Map Pathfinding Visualizer - A Comparative Study

A PROJECT REPORT

Submitted by

BL.EN.U4CSE22143	Pranav Gokhale
BL.EN.U4CSE22155	Sriharsha vvs
BL.EN.U4CSE22162	Viraj Kisan Daule
BL.EN.U4CSE22169	Pranav Biju Nair

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

AMRITA SCHOOL OF ENGINEERING, BANGALORE

AMRITA VISHWA VIDYAPEETHAM

BANGALORE 560 035

November – 2024

ACKNOWLEDGEMENT

The satisfaction that accompanies successful completion of any task would be incomplete without mention of people who made it possible, and who's constant encouragement and guidance have been a source of inspiration throughout the course of this project work. We offer our sincere pranas at the lotus feet of “AMMA”, MATA AMRITANANDAMAYI DEVI who showered her blessing upon us throughout the course of this project work.

We owe my gratitude to Prof. Manoj P., Director, Amrita School of Engineering, Bangalore. We thank Dr. Gopalakrishnan E. A., Principal, Amrita School of Computing, Bangalore for his support and inspiration.

It is a great pleasure to express our gratitude and indebtedness to our project guide Ms. Sreebha Bhaskaran, Assistant Professor (Sr.Gr.), Department of Computer Science and Engineering, Amrita School of Computing, Bangalore for her valuable guidance.

We would like to thank you and express our gratitude to all faculty members for their academic support. Finally, we are forever grateful to our parents, who have loved, supported, and encouraged us in all our endeavours.

BL.EN.U4CSE22155 Sriharsha vvs

BL.EN.U4CSE22143 Pranav Gokhale

BL.EN.U4CSE22169 Pranav Biju Nair

BL.EN.U4CSE22162 Viraj Kisan Daule

I. INTRODUCTION

The Pathfinding Visualizer is an innovative and interactive tool designed to provide users with a visual understanding of how different pathfinding algorithms operate in real-world scenarios. By allowing users to explore routing through algorithms such as A*, Dijkstra's, Greedy Best-First Search (BFS), and Bi-Directional Search, D* lite, Fringe Search , the visualizer serves as an educational and practical platform for comparing and understanding the efficiency of these techniques. The project emphasizes realistic applications by simulating navigation within urban landscapes, offering users insights into the performance and optimization of pathfinding algorithms under various geographic and infrastructural conditions.

The primary objective of this project is to facilitate a more intricate and practical comparison of pathfinding algorithms in real-life scenarios. The visualizer enables users to select specific locations and calculate routes between two points, typically within a city or urban setting. To maintain realism, a threshold distance is implemented, ensuring that only feasible, real-world scenarios are simulated. This approach allows for localized exploration of algorithm performance, giving users the ability to assess efficiency within chosen environments and geographic scales, ranging from city-level routes to regional networks.

In particular, the visualizer has been adapted to Bengaluru's map for a localized and detailed analysis. By leveraging the complex urban network of the city, the project examines how advanced algorithms can navigate through intricate road systems and varying geographic conditions. For instance, algorithms like Greedy A*, Dijkstra's, and Depth-First Search (DFS) are employed to identify and compare multiple paths from a source to a destination. The visualizer not only highlights the most efficient path but also provides performance metrics for each algorithm, enabling users to make informed decisions about the best route for their specific needs.

Overall, this project combines the power of visualization and algorithmic computation to explore the practical applications of pathfinding in diverse urban settings. By testing algorithms under realistic conditions and benchmarking them

in specific locations, such as Bengaluru, the visualizer aspire s to bridge the gap between theoretical pathfinding techniques and their real-world usability. This approach not only enhances our understanding of algorithmic performance but also contributes to the development of adaptable and efficient navigation systems for urban environments worldwide.

A. MOTIVATION

Efficient navigation in urban environments is a really difficult task and indeed challenging in terms of functionalities- logistics, urban planning, emergency services-and most navigation systems fail to provide solution for multiple geographic cases. This motivates our project to understand, compare, and optimize the real-world pathfinding algorithms.

This project provides an interactive tool for localized exploration of path-finding solutions by developing a visualizer on a real-world map, such as that for Bengaluru. Users can see how these algorithms- A*, Dijkstra, and Greedy BFS - fare with these city layouts and understand the relative strengths and weaknesses in context.

This project bridges the gap between theoretical studies and practical applications by allowing the comparison of algorithms in specific urban settings. It emphasizes how those algorithms perform in different scenarios in order to determine the most efficient paths. This, in turn, contributes to developing smarter, more responsive navigation solutions for real-world use cases which truly benefit different kinds of industries and applications.

