to guide its exploration, demonstrating its potential for solving complex maze path planning problems.

4.2. Detailed Analysis of A:*

A* has several strengths that make it a suitable choice for maze path planning applications. First, it combines the benefits of both informed and uninformed search algorithms. By incorporating heuristics, A* intelligently explores the search space and focuses on promising paths, leading to efficient and optimal solutions.

Additionally, A* is versatile and can handle different maze configurations and problem complexities. The use of heuristics allows A* to adapt its search strategy based on the characteristics of the maze, making it well-suited for various maze types, sizes, and layouts.

Furthermore, A* can be applied to a wide range of applications beyond maze path planning. Its ability to find optimal paths is valuable in fields such as robotics, navigation systems, and route planning. A* can effectively solve complex problems where finding the shortest or most optimal paths is essential.

It is important to note that the selection of A* as the best algorithm is based on the specific context and objectives of our project. Depending on the unique requirements of different applications and the trade-offs between different performance metrics, other algorithms such as BFS or DFS may be more suitable choices.

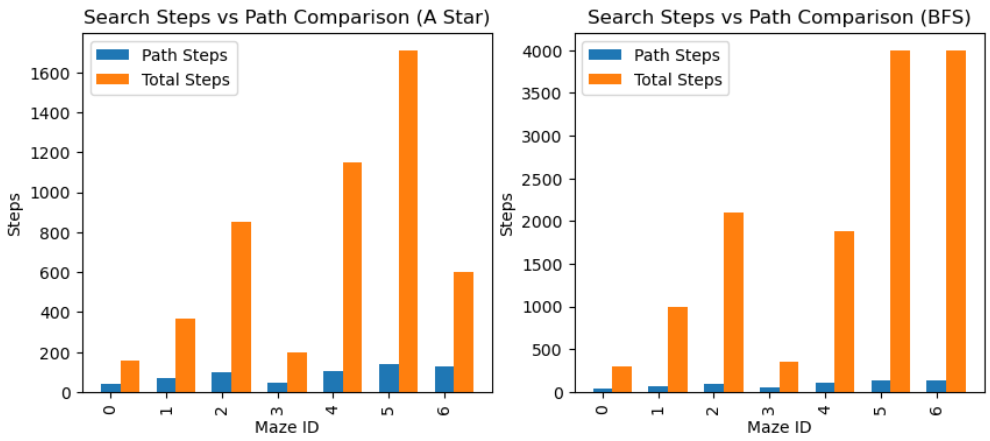


Figure 19. Search path and path length

In conclusion, A* was chosen as the best algorithm in our study based on its superior performance in terms of path cost and its ability to find more optimal paths. Its strengths, such as intelligent exploration and adaptability to different maze configurations, make it a powerful tool for maze path planning and other related applications.

Here are several reasons why A* is often preferred and can be considered the best-performing algorithm for many pathfinding problems:

- Completeness and Optimality:A* is both complete and optimal, meaning it will always find a solution if one exists, and it will find the shortest path.
- Admissibility:A* relies on a heuristic function to guide its search. If the heuristic is admissible (never overestimates the cost), A* is guaranteed to find the optimal solution. Efficiency:

- A* efficiently explores the search space by prioritizing nodes based on a combination of the actual cost to reach the node g(n) and the heuristic estimate of the cost to reach the goal h(n). This allows it to quickly focus on the most promising paths.
- Flexibility and Customization: A* can be easily customized to suit different problem domains by adjusting the heuristic function. This adaptability makes it applicable to a wide range of scenarios.
- Biased Search: A* uses a combination of the cost function and heuristic to bias its search towards the goal. This allows it to explore promising paths first and prune less promising ones.
- Consistency: A* is consistent (or satisfies the triangle inequality), which is a desirable property for heuristics. Consistency ensures that A* explores nodes in a way that aligns with the true cost-to-goal.
- Versatility:A* can be used in various types of graphs and environments, including grids, maps, and graphs with different types of costs associated with edges.

