

Performance Analysis of Search Algorithms in Grid Environment

Nguyen Khanh Toan

Swinburne University of Technology

Melbourne, Australia

104180605@student.swin.edu.au

I. Abstract

This research investigates the performance of various search algorithms in the context of search algorithms in grid environments. The study evaluates Breadth-First Search (BFS), Depth-First Search (DFS), Greedy Best-First Search (GBFS), A* (A star), and bi-directional search variants of BFS and A* across multiple grid maps with varying complexities. Execution times for each algorithm on different maps are recorded and analyzed to determine their efficacy in finding optimal paths while navigating obstacles. The results highlight A* as a consistency and efficient choice for pathfinding tasks, demonstrating superior performance in terms of speed and reliability compared to other algorithms tested. These findings contribute to advancing the understanding and application of search algorithms in real-world robotic navigation scenarios.

II. Introduction

In the realm of artificial intelligence and robotics, efficient pathfinding algorithms are crucial for enabling autonomous navigation in complex environments. This research aims to compare the performance of several search algorithms—Breadth-First Search (BFS), Depth-First Search (DFS), Greedy Best-First Search (GBFS), A* (A star), and custom variations of bi-directional search (CUS1 and CUS2)—across multiple grid maps. The evaluation criteria include execution time measured in seconds for each algorithm on various test maps, aiming to identify which algorithm performs best in terms of speed and reliability.

III. Methodology

The experiment utilizes Python programming language to implement the search algorithms and measures their performance on 11 different grid maps (map0 to map10). Each algorithm's runtime is recorded in seconds for completing pathfinding tasks from a start position to a goal while navigating around obstacles. The averages of execution times across all maps are calculated to provide a comprehensive comparison.

A comprehensive explanation, implementation details, and complexity analysis of all the algorithms are thoroughly presented in the Robot Navigation Documentation.[1][2]

The program for measuring is from the “Robot Navigation” custom project.

```

PS D:\Swinburne-Studying\COS30002-AI_for_Games\Task-19-Cus\Robot Navigation> py main.py map0.txt bfs
pygame 2.5.2 (SDL 2.28.3, Python 3.12.0)
Hello from the pygame community. https://www.pygame.org/contribute.html
Type 'display' to the command prompt for the display

Search Strategy: BFS
Final Node: <Node (7, 0)>
Number of Explored nodes: 33
Robot Instructions:
['Down', 'Right', 'Right', 'Right', 'Right', 'Up', 'Up', 'Right', 'Right', 'Right']
Run time in average: 0.64 milliseconds

```

The specification used for measuring:

CPU: Intel(R) Core(TM) i5-11400H @2.7GHz (12CPUs)

Memory: 8192 MB RAM

GPU: NVIDIA GeForce RTX 3050 Laptop GPU 8GB VRAM

IV. Results

To evaluate the average performance of each algorithm, I perform 10 iterations for each predefined test case in the map set. Then I compute the average time taken across all 10 iterations to obtain an overall performance score for each algorithm. This method helps reduce biases by incorporating results from multiple test cases, offering a more comprehensive and representative assessment of the algorithms' effectiveness.

Overview of Performance Metrics:

- **BFS:** Breadth-First Search
- **DFS:** Depth-First Search
- **GBFS:** Greedy Best-First Search
- **A*:** A-Star Search
- **CUS1:** Bi-directional BFS
- **CUS2:** Bi-directional A*

Algo\Test	map0	map1	map2	map3	map4	map5	map6	map7	map8	map9	map10*	Average
BFS	0.47	0.41	3.43	0.74	54.92	27.54	0.40	19.14	7.93	23.40	0.48	12.62
DFS	0.48	0.42	3.80	0.85	3886.14	6.24	1.01	17.41	26.78	11.32	0.44	359.54
GBFS	0.53	0.43	1.15	0.63	1.05	4.37	0.46	5.63	1.01	2.50	0.40	1.56
A*	0.57	0.39	1.59	0.78	3.38	2.40	0.54	18.45	2.49	12.14	0.39	3.83
CUS1	0.69	0.46	3.52	0.98	25.25	26.25	0.43	8.84	6.11	11.08	0.48	7.55
CUS2	0.64	0.62	1.76	0.95	3.77	1.90	0.41	9.91	2.91	17.13	0.38	3.67

Figure 1: Runtime measurement in average of algorithm in certain map

*Note: Map 10 is created with initial node and goal node at the same position. This help indicated the runtime of the function in common progress and checked if the function would successfully pass this test case.