B. PROBLEM STATEMENT

This project focuses on the application and optimization of advanced pathfinding algorithms, including A*, Dijkstra's, Greedy Best-First Search, Bi-Directional Search, D* lite and Fringe Search to enable efficient navigation within predefined urban networks. By using real-world maps and allowing users to select specific starting and ending points, the project aims to identify and present the shortest and most efficient routes between two locations. The algorithms are tested under realistic geographic conditions to evaluate their performance in terms of speed, accuracy, and suitability for different scenarios.

This project operates and focuses solely on the static structure of the chosen urban map. This allows for a pure evaluation of the algorithms' capabilities in handling fixed-path scenarios, offering insights into their strengths and limitations in real-world environments.
The project emphasises creating an intuitive and interactive visualizer that showcases how each algorithm functions in determining paths through an urban setting. By conducting rigorous testing across various locations and scales, the visualizer helps highlight the efficiency and performance metrics of the algorithms, providing a clear and comparative understanding of their behaviour. This systematic approach ensures the development of a navigation system that is both effective and adaptable across diverse urban settings.

C. CONTRIBUTION IN THE PAPER

This project introduces an innovative and practical approach to evaluating and visualizing pathfinding algorithms, combining theoretical analysis with real-world application. This project does not apply abstract or simulated scenarios for the solutions but puts multiple algorithms such as A*, Dijkstra's, Greedy BFS, Bi-Directional Search, D* lite and Fringe Search on real-world maps for practical comparison purposes. This is possible by allowing the user the selection of specific locations within a city and visualization of results on routing, hence providing an intuitive and interactive way of understanding strengths and limitations of different algorithms.

The key contributions of the project include a more detailed comparative analysis of algorithms in terms of metrics involving computational efficiency, path length, and adaptability to urban constraints. This analysis is enhanced by an interactive interface where users can dynamically choose origin and destination points. To that extent, a user can view multiple routes and even identify the most efficient path. Additionally, performance metrics are determined in real time, which aids users in making data-driven decisions on which route would best optimize the path.

The project bridges the gap between theoretical design of algorithms and practicability in urban environments. It points out the trade-offs among different algorithms, thus displaying their applications in intelligent

transportation systems, logistics, and emergency navigation. In addition, systematically analysing the applicability of these algorithms in diverse geographic and urban conditions offers a strong foundation for further research in real-world pathfinding and navigation systems.

D. ARRANGEMENT OF THE PAPER

1. Introduction
   - Introduction
   - Motivation
   - Problem Statement
   - Contribution in this Paper
   - Arrangement of the Paper
2. Literature Survey
3. System Model and Problem Formulation
   - Mathematical Model
   - Algorithm/Design/Architecture Diagram
4. Performance Evaluation
   - System/Tool Specifications
   - Simulation Setup/Implementation Details
   - Results
5. Conclusion
   - Conclusion
   - Future Scope
6. References

II. LITERATURE SURVEY

The paper [1] discusses the usage and optimization of pathfinding algorithms during game development, particularly in relation to enhancing the popular and highly utilized A* algorithm for increased efficiency and effectiveness. As online games continue to become more important in the conception of network entertainment, efficient pathfinding for game characters has come to be an important research area.

The research categorizes common pathfinding approaches, the blind algorithms, and their representative: DFS, BFS, and iterative deepening. Heuristic algorithms, such as A*, IDA*, also happen to be representative among pathfinding approaches. While blind algorithms search globally without any specific direction, heuristic algorithms use a goal-oriented strategy for better efficiency. For example, DFS gives more importance to exploring depth, leading to longer execution times on average. Conversely, BFS expands by searching nearby nodes first, offering faster results but demanding higher memory. Iterative deepening improves DFS by introducing depth limits, whereas IDA* integrates evaluation functions to predict path existence, enhancing search optimization at the cost of potential evaluation challenges.

It is one of the most known heuristic methods in artificial intelligence, designated for pathfinding purposes. Its main advantage is that it exploits heuristic functions to guide the search towards the target. This paper is a continuation of other research studies that incorporated dynamic programming or machine learning into A* improvements. The two methods treated apply to multi-agent systems in cases of complexity regarding the environment through which an agent must navigate.

It proposes an enhanced A* algorithm with fewer search nodes and optimized heuristic function calculations, thus shortening times in pathfinding. Experimental results reflect high performance relative to the standard A* algorithm, especially in a variety of game map cases. By reducing computation times and ensuring reliable pathfinding, this optimized algorithm improves gameplay, as it enhances playability and responsiveness. In conclusion, the relevance of optimizing the A* algorithm for game development purposes is still up to date with possible further improvements introduced through innovative techniques.

This paper [2] presents a method for finding optimal movement routes for military troops with the aid of terrain passability maps-a critical component of the Intelligence Preparation of the Battlefield (IPB) process. This paper discusses the need for an analysis of geographic environments to determine terrain passability for vehicle types, which serves as the foundation for troop movement planning and essentially identifies all NO GO and SLOW GO terrain as areas to be avoided. Thus, the proposed methodology eases the efficient routing process that meets the varied needs of operation by automating passability map generation.