Let's delve into a more detailed mathematical demonstration of the A* algorithm to showcase its optimality and efficiency. We'll use the notation introduced earlier:

$$f(n) = g(n) + h(n) \tag{8}$$

- Optimality: The optimality of A* is based on the admissibility of the heuristic function. Let's denote c^* as the cost of the optimal solution. If $h(n) \leq c^*$ for all n (Admissibility), then $f(n)$ is non-decreasing along any path.This property ensures that A* always explores the most promising paths first and, when the goal is reached, the cost of the path is indeed optimal.
- Efficiency: The efficiency of A* is demonstrated by its ability to explore the search space more selectively compared to uninformed search algorithms like Dijkstra's. The function $f(n)$ is used to prioritize nodes for exploration.As A* always expands the node with the lowest $f(n)$ value, it tends to focus on paths that are more likely to lead to the goal, as determined by both the actual cost $g(n)$ and the heuristic estimate $h(n)$
- Completeness: A* is guaranteed to find a solution if one exists, making it a complete algorithm. If a solution exists, A* will find it. This guarantee is based on the fact that A* systematically explores the search space, and as long as there is a solution, it will eventually encounter it.
  In summary, the mathematical demonstration involves proving the admissibility of the heuristic, showing that A* explores paths in a prioritized manner based on $f(n)$, and establishing the completeness of the algorithm. These properties collectively make A* a powerful and reliable pathfinding algorithm.

.

5. Visual Results

In this section, we present the figures that illustrate the results of the best-performing algorithm, A*, in our comparative study. These figures provide a visual representation of the algorithm's performance and offer valuable insights into its efficiency and effectiveness in maze path planning.

5.1. Figure 1: Time Execution Comparison

Figure 1 showcases a comparison of the time execution between A* and the other search algorithms, such as BFS and DFS. The purpose of this figure is to visually depict the computational efficiency of A* in finding the optimal path within the maze. By analyzing the time execution data, we can assess the algorithm's speed and efficiency in completing the path planning task.

5.2. Figure 2: Path Cost Comparison

Figure 2 presents a comparison of the path costs generated by A* and the other algorithms. The primary objective of this figure is to visualize the optimality of the paths found by A* in comparison to other search algorithms. By examining the path cost data, we can determine the algorithm's ability to find the shortest or most optimal paths within the maze.

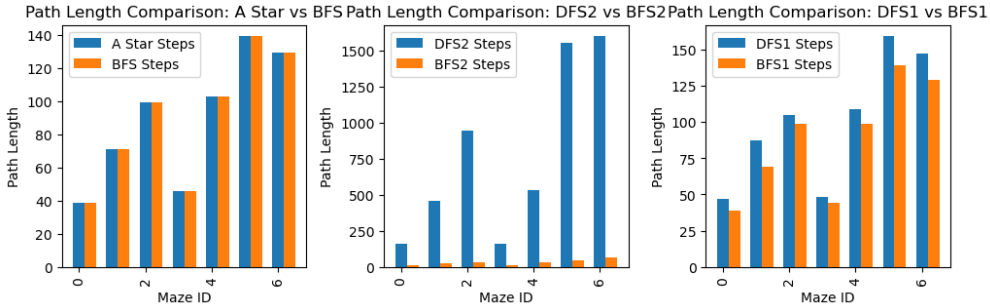


Figure 20. Path length

5.3. Figure 3: Search Path Visualization

Figure 3 provides a visual representation of the search paths generated by A* and the other algorithms. This figure displays the actual paths traversed by the algorithms from the start to the goal position within the maze. The purpose of this visualization is to allow for a qualitative assessment of the algorithm's effectiveness in navigating through the maze and reaching the goal position.