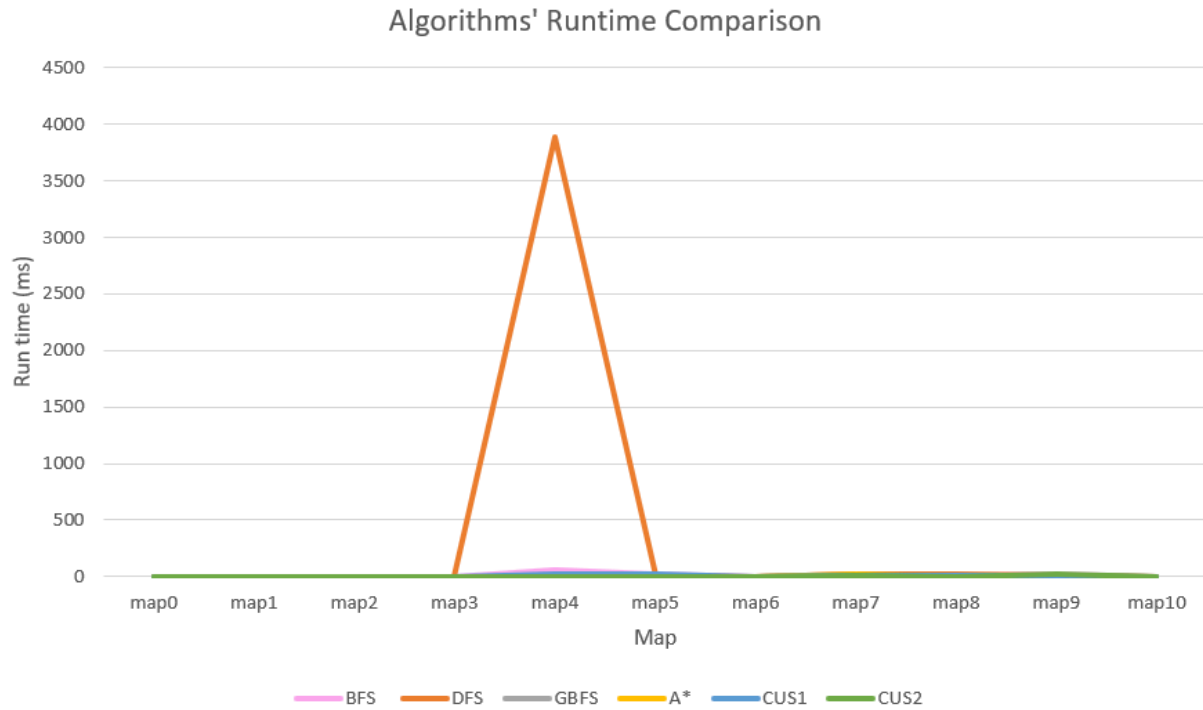


Figure 2: Algorithms' runtime comparison

In Figure 2, it's evident that the runtime of DFS on map 4 is an outlier, with a value nearing 4000ms. To provide a more comprehensive analysis and allow for clearer comparison of the other algorithms, I chose to set the runtime value of DFS for map 4 to 0. This adjustment helps in excluding the skewed data point and better demonstrates the performance of the other algorithms across the maps, facilitating a more balanced visual and analytical assessment.