The research methodology addresses the Military Unit Path Finding Problem (MUPFP), which consists of finding a secure and cost-effective path for military units in such a manner that costs are reduced, and threats and obstacles are avoided. The study authors enlarged their previous work by involving 3-class passability maps and proprietary software on automated route generation. All the generated routes are classified into three types: easy routes that are longer but avoid poor pass ability areas, difficult routes that are shorter but traverse difficult terrain, and standard routes that balance the extremes. This allows commanding officers to choose routes based on mission requirements for tactical purposes.

The study shows the application of the methodology in the test area along the Polish coast of the Baltic Sea. There, routes were created within a 500-m wide corridor using Level 2 Vector Map data. An automatic system allows the possibility of creating complex graphs with noticeably more nodes and edges than graph structures present in typical, road-based graphs, which allows more comprehensive analysis regarding cross-country movement. For example, the testing arena consisted of more than 343,000 elements in graphs based on passability maps, whilst its road-based counterpart contained 31,000 elements; that suggests that the system can support large-scale datasets.

The proposed system has a main advantage by generating routes near real-time, computations taking only a few seconds. Such speed is essential considering that IPB processes are dynamic and time-sensitive decisions in practice; very often changes in battlefield conditions occur in a short span of time. The study thus points out the requirement for automation of as much of the IPB process as possible to enhance operational efficiency.

The methodology is thereby verified, proving that the system could successfully generate many route options, for easy paths through challenging paths, while maintaining a good standard for balanced compromise. It will prove vital in military operations because it encompasses choices that would be tailored for particular scenarios to be presented to decision-makers. Considering also its implications for pathfinding in the urban setting and for robotic navigation, the potential value of such methodologies shines through.

The paper [3] studies the Multi-Agent Pathfinding problem in topological maps, particularly constrained to narrow corridors, through which multiple robots have

to pass without collisions. Traditional algorithms for solving the MAPF problem are designed to find optimal entry orders for robots moving through a corridor such that their paths do not cross each other. However, such methods still may pose safety risks when multiple robots try to pass a corridor simultaneously, especially in confined areas. To avoid these safety issues the authors propose a new solution that will prevent more than two robots to pass through the same corridor at any given time by assigning a spatial constraint on each of the corridor's entrance. This means that, at any given time, it is always one robot occupying the corridor without chances of accidents.

The background of this research is in the booming logistics industry, where burgeoning e-commerce has amplified the need for round-the-clock operation autonomous robots. Such technologies as artificial intelligence, computer vision, and robotics are being increasingly applied in automated warehousing to make the systems more efficient in terms of working operations. In such a scenario, MAPF is critical for providing a solution on the real-time coordination of multiple robots towards their target locations without collision in large-scale warehouse-based environments. Most of the studies on the MAPF have focused primarily on grid-based maps, appropriate for small AGVs, but in large robots like AGFs, stable navigation on topological maps is required.

In topological maps, robots move along predefined lanes connecting waypoints for large robots. This provides increased stability but brings pathfinding challenges. The reason is that there exist frequent corridor conflicts; two robots attempt to enter the same corridor from opposite directions. The authors have therefore proposed the Priority-Based Search algorithm, that computes paths in sequence according to priorities of robots, thereby avoiding conflict due to corridors and increasing solution efficiency. Extending further, the MAPF method introduces corridor occupancy-based prevention of more than two robots from occupying the same corridor at the same time. Such a method will enforce safe and efficient multi-agent pathfinding in topological maps through assignment of constraints that allow just one robot to enter into a corridor at any time.

The proposed corridor occupancy-based MAPF method improves the PBS algorithm by introducing further constraints, in particular regarding corridor entry control, so that each corridor is occupied by one robot only at any time. This greatly improved robustness of the safety level of their navigation for narrow corridors as well as a large number of agents. The findings of the study show how this approach can adequately design multiple robots to navigate collision-free

paths within topological maps, thus improving the safety and efficiency of autonomous navigation in the complex logistics environment.

The focus of the study [4] is on optimizing disaster response in terms of navigating routes by enhancing route navigation for emergency volunteers through the pathfinding algorithms integrated with Augmented Reality (AR). As Indonesia is extremely vulnerable to natural disasters, including floods, earthquakes, and landslides, the country experiences a massive rise in disasters yearly; in 2020 alone, there were 2,952 reported disasters with more than 6.2 million victims. Though the number of disasters is increasing, and the engagement of volunteers is also incrementing, reaching the affected locations does not seem smooth due to the constraints existing in the traditional 2D map, especially in rural or inaccessible areas. Volunteers generally face delays and navigation issues, implying more effective route visualization and planning.