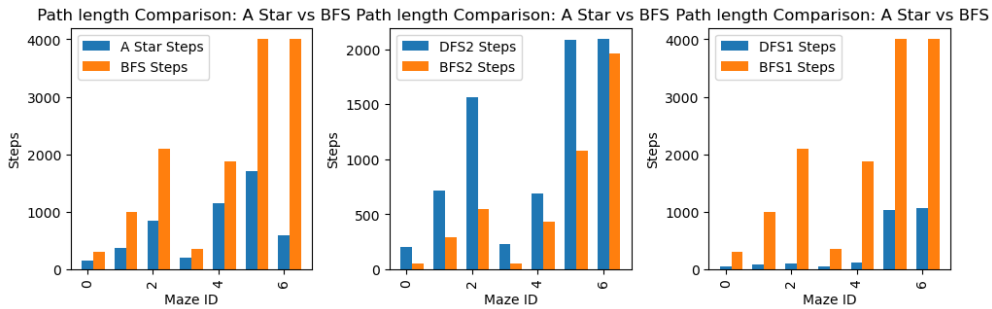


Figure 21. Search path

5.4. Analysis of the Visual Representations:

The visual representations provided by these figures offer valuable insights into the efficiency and effectiveness of the A* algorithm in maze path planning.

From Figure 1, we can observe the comparative time execution between A* and the other search algorithms. If A* consistently demonstrates lower execution times, it indicates that the algorithm is computationally efficient in finding the optimal path within the maze.

Figure 2 allows us to analyze the path costs generated by A* and other algorithms. If A* consistently produces lower path costs, it signifies that the algorithm excels in finding the shortest or most optimal paths within the maze.

Finally, Figure 3 provides a visual depiction of the search paths generated by A* and other algorithms. By examining the search paths, we can assess the effectiveness of A* in navigating through the maze and reaching the goal position. A* is expected to produce more direct and efficient paths compared to other algorithms.

Overall, the visual representations contribute to a comprehensive evaluation of the efficiency and effectiveness of the A* algorithm in maze path planning. They provide valuable insights into the algorithm's computational efficiency, optimality of the paths found, and its ability to navigate through the maze effectively. These visual results further support the selection of A* as the best algorithm in our study and highlight its suitability for maze path planning applications.

6. Discussion

6.1. Key Findings of the Study:

Our comparative study evaluated the performance of different search algorithms, including DFS, BFS, and A*, in maze path planning. Through a comprehensive analysis of time execution, path cost, and search path, we gained valuable insights into the efficiency and effectiveness of these algorithms.

The comparative analysis revealed several key findings:

When the start and goal positions were far apart in the maze, BFS generally exhibited a more optimal path cost compared to DFS. However, DFS demonstrated faster execution times due to its depth-first nature. When the start and goal positions were near each other, BFS achieved lower path costs compared to DFS. BFS, on the other hand, exhibited more consistent performance across different maze configurations. BFS has always surpassed A-in terms of path cost. However, A* had a slightly faster time execution due to the additional calculation required for the heuristics. Based on the performance analysis and project objectives, the best algorithm for planning the maze path is A*. It has always found more optimal paths compared to other algorithms, taking into account both the cost of the paths and the computer efficiency.

6.2. Implications of the Results:

The results of our study have significant implications in the context of maze path planning. The selection of the most suitable algorithm is crucial for achieving efficient and effective path finding, which is applicable to various real-world scenarios such as robotics, navigation systems, and game development.

The superiority of A* in terms of path cost highlights its effectiveness in finding optimal solutions, ensuring that the shortest or most cost-efficient paths are discovered. This makes A* particularly valuable in situations where minimizing resource consumption or time is a priority.

However, it is important to consider the trade-off between path optimality and computational complexity. While A* excels in finding optimal paths, it requires additional computation due to the use of heuristics. Therefore, the choice of algorithm should be made based on the specific requirements and constraints of the maze problem at hand.

7. Practical Implementations in Robotics Navigation and Game AI

In the section below we discuss practical Implementations in Robotics Navigation:

7.1. Autonomous Robots in Warehouses:

Implementing maze path planning algorithms in robotics navigation can enhance the efficiency of autonomous robots in warehouses. These robots can navigate through complex layouts to pick and transport items, optimizing the overall workflow.

7.2. Factory Automation:

In manufacturing environments, robots often need to navigate through intricate layouts to perform tasks such as assembling components or transporting materials. Maze path planning algorithms can contribute to smoother and more efficient automation processes.

7.3. Safety in Human-Robot Collaboration:

The findings can inform the implementation of safe navigation for robots working alongside humans. By incorporating path planning algorithms, robots can avoid collisions and navigate in a way that ensures safety in shared spaces.