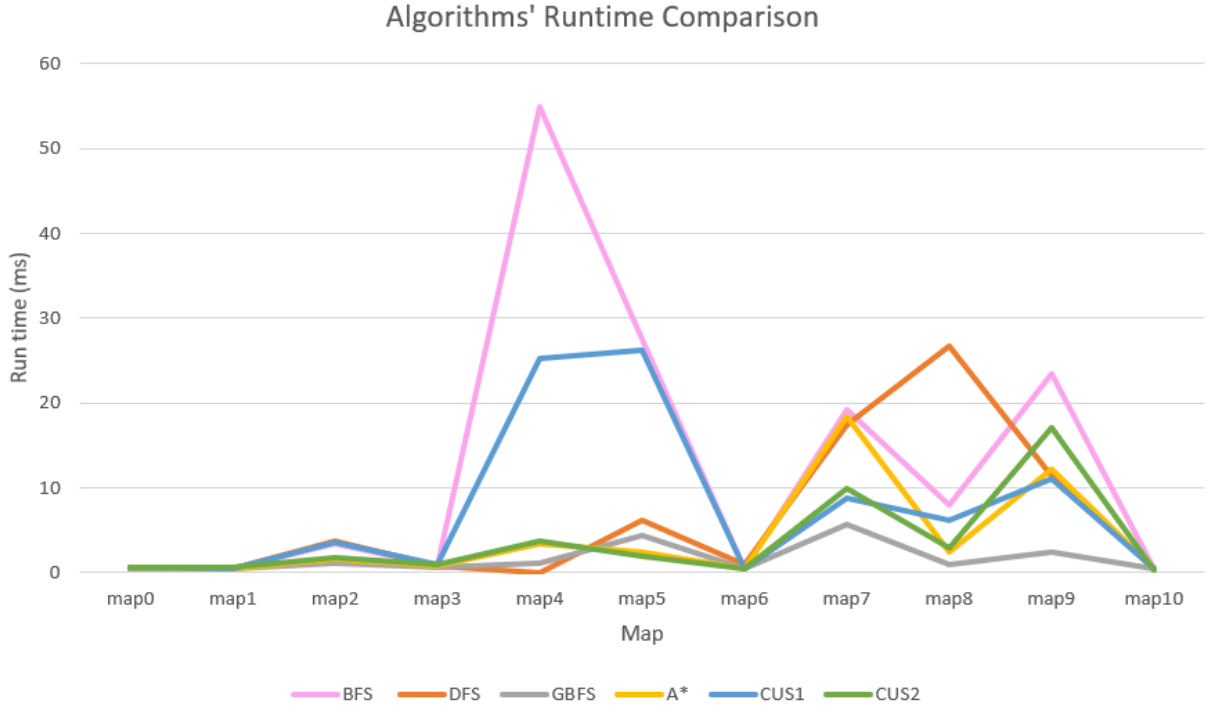


Figure 3: Algorithms' runtime comparison (DFS map 4 = 0)

V. Discussion

Summary of Trends in Algorithms' Runtime Comparison

The provided line chart illustrates the runtime performance of six search algorithms—BFS, DFS, GBFS, A*, CUS1, and CUS2—across 11 different maps. The runtime is measured in milliseconds (ms) and is plotted against each map, labeled from map0 to map10.

A significant trend observed is the consistently low and stable runtime of GBFS, A*, and CUS2 across most maps, indicating efficient performance in a variety of scenarios. In contrast, DFS demonstrates notably unusual behavior, with a dramatic spike on map4 where the runtime is nearly 4000 ms (Figure 2), making it an outlier. Similarly, BFS shows a substantial increase in runtime on map4 and map5, with peaks reaching above 50 ms and 20 ms respectively, suggesting inefficiencies in handling more complex or larger maps.

CUS1 shows moderate performance, with occasional spikes such as on map4, map5 and map9, indicating variability in its runtime efficiency depending on the map characteristics. The other algorithms (GBFS, A*, and CUS2) show much lower runtime variations and consistently better performance, particularly on maps with straightforward or fewer obstacles.

Overall, the chart highlights GBFS, A* and CUS2 as the most efficient and stable algorithms across different maps, while BFS and DFS exhibit higher runtimes, particularly in maps with greater complexity, making them less suitable for such environments.

Detailed Comparison and Analysis

1. BFS (Breadth-First Search)

- **Performance:** BFS shows moderate performance across most maps, with a significant increase in time on “*map4*” (54.92 ms) and “*map5*” (27.54 ms), leading to a relatively high average time of 12.62 ms.
- **Analysis:** BFS explores all possible nodes at the present depth before moving on to nodes at the next depth level. This can be efficient for small maps or shallow goals but can become costly for larger, more complex maps like “*map4*” and “*map5*”. The high time on these maps indicates that BFS might be traversing a large number of unnecessary nodes before reaching the goal.