These challenges require the exploration of the potential application of two pathfinding algorithms, Sparse A* and Dijkstra, combined with AR technology. The heuristic approach by which Sparse A* efficiently navigates large obstacles in terrains and reduces travel time by multi-layer and multi-heuristic adaptations is known. Dijkstra's algorithm also calculates gradients iteratively to identify reliable solutions for shortest paths. The algorithms are supported by the Haversine formula, which provides for accurate distance measures between geographical points given by their latitude and longitude; they're integrated to try and make route accuracy increase and time wastes decrease during disaster response.

The study here shows that the inclusion of AR with these pathfinding algorithms forms a solution that transforms route visualization. AR offers the means to have visual objects in real-time on devices, such as smartphones, tablets, and smart glasses, which overlay navigation paths directly onto the physical surroundings. It thus helps volunteers to efficiently locate disaster sites even during nighttime or poorly lit areas. Previous research reveals the feasibility of AR in outdoor navigation and its integration with GPS for the real-time location-based tracking system.

Experimental results had shown that Sparse A* was in much faster and more efficient speed than Dijkstra. Simulations and outdoor tests on a 2D map defined by latitude and longitude further revealed that Sparse A* always detected the endpoints sooner compared to Dijkstra. For example, Sparse A* showed dramatic

fewer node evaluations such as 364, 10,326, and 4,935, respectively, while Dijkstra needed more: 1,753, 41,185, and 22,936. Combining these distances with the Haversine formula, the obtained results are compared with Google Maps, proving this algorithm correct.

AR integration also enhanced navigation by visually tagging routes on mobile devices. AR-based tagging helped volunteers successfully track paths from beginning to end points, while maximising their effectiveness during disaster responses. At night, this was even more useful, when visibility is low. Using location-based AR, volunteers were intuitively guided and made fewer mistakes, thereby responding quickly to disasters.

In conclusion, the study emphasises that the hybridization of pathfinding algorithms with AR enhances navigation in disaster response. Sparse A* and Dijkstra's algorithms, along with the Haversine formula, provide for robust capabilities in pathfinding, whereas AR enhances visualisation and usability. This integrated approach ensures faster and more accurate route planning as well as equips volunteers with the tools they need to navigate complex environments and save more time and lives in emergency scenarios.

The study [5] evaluates the performance of two widely applied pathfinding algorithms, A* and RRT, in solving two-dimensional maze problems. An analysis of such strengths and weaknesses by the two respective algorithms in terms of accuracy, efficiency, and exploratory ability will be used in this research. To carry out the experiments, MATLAB software was utilised to simulate a variety of maze environments; the complexity of the mazes in each environment was defined by parameters including number of branches and the thickness of the walls. The experimental conditions for two distinct groups were created, which ranged from quite simple to quite complex mazes for the comparison of the performance of the algorithms across different levels of complexity.

On the efficiency front, A* outperforms RRT. More efficient applications requiring accurate navigation are aided by A*, as it produces paths with shorter lengths and fewer turns more rapidly by choosing nodes based on the shortest distance from the start to the goal. On the other hand, RRT randomly generates nodes due to which the path generated is longer and takes more time to traverse through the maze. However, this random nature helps in increasing the exploratory ability. It can cover large areas in an unstructured environment.

The exploring abilities of both algorithms were tested by varying the complexity of the maze. In simpler mazes, RRT performed relatively well since it was able to explore the environment even if it generated fewer nodes compared to A*. Since RRT is random, it allows the generated path tree to extend to farther areas, which increases its coverage to broader regions. For the complex mazes, A* showed a better exploration ability, covering nearly half of the area with a higher branch count. This shows that A* can be effectively traversed and used to explore even in more intricate and confined spaces, which is particularly useful in scenarios such as rescue missions in complex terrains.

Comparing the two algorithms, A* proved to be more accurate as well as efficient and thus the best suited for emergency transport in tight and narrow environments, where time and accuracy are of paramount importance. The RRT, although less efficient, is better suited for exploratory tasks, especially large open and unstructured environments. Moreover, it is stronger in generating random paths, meaning broad exploration in any unknown region, thus useful for environmental investigations or cases where possibly unknown regions have to be mapped.

While the study shed insightful light on the exploration and distances realized, there are some limitations to the findings. Distance calculations between nodes in RRT do not necessarily align with those in A*. Perhaps the nature of this discrepancy has an impact on the reliability of the exploration area measured in terms of the number of nodes. Further study is needed to more rigorously assess the exploration capabilities of each algorithm, this time within environments of considerably more complex constraints.

The strengths of A* in efficiency and precision, especially in narrow, complex environments, have been emphasised. The study, however, highlighted the strength of RRT particularly in exploration tasks and, thus, is suitable for broad environmental surveys. On the other hand, the two algorithms can be easily applied to various terrains depending on the specific requirements of the task at hand.