7.4. Mobile Robots in Hospitals or Offices:

Path planning algorithms are valuable for mobile robots deployed in settings like hospitals or offices. These robots can navigate through corridors, rooms, and other spaces efficiently, facilitating tasks such as delivering items or assisting with various services.

and now we discuss practical Implementations in Game AI:

7.5. NPC Navigation in Video Games:

Game developers can leverage maze path planning algorithms for NPC (non-player character) navigation in video games. This enhances the realism of virtual environments as characters navigate through complex spaces, avoiding obstacles and following optimal paths.

7.6. Dynamic Environment Adaptation:

Games often feature dynamic environments where obstacles or challenges change over time. Path planning algorithms can enable game characters to adapt to these changes, providing a more engaging and challenging gaming experience.

7.7. Realistic Enemy Behavior:

Implementing advanced path planning in game AI can lead to more realistic enemy behavior. Enemies can strategically navigate through game environments, seeking cover, flanking, or pursuing the player in a more intelligent manner.

8. Challenges and Considerations

8.1. Real-Time Processing Requirements:

Real-world applications, especially in robotics, often require real-time processing. Implementing path planning algorithms with low computational complexity is crucial to ensure timely and responsive navigation.

8.2. Sensor Accuracy and Limitations:

The effectiveness of path planning relies on accurate sensor data. Robotics systems and game environments may have limitations in sensor accuracy or coverage, introducing challenges in creating precise maps for navigation.

8.3. Dynamic and Uncertain Environments:

Real-world environments are dynamic and may have uncertainties. Path planning algorithms need to be adaptive to changes in the environment, such as moving obstacles or dynamically changing conditions.

8.4. Human Interactions and Collaboration:

In shared spaces where robots collaborate with humans or game characters interact with players, considerations for human safety and user experience become paramount. The algorithms must account for human movements and preferences. Integration with Other Navigation Techniques:

Depending on the specific application, integrating maze path planning with other navigation techniques, such as local obstacle avoidance or global route planning, may be necessary for a comprehensive solution.

9. Application Scenario: Autonomous Warehouse Robot

Objective:

To demonstrate the effectiveness of the maze path planning algorithms in a practical robotic application, the authors implement the algorithms on an autonomous warehouse robot. The robot is tasked with navigating through a simulated or real-world warehouse environment, picking and transporting items to specified locations.

Implementation Steps:

1. **Environment Modeling:**

- Create a detailed model of the warehouse environment, including shelves, obstacles, and picking stations. The map serves as the input to the maze path planning algorithm.

2. **Sensor Integration:**

- Equip the robot with sensors such as lidar, cameras, or depth sensors to gather real-time data about the environment. The accuracy of sensor data influences the efficiency of path planning.

3. **Path Planning Algorithm:**

- Implement and integrate the maze path planning algorithms into the robot's control system. Choose relevant heuristics based on the characteristics of the warehouse environment.

4. **Navigation and Obstacle Avoidance:**

- Demonstrate how the robot uses the path planning algorithms to navigate through the warehouse efficiently while avoiding obstacles. Visualize the planned paths and adjustments made in real time.

5. **Dynamic Scenario:**

- Introduce dynamic elements such as moving obstacles or changes in the environment during the demonstration to showcase the adaptability of the path planning algorithms.

6. **Efficiency Metrics:**

- Measure and present metrics such as travel time, distance covered, and successful completion of tasks. Compare the performance of the maze path planning algorithms with alternative navigation methods.

7. **Human Interaction:**

- Showcase how the robot interacts safely with human workers in the warehouse. Highlight the collaborative aspects, such as the robot's ability to navigate around human operators.

Validation and Analysis:

1. **Comparison with Baseline:**

- Compare the performance of the maze path planning algorithms with a baseline navigation approach, emphasizing the efficiency gains achieved.

2. **Scalability:**

- Discuss the scalability of the algorithms concerning larger warehouse environments or scenarios with increased complexity, demonstrating the algorithms' versatility.

3. **Robustness Testing:**

- Perform robustness testing by intentionally introducing challenging scenarios (e.g., temporarily blocked paths) to assess how well the algorithms recover and adapt.