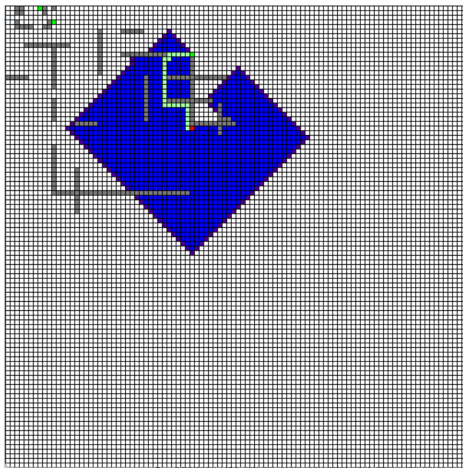


Figure 4: map 4 for BFS

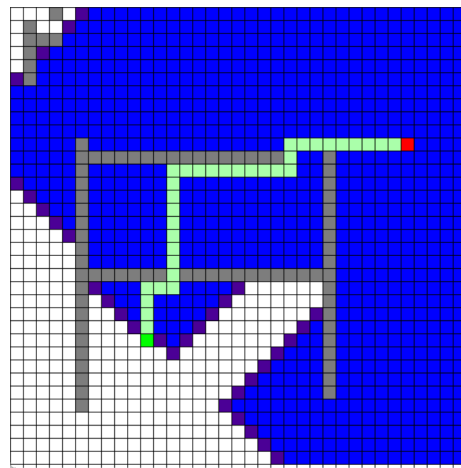


Figure 5: map 5 for BFS

2. DFS (Depth-First Search)

- **Performance:** DFS exhibits extremely poor performance on “*map4*” (3886.14 ms), leading to a very high average time of 359.54 ms.
- **Analysis:** DFS can get stuck exploring long, irrelevant paths before finding a solution. The massive time taken on “*map4*” suggests that DFS is heavily penalized by deep, complex maps where it must backtrack extensively. This makes DFS less suitable for large and complex maps where the goal is not immediately reachable.

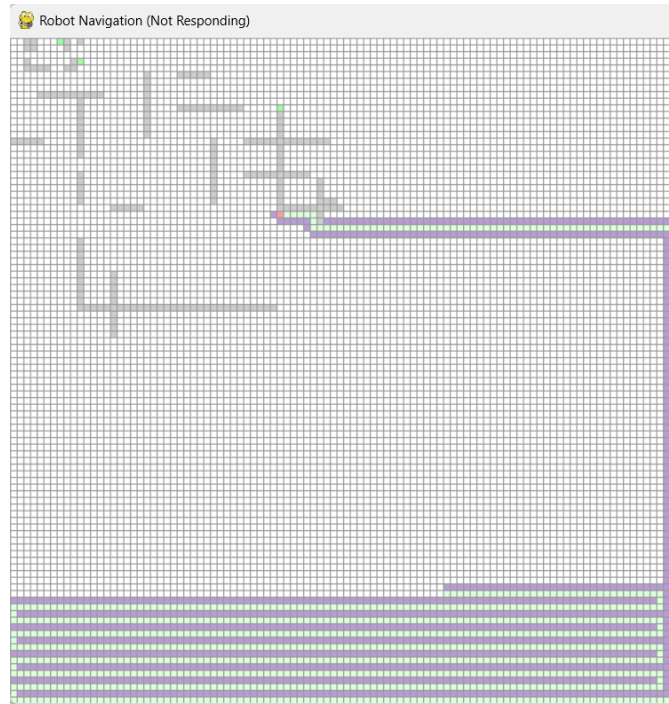


Figure 6: map 4 for DFS (Cannot record all due to crashed)

3. GBFS (Greedy Best-First Search)

- **Performance:** GBFS performs exceptionally well on most maps, with a very low average time of 1.56 ms, notably outperforming other algorithms.
- **Analysis:** GBFS prioritizes exploring the most promising paths first by using a heuristic, which makes it efficient in quickly finding solutions for most maps. Its consistently low times across different maps indicate that it effectively avoids unnecessary exploration, which is particularly beneficial for maps where the goal is somewhat predictably located. However, GBFS still has some drawbacks when underestimate the true cost, which is discussed in “map9” of Figure 12.

4. A* Search

- **Performance:** A* offers a balanced performance, with a reasonable average time of 3.83 ms, showing efficiency and consistency across most maps.
- **Analysis:** A* combines the strengths of BFS and GBFS by considering both the cost to reach the node and the estimated cost to the goal. This makes it a robust choice for diverse maps, offering a good balance between exploration and efficiency. The slightly higher time on “map7” (18.45 ms) may indicate areas where the map complexity challenges A*'s balance between cost and heuristic.