The paper [6] introduces a new graph partitioning technique, namely the Regions Discovery Algorithm (RDA), which aims to enhance the speed of pathfinding based on the maintenance of optimality in grid-based environments. In fact, pathfinding problems are usually constrained by limited processing time,

memory, and computational power as the size and complexity of the search space grow. To this end, RDA downsizes the space of search by gathering regions based on local features while managing to reduce the space of search up to 47%. As a result, execution time is faster and consumes less memory without removing potential solution paths.

Because of its efficiency and optimality, the A* algorithm is very popular in pathfinding. However, for large and complex environments, A* performance bottlenecks often arise as a result of underpowered computational equipment. Hierarchical pathfinding techniques can alleviate this problem with HPA* and HNA*. These techniques may even provide suboptimal paths as the environment starts to become more complex. Unlike traditional methods, which just divide the graph evenly, RDA improves on this by ignoring irrelevant regions to attain preserved optimal solutions while decreasing the complexity in search.

The algorithm is characterized by the ease and effectiveness of its simplicity, and it does not depend on precomputed paths or adjustments in the graph representation, so the algorithms are domain-independent and flexible in dynamic settings. By applying lookup tables in regions that will not contribute to any solution paths, RDA reduces search space and reduces computational effort while ensuring optimal solutions. This makes it a promising enhancement to A* techniques of finding appropriate solutions that do not compromise on the quality of the solution while improving search performance.

The study [7] focuses on pathfinding in car racing simulation games where paths require Non-Playable Characters (NPCs) to navigate tracks and avoid obstacles. Here, the combination of A* Algorithm with a Dynamic Pathfinding Algorithm (DPA) is proposed to further improve the NPC's ability to achieve shortest path as well as getting around both static and dynamic obstacles. The results obtained from the experiments are given below. It shows that combining DPA with A* generates better results compared to A* alone, especially when the barriers are dynamic, but it is effective only under specific conditions.

In simulation games, such as car racing, NPCs are implemented to either assist or compete with human players. In racing games, for example, the NPCs aim at crossing the finish line first by performing efficient pathfinding and avoiding obstacles. Extensive studies have already proven that pathfinding using the A* algorithm is efficient, especially in dynamic obstacle scenarios. But the problems,

such as non-smooth paths and computational load, have inspired several improvements, which include A* variants that reduce the waypoints or adjust for forward-direction searches.

Based on the previous work, this paper discusses more utilization of A* with DPA to make more efficient pathfinding for game NPCs in car racing games. By combining both algorithms, NPCs will navigate the empty track and obstacles better. The results show that the NPCs that use the combined approach outperform when using A* alone and are mostly effective when obstacles are dynamic in tracks. The method of grid representation also plays a prominent role in the A* pathfinding performance.

This paper [8] focuses on pathfinding in car racing simulation games. Non-Playable Characters (NPCs) navigate through tracks, avoiding obstacles to achieve their goal: reaching the finish line. By developing the Dynamic Pathfinding Algorithm (DPA) combined with the A* Algorithm, this paper demonstrates that the NPCs who use the two algorithms together could perform better than the ones which rely only on DPA. This combination makes the NPCs to better navigate tracks with static and dynamic obstacles. The position of obstacles and the shape of the racing trajectory substantially influence the performance of the DPA.

Previous works on artificial intelligence for racing games used A* for pathfinding and obstacle avoidance. The A* algorithm can navigate through dynamic and multiple obstacles on tracks. Research also presented some variants on A* to decrease the waypoints and computational load. Several studies also used the DPA to solve dynamic obstacle evasion. These improvements were on the competence of an NPC to determine the shortest way while navigating around moving barriers with efficiency.

More experiments show that if the control of NPCs is implemented either by humans or artificial intelligence, they have to perform efficient pathfinding in order to cross the finish line. The combination of A* and DPA gave NPCs better performance especially over tracks with obstacles. From the research, one can tell the conclusions as results in a more effective navigation system whose representation was crucially dependent on the grid representation and the positioning of the obstacle.

The study [9] describe a very efficient hierarchical pathfinding algorithm, Floyd-A* (FA*), designed specifically for mobile service robots that could work in indoor environments. The scheme makes use of key point extraction and region partitioning based on characteristics of spaces indoors to allow for better representation of abstract maps for faster pathfinding. This solves unique problems robots encounter when having to navigate their way through complex indoor environments.

The offline search phase in the FA* algorithm makes use of the Floyd-Warshall algorithm to gather useful prior knowledge that enhances the efficiency of subsequent query operations. It replaces the need for real-time online searches on abstract maps that are used to streamline the path-finding process. Additionally, the matrix memory mechanism is incorporated to decrease memory overhead and expand the capacity to store abstract paths, further optimizing the overall system performance.