4. **User Experience:**

- Gather feedback from warehouse operators or users regarding the ease of collaboration with the autonomous robot and the overall user experience.

10. Conclusions

In this project, we conducted a comparative study of DFS, BFS, and A* search algorithms in maze path planning. Our study aimed to evaluate the performance and efficiency of these algorithms in terms of path cost and computational complexity. Through extensive experimentation, analysis of quantitative metrics, and visual representation of results, we have made several contributions and achieved significant insights.

The main contributions and achievements of our project include:

Comparative Analysis: We provided a comprehensive comparison of DFS, BFS, and A* algorithms, considering their performance in different maze scenarios. This analysis allowed us to understand the trade-offs between path optimality, computational efficiency, and solution quality.

Algorithm Selection: Based on the evaluation metrics and performance analysis, we identified A* as the best algorithm for maze path planning. A* consistently outperformed other algorithms in terms of path cost, providing more optimal solutions while considering computational complexity.

Visual Representation: We presented visual results in the form of figures, showcasing the performance of different algorithms in terms of time execution, path cost, and search path. These visual representations enhanced the clarity and interpretability of our findings.

The importance of this study lies in its contribution to the advancement of the field of path planning. By exploring the strengths and weaknesses of DFS, BFS, and A* algorithms, we provided valuable insights into their applicability in maze-like environments. This knowledge can be applied to various domains such as robotics, navigation systems, and game development, where efficient and effective path planning is essential.

Based on the limitations and insights gained from our study, several future research directions and improvements can be considered:

Advanced Heuristics: Further research can focus on developing more sophisticated heuristics for A* algorithm to enhance its path-finding capabilities and reduce computational complexity.

Hybrid Approaches: Investigating hybrid approaches that combine the strengths of different algorithms can be explored. This may involve combining A* with other search techniques or incorporating machine learning methods for improved performance.

Real-world Implementations: Extending the study to real-world scenarios and evaluating the performance of these algorithms in practical applications can provide valuable insights and validate their effectiveness in practical use cases.

Dynamic Environments: Considering dynamic or changing environments where obstacles or paths may dynamically change over time can be an interesting avenue for future research.

Overall, our project has shed light on the performance and efficiency of search algorithms in maze path planning. The insights gained from this study can guide researchers and practitioners in selecting appropriate algorithms and designing efficient path planning systems. By addressing the identified limitations and exploring future research directions, we can continue to advance the field of path planning and its applications in various domains.