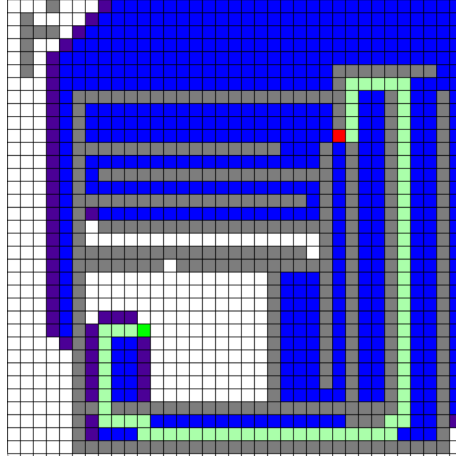


Figure 7: map 7 for Astar

5. CUS1 (Bi-directional BFS)

- **Performance:** CUS1 has varied performance, with relatively high times on “*map4*” (25.25 ms) and “*map5*” (26.25 ms), leading to an average time of 7.55 ms.
- **Analysis:** CUS1 employs a bi-directional search strategy, which involves simultaneously searching from both the start and goal points. This approach is generally effective in reducing search time by meeting in the middle. However, the performance drop on maps like “*map4*” and “*map5*” indicates that CUS1 may struggle with large map scale or intricate goal locations. The increased times suggest that bi-directional BFS might face challenges with inefficient node expansion when handling larger or more complicated maps, potentially due to the difficulty in managing multiple search fronts and coordinating their intersection.

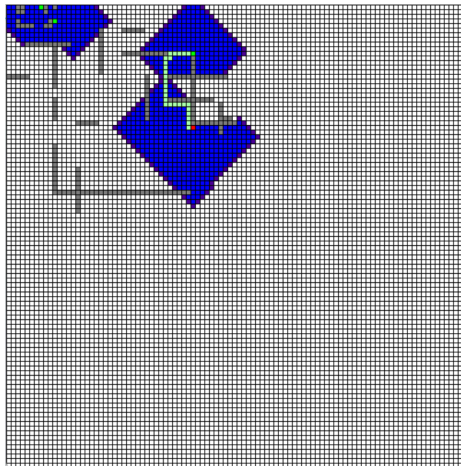


Figure 8: map4 for CUS1

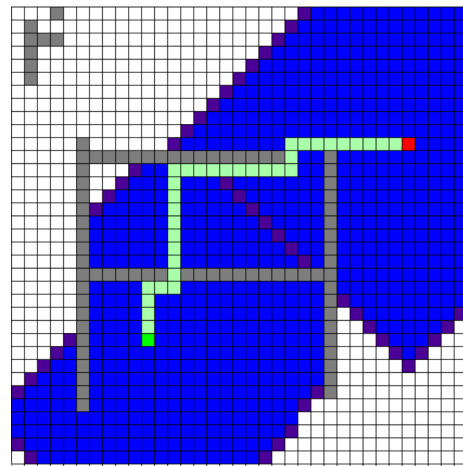


Figure 9: map5 for CUS1

6. CUS2 (Bi-directional A star)

- **Performance:** CUS2 generally performs well, with an average time of 3.67 ms. It shows stability across maps, except for a few spikes like “*map7*” (9.91 ms) and “*map9*” (17.13 ms).

- Analysis:** CUS2 utilizes a bi-directional A* search, combining the strengths of A*'s heuristic-driven pathfinding with a simultaneous search from both the start and goal points. This approach typically provides a balanced and efficient solution for diverse map challenges. The consistently reasonable times suggest that CUS2 is good at leveraging heuristics for efficient pathfinding. However, the elevated times on “*map7*” and “*map9*” indicate that certain map features, such as complex obstacle arrangements or large space to search, might pose difficulties and ineffectiveness. These complexities could result in increased search effort and time as the algorithm navigates the terrain and reconciles the two search fronts.