Experimental evaluations and benchmark tests show that FA* accelerates traditional A by one to two orders of magnitude, while improvements in efficiency are considerable. In speed and memory utilization, FA* also outperforms other A* variants on grid maps. Such findings could prove the feasibility of using the FA* algorithm in practical applications, especially for indoor navigation of mobile service robots.

The paper [10] explores pathfinding in mobile games, a category that includes puzzles and card games where finding the fastest and shortest routes is crucial. For these applications, the A* (A Star) and Basic Theta* algorithms are frequently employed. The A* algorithm, originally introduced by Peter Hart, uses a heuristic function to assess costs and prioritize nodes based on their distance from the goal. This approach ensures that paths are both complete and optimal, making it a popular choice for grid-based pathfinding problems. However, the Basic Theta* algorithm, developed by Alex Nash in 2007, offers enhancements over the traditional A* by allowing more flexible routing that is not strictly confined to grid patterns. This flexibility enables Basic Theta* to adapt better to environments where grid constraints do not perfectly align with the actual layout of the terrain.

In pathfinding games, especially those on platforms built on Android, performance is measured with regard to three criteria: completeness, time complexity, and optimality. Completeness checks if the existence of a solution is

retrievable using the algorithm, while time complexity measures the efficiency of the algorithm in terms of the resources being used. Optimality deals with the quality of the path obtained with respect to distance or cost. Simulations for square grid maps with pre-specified start and goal points, as well as unwalkable points that are marked on the map, were done to compare the A* and Basic Theta* algorithms in a controlled environment. The results clearly show that both algorithms satisfy the completeness requirement and have similar time complexity. However, A* is more optimal when less nodes are involved because it systematically explores nodes to come up with the most optimal path. A Basic Theta* will likely yield the shortest path results, especially in those cases where a less strict routing is advantageous.

The strength and weakness of A* and Basic Theta* are pitted against each other because every application will possess different requirements for the pathfinding game. For applications in which route optimality matters, and computational resources are not a limiting factor, the systematic approach of A* is more suitable than both. On the other hand, for games where the shortest path is favored and flexibility in routing is considered important, Basic Theta* proves to be more effective. In summary, which algorithm to choose should ideally depend on the specific needs of the game as well as on the trade-off between computational efficiency and quality of the path.

The paper [11] develops an analysis of pathfinding algorithms, especially in mobile games that are into puzzles and card games, where the most efficient search technique will be required in finding the fastest and shortest routes. Two algorithms, A* (A Star) and Basic Theta*, were tested for their effectiveness. The A* algorithm, developed by Peter Hart, is a heuristic-based method of calculating the cost at every node and prioritizing nodes according to their distances from the goal. A* is optimal for grid-based environments but can be limited by its use of a grid pattern. On the other hand, Basic Theta* algorithm, developed by Alex Nash in 2007, creates A* with more flexibility in routing not constrained to grid paths which requires smoother and more direct paths, particularly in dynamic environments.

The discussion evaluates the performance of both algorithms on square grid maps, used in Android-based pathfinding games, with criteria such as completeness, time complexity, and optimality. Completeness deals with whether or not the algorithm will find a path, should one exist. It measures the time-

complexity of the algorithm. Its ability to evaluate the quality of the found path through calculating whether it is optimal or not, as shorter paths are preferable. According to simulation results, A* and Basic Theta* satisfy completeness criteria and provide similar time complexity. However, when there are fewer nodes involved, A* outperforms, as it prioritizes nodes more efficiently, leading to less nodes to be explored. Meanwhile, Basic Theta* provides the shortest path when solving non-grid-based routing advantages.

The findings of this study indicate that A* is preferable to Basic Theta* only in scenarios where specific requirements best fit the game. A* is more practical where the desire for optimality and efficiency in the nodes explored is of great importance, but in games where the shortest path would be of utmost priority with dynamic obstacles on the ground, Basic Theta* is more effective. The study concludes that developers must choose the pathfinding algorithm relative to their game's environment and performance requirements, all the while weighing the tradeoffs of optimality, path length, and computational efficiency.

The paper [12] discusses the challenge in strategy games and maze-solving in terms of pathfinding to avoid obstacles while arriving at the shortest path possible. Pathfinding is an essential component in games that require agents or characters to navigate through complex environments. The work has demonstrated the strength of a well-established pathfinding technique, A* algorithm, in real-time decision-making in dynamic and complex gaming environments. In particular, A* is known for its optimal solution where it seeks the shortest route efficiently even in large maps with a significant number of obstacles, which is why it's one of the most widely used real-time applications in games.