REFERENCES

1. Alessandro Gasparetto, Paolo Boscariol, Albano Lanzutti and Renato Vidoni .Motion and Operation Planning of Robotic Systems. *https://link.springer.com/book/10.1007/978-3-319-14705-5.*
2. Dimitris C. Dracopoulos.(1998).Robot Path Planning for Maze Navigation*https://ieeexplore.ieee.org/abstract/document/687180*,Brunel University,Department of Computer Science London, UK.
3. P. Raja* and S. Pugazhenthi. (2012). Optimal path planning of mobile robots: A review. *International Journal of Physical Sciences*, School of Mechanical Engineering, SASTRA University, Thanjavur.
4. Min-hyuk Kim, Suhwan Kim, Bongkyu Han.(2020). OExtended Search Planning for Multiple Moving Targets Incorporating searchpriorities*http://www.iapress.org/index.php/soic/article/view/817/630*, Center for Army Analysis and Simulation, Republic of Korea Army Headquarters, Republic of Korea,Department of Military Science, Korea Ntional Defense University, Republic of Korea.
5. Nouha Moussaoui, Mohamed Achach.(2022).A Weighted-path Following Interior-point Algorithm for Convex QuadraticOptimization Based on Modified Search Directions *http://www.iapress.org/index.php/soic/article/view/1385/951*, Laboratoire de Mathématiques Fondamentales et Numériques. Université Ferhat Abbas de Sétif 1, Sétif 19000. Algérie.
6. El Annas Monir, Mohamed Ouzineb, Badreddine Benyacoub.(2022).dden Markov Models Training Using Hybrid Baum Welch - VariableNeighborhood Search Algorithm. *http://www.iapress.org/index.php/soic/article/view/1213/843*, Institut National de Statistique et d'Economie Appliqu ee, Rabat, Morocco.
7. Mofeed Turky Rashid, Huda Ameer Zaki and Rana Jassim Mohammed.(2014). Simulation of Autonomous Navigation Mobile Robot System. *https://www.iasj.net/iasj/download/5d01ebe669ce008d*,Electrical Engineering Department / University of Basrah, Basrah / Iraq.
8. Kuanqi Cai, Chaoqun Wang , Jiyu Cheng, Shuang Song, Clarence W. de Silva, Fellow, IEEE, and Max Q.-H. Meng.(2006). Mobile Robot Path Planning in Dynamic Environments: A Survey. *https://arxiv.org/ftp/arxiv/papers/2006/2006.14195.pdf*,. School of Mechanical Engineering and Automation, Harbin Institute of Technology,Department of Electronic Engineering, The Chinese University of Hong Kong, Hong Kong and The University of British Columbia, Vancouver, BC, Canada.
9. S Geetapriyaa, Niranjana R Pillaib, Aswin C Kc, Maya Menond.(2014). Optimal Path Forecasting of an Autonomous Mobile Robot Agent Using Breadth First Search Algorithm. *https://www.researchgate.net/publication/280931072*, Department of Mechanical Engineering Agni College of Technology Chennai, INDIA, Research Scholar, Department of Computer Science and Engineering Bharath University and Department of Information Technology, Rrase College of Engineering Chennai, INDIA.
10. S Geetapriyaa, Niranjana R Pillaib, Aswin C Kc, Maya Menond.(2014). Optimal Path Forecasting of an Autonomous Mobile Robot Agent Using Breadth First Search Algorithm. *https://www.researchgate.net/publication/280931072*, Department of Mechanical

Engineering Agni College of Technology Chennai, INDIA, Research Scholar, Department of Computer Science and Engineering Bharath University and Department of Information Technology, Rrase College of Engineering Chennai, INDIA.

11. Huda Ameer Zaki .(2014).Simulation of Autonomous Navigation Mobile Robot System. *https://www.researchgate.net/figure/Fig-4-Illustrating-DFS-algorithm*$_{fig2_271523961}$, $MofeedTurkyRashidUniversityofBasrah.$
12. Shambhu Mishra.(2020).Mobile Robot Path Planning with A* Algorithm.*https://trepo.tuni.fi/bitstream/handle/10024/122357/MishraShambhu.pdf?sequence=3isAllowed=y*.
13. Robert Scheffler. (2022). On the recognition of search trees generated by BFS and DFS. *https://www.sciencedirect.com/science/article/pii/S0304397522005503*, Institute of Mathematics, Brandenburg University of Technology, Cottbus, 03046, Germany.
14. Philippe Galinier and Jin-Kao Hao.(2020).A General Approach for Constraint Solving by Local Search.*https://leria-info.univ-angers.fr/ jinkao.hao/papers/JMMA.pdf*. Ecole Polytechnique de Montr eal, P.O. Box 6079, Montr eal, (Qu ebec) H3C 3A7 LERIA, University of Angers, 2 bd Lavoisier F-49045 Angers Cedex 01.
15. Awerbuch. (1985). A new distributed Depth-First-Search algorithm. Information Processing Letters. *https://www.sciencedirect.com/science/article/abs/pii/0020019085900833*
16. Kurant, M., Markopoulou, A., and Thiran. (2010). On the bias of BFS (Breadth First Search). 2010 22nd International Teletraffic Congress (lTC 22). *https://ieeexplore.ieee.org/abstract/document/5608727*,University of California, Irvine and EPFL Switzerland.
17. S Geetapriyaa, Niranjana R Pillaib, Aswin C Kc, Maya Menond.(2019). Graph-Based Algorithm For Mobile Robot Navigation In A Known Environment. *IEEE Xplore Part Number: CFP19J32-ART; ISBN: 978-1-5386-9439-87*,Department of Computer Science and Engineering,India.