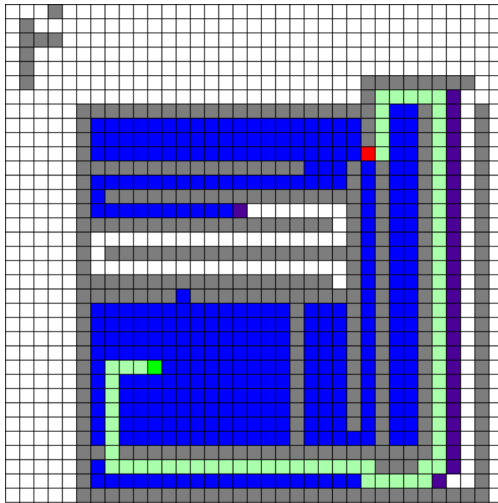


Figure 10: Map 7 for CUS2

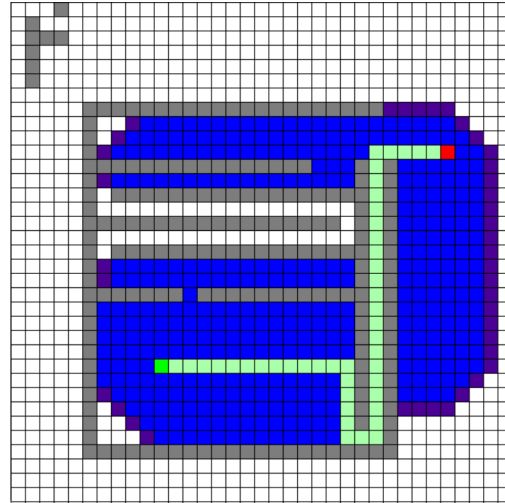
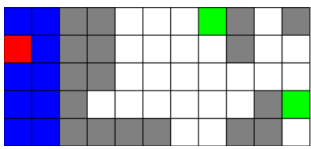
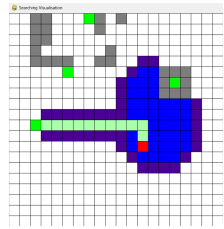
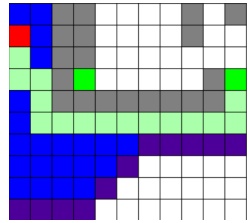
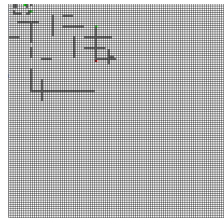
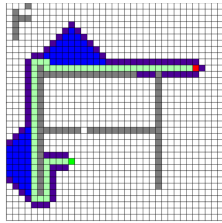
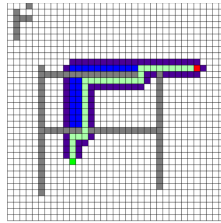


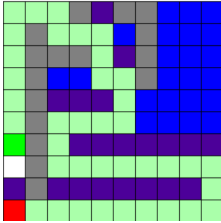
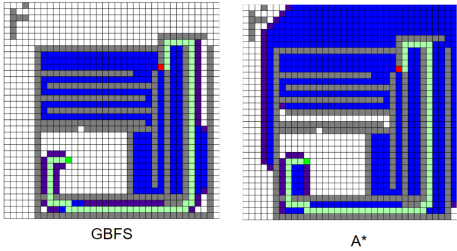
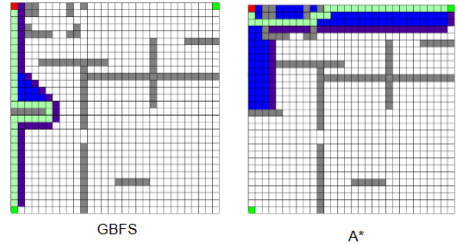
Figure 11: Map 8 for CUS2

Further discussion and comparison of remarkable point between the algorithms on each map will be detailed in the next section

Detailed explanation and purpose for each map creation

Map for test case	Purpose Brief	Discuss
map0	Validate basic program functionality.	The program successfully perform the map0
map1	Test inability to reach the goal.	The program effectively handles: <ul style="list-style-type: none"> - Obstructing grid elements - Possible action for the node - Terminating the algorithm when no nodes remain in the frontier 
map2	Assess handling of blocked goal nodes.	When a goal is completely surrounded: <ul style="list-style-type: none"> - Initial nodes cannot reach the goal. - The heuristic function is successfully implemented, expanding nodes closest to

		<p>the goal while avoiding explored or frontier nodes.</p> 
map3	<p>Test ability to choose closest goal node with heuristic validation</p>	<p>Designed to validate the heuristic function and goal test function when the nearest goal cannot be reached:</p> <ul style="list-style-type: none"> - The program successfully switches to another goal after exploring all nodes near the first goal. - The goal test consistently checks each goal in the list, allowing all goals to be evaluated within a state check. 
map4	<p>Display visual can handle large grid</p>	<p>The grid and its visuals effectively manage large scales, with each cell capable of scaling up or down.</p> 
map5	<p>Test hidden entrance detection for GBFS and A*.</p>	<p>This case examines the behavior of GBFS and A* when a small entrance is hidden:</p> <ul style="list-style-type: none"> - GBFS takes a longer path due to not recognizing the small entrance. - A* identifies the shorter path by always checking the cost to the current state, switching to a lower-cost node if it found that current node costly than the node in priority queue. <div style="display: flex; justify-content: space-around; align-items: flex-end;"> <div style="text-align: center;">  <p>GBFS</p> </div> <div style="text-align: center;">  <p>A*</p> </div> </div>