Experiments performed on map and maze images illustrated the ability of A* to look for a shortest path with an accuracy of over 85%. These experiments have shown how A* can 'find its way' through difficult environments that have obstacles preventing direct routes. The results of the study support the fact that although the algorithm is sound and applicable to simpler scenarios, it is also valid in more complex gaming maps, where obstacles density might decrease performance. A* makes a reliable choice for real-time pathfinding in games because it combines efficiency with the optimality of the paths.

It is extremely important for the development of realistic AI in games to understand pathfinding algorithms because they make the movement of agents through dynamic environments intelligent and responsive. When using A* for adaptive movement, the gameplay is more interesting as agents are able to react to changing conditions and recognize obstacles in real time. This means that the ultimate conclusion is that pathfinding algorithms, such as A*, while solving the technical challenges of navigation, also play a crucial role in making the overall gaming experience more realistic and interactive by depicting accurate AI behavior.

This research [13] presents the design and implementation of a shortest path algorithm tailored for labyrinth discovery in a multi-agent environment. In this scenario, robot agents begin with no prior knowledge of the maze and must learn as they explore. Each agent is responsible for solving a portion of the maze and updates a shared memory, allowing other agents to benefit from its discoveries. As an agent reaches the destination cell, it helps other agents connect their previously discovered paths to this endpoint. The proposed algorithm not only accounts for the coordinate distance but also considers factors such as the number of turns and moves required to traverse the path, making it more comprehensive than standard shortest path algorithms.

The performances are compared between the proposed shortest path algorithm and several other maze-solving methods such as Flood-Fill, Modified Flood-Fill, ALCKEF, as shown. Results demonstrate that the new algorithm outperforms the other traditional approaches due to the provision of a solution closer to the ideal one. In turn/move considerations, the paths created are more efficient, particularly in mazes whose complexity is such that simple distance measures are not applicable. Second, the new algorithm can be an application layer on top of previous algorithms in improving a subsequent runs and their practical applicability.

This multi-agent cooperative labyrinth discovery approach finds vast real world applications in Traffic Management Systems, Multi-Layer PCB Routing, Transporter Robots in Factories, Network Routing, and Trajectory Planning. Because this approach allows agents to share information amongst themselves and build upon the discoveries of their peers, it is particularly well-suited to environments in which cooperation and information exchange directly enhance efficiency. The proposed shortest path algorithm actually offers a robust solution

for cooperative tasks, and hence its versatility in various applications hints at potential future use in complex, real-time systems.

This paper [14] studies the use of robots for search and rescue operations in hazardous scenarios like collapsed buildings or earthquake ruins. It can be ideally modeled as a maze problem. Here, the concern is more about the efficiency in finding and saving people trapped, since these rescuers need to do it faster than anything else. This research compares the performance of three distinct algorithms for A*, differing in the heuristic functions used, to study their potential as maze searchers. The study also features the Depth-First Search (DFS) algorithm, as a benchmark to evaluate the efficiency of these heuristic-driven A* algorithms. These experiments show that the variants of A* algorithms, especially the Euclidean distance version within the heuristic function, significantly outperform DFS in most cases, with faster search results of higher accuracy.

There is further investigation into the effectiveness of different pathfinding algorithms. Included in this set are the popular Flood-Fill algorithm, which, based on the depth-first approach, is examined. Whereas the Flood-Fill algorithm performs well in other applications that use it for maze solving, it is not appropriate for search and rescue scenarios when time is an issue. The A* and D* algorithms are well-known path-finding techniques that have been extensively used in robotics for maze solving and navigation tasks. Based on previous work regarding robotic path planning, A* proved to be quite efficient as compared to the D* algorithm in terms of time efficiency and accuracy.

The experimental results of the research have shown that A* algorithms with different heuristic functions perform variably when searching through unknown mazes. On average, A* outperforms DFS which explores the maze blindly without heuristic guidance. Among tested variants, A*, where the second heuristic function incorporates Euclidean distance, gave the best result regarding the finding of a shortest path. This was followed by A using other heuristic approaches, and this was least effective with DFS, because it lacked heuristic information. The paper concludes that the A algorithm is largely well-suited for real-world search and rescue operations, and future work will be on optimizing the return route of the robots so they can navigate the maze effectively while considering the requirement to pick the best route when rescuing.

Motard et al. [15] proposes an online incremental vision-based topological map learning method for AIBO robots. The topological map is represented through a graph, where the vertices encode views from the robot's environment and the edges represent the spatial relationships between these views. The views are characterized by their SIFT keypoints. The proposed map learning method has been successfully applied to a homing application.

III.SYSTEM MODEL AND PROBLEM FORMULATION

A. MATHEMATICAL MODEL

A* Algorithm:

f(n) = g(n) + h(n)

Pseudocode:

Initialize:

- Open set with the start node.
- g-score of the start node = 0.
- f-score of the start node = h(start,goal)

While the open set is not empty:

- Pick the node with the lowest f-score from the open set as the current node.
- If the current node is the goal:
    - Reconstruct and return the path.
- Remove the current node from the open set.
- For each neighbour of the current node:
    - Calculate the tentative g-score.
    - If g-score is better:
        - Update the neighbour's g-score.
        - Update the neighbour's f-score: g-score + heuristic.