map6	Assess algorithm preference for path length with nearby goals.	<p>This test checks if algorithms take a longer path despite a nearby goal:</p> <ul style="list-style-type: none"> - All algorithms except DFS choose the short path. - This happens because the DFS algorithm follows an action order that prioritizes exploring nodes in the sequence: up, left, down, and right. When the algorithm retrieves a node from the stack, it always processes the last element added, which is the right node. As a result, even though the shorter path involves moving upward, DFS instead takes the rightward path due to its prioritization, leading it on a longer route to reach the goal.. 
map7	Compare short and long path selection by GBFS and A*.	<p>This test assesses GBFS and A* performance:</p> <ul style="list-style-type: none"> - In this test, both algorithms find a short path to the goal. - However, GBFS takes a direct route to the goal, exploring fewer nodes and taking less time compared to A* due to not considering path cost. 
map8	Evaluate goal selection when two goals are equal distance. (1 goal right and 1 goal in under)	<p>This test checks the order of action priorities:</p> <ul style="list-style-type: none"> - All algorithms except DFS expand the lower goal first, similar to the reason discussed in map6. - A* selects the right goal after finding it has a lower cost than the downward path. 

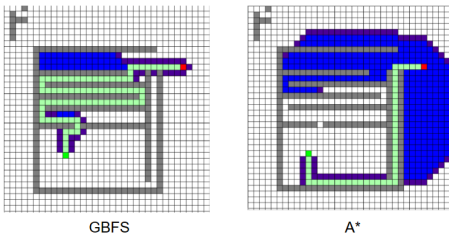
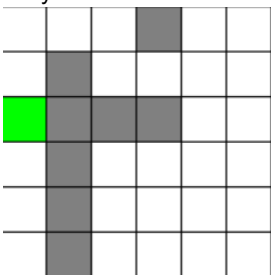
map9	Examine GBFS disadvantages in path selection.	<p>This test aims to highlight GBFS limitations:</p> <ul style="list-style-type: none"> - GBFS, although expanding fewer nodes, selects a longer path to the goal. - A* explores more nodes but chooses the shorter path.  <p style="text-align: center;">GBFS A*</p>
map10	Test recognition of the initial node as the goal node.	<p>This case tests if the initial state being the same as the goal state is recognized:</p> <ul style="list-style-type: none"> - The algorithms successfully identify this special case by checking the goal state from the start, confirming they don't expand unnecessarily. 

Figure 12: Purpose of map for test cases

VI. Conclusion

In this research, I evaluated the performance of several search algorithms, including BFS, DFS, GBFS, A*, Bi-directional BFS (CUS1), and Bi-directional A* (CUS2), across various grid environments. A* emerged as the most robust and efficient algorithm, consistently outperforming others in terms of speed and reliability due to its balanced consideration of path cost and heuristic estimates. While GBFS also demonstrated strong performance, particularly on maps where the goal was predictably located, however, due to “greedy” characteristics, it might have struggled with longer paths when the heuristic underestimated costs, this might lead to execution costs soaring up, especially in real life application. The bi-directional approaches, CUS1 and CUS2, showed promising results by reducing search time but faced challenges on larger maps with complex obstacles.

Overall, this analysis highlights the importance of selecting the appropriate search algorithm based on the specific navigation context, with A* being a versatile choice for most scenarios. These insights are valuable for advancing autonomous navigation technologies and optimizing pathfinding in complex environments.

VII. Bibliography

[1] K. T. Nguyen, "Robot Navigation Search Documentation," Jun. 13, 2024.

[https://github.com/ToanNguyenSwinAdventure/COS30002-104180605/tree/main/19%20-%20Do%20-%20Custom%20Project%20\(D_HD\)%20Documents/Robot%20Navigation](https://github.com/ToanNguyenSwinAdventure/COS30002-104180605/tree/main/19%20-%20Do%20-%20Custom%20Project%20(D_HD)%20Documents/Robot%20Navigation)

[2] S. Russel and P. Norvig, *Artificial intelligence: a Modern approach.*, 4th ed. Prentice Hall, 2021.