        - Add the neighbour to the open set (if not already present).

Return failure if the goal is not found.

Greedy Best-First Search

f(n) = h(n)

Pseudocode:

- Initialize:
    - Open set with the start node.
- While the open set is not empty:
    - Pick the node with the lowest heuristic h(current,goal) as the current node.
    - If the current node is the goal:
        - Reconstruct and return the path.
    - Remove the current node from the open set.
    - For each neighbour of the current node:
        - If the neighbour is not already in the open set:
            - Add it to the open set.
            - Record the current node as the predecessor.
- Return failure if the goal is not found.

Dijkstra's Algorithm

f(n) = g(n)

Pseudocode:

Initialize:

- Distance of all nodes = ∞, except the start node (0).
- Priority queue containing the start node.

While the priority queue is not empty:

- Remove the node with the smallest distance as the current node.

- For each neighbour of the current node:
    - Calculate the alternative distance: dist[current]+cost(current,neighbour).
    - If the alternative distance is smaller:
        - Update the neighbour's distance.
        - Add the neighbour to the priority queue.

Return the shortest distances to all nodes.

Bidirectional Search

$$g_{start}(n) + g_{goal}(n) = \mathbf{g(n)}$$

Pseudocode:

Initialize:

- Open set for both forward (from start) and backward (from goal) searches.

While both open sets are not empty:

- Expand one level in the forward search.
    - If any node in the forward search is in the backward search:
        - Return the path.
- Expand one level in the backward search.
    - If any node in the backward search is in the forward search:
        - Return the path.

Return failure if no path is found.

*D Lite**

g(n)=min(g(n),c(u,n)+g(u))

Pseudocode:

Initialize:

- Priority queue.
- g(goal)=, all other nodes = ∞.

While the priority queue is not empty:

- Remove the node with the lowest priority as the current node.
- For each neighbour:
    - Update the cost of the neighbour based on the current node.
    - Recalculate the priorities and update the queue.

Reconstruct and return the shortest path.

Fringe Search

f(n) = g(n) + h(n)

Pseudocode:

Initialize:

- Fringe list with the start node.
- Next fringe list as empty.

While the fringe is not empty:

- For each node in the fringe:
    - If the node is the goal:
        - Reconstruct and return the path.
    - For each neighbour of the node:
        - If the neighbour has not been visited:
            - Add it to the next fringe list.
- Replace the fringe with the next fringe.
- Reset the next fringe to empty.

Return failure if no path is found.

B. ALGORITHM / DESIGN / ARCHITECTURE DIAGRAM

i)Algorithms Used:

- **A* Algorithm**
  - This method finds the shortest path between two points by exploring paths with the smallest total estimated cost, combining the actual distance from the start with an estimate to the goal. It iteratively evaluates and updates paths until the destination is reached or all possibilities are exhausted.
  - Advantages:
    i. Efficient by focusing on promising paths.
    ii. Ensures the shortest path when criteria are met.
    iii. Works on various graph types.
  - Limitations:
    i. Performance drops with poor estimations.
    ii. Requires significant memory for large graphs.
  - Applications:
    GPS navigation systems, game development for AI pathfinding, robotics for real-world path planning.
- **Bidirectional Search**
  - This approach simultaneously searches from both the start and end points, expanding nodes in both directions until the paths intersect. It alternates between exploring each frontier and updates nodes based on connections, aiming to reduce the search space and converge efficiently.
  - Advantages:
    i. Faster than unidirectional search in many cases.
    ii. Reduces exploration space significantly.
    iii. Effective for symmetric or well-connected graphs.
  - Limitations:
    i. Requires more memory for dual-frontier storage.
    ii. Difficult to implement with inconsistent graph weights.
  - Applications:

Large-scale network routing, social network analysis, AI navigation in symmetrical spaces.

- **Dijkstra's Algorithm**
  - This algorithm finds the shortest path by exploring nodes in order of increasing cost from the start. It iteratively selects the nearest unexplored node, updates costs for its neighbours, and continues until the destination is reached or all nodes are explored.
  - Advantages:
    i. Guarantees the shortest path in weighted graphs.
    ii. Works well with positive edge weights.
    iii. Simple and widely applicable.
  - Limitations:
    i. Inefficient for large graphs without optimization (e.g., a priority queue).
    ii. Cannot handle negative edge weights.
  - Applications:
    Network routing optimization, scheduling and resource allocation, AI navigation for static environments.
- **Greedy Algorithm**
  - This method selects paths based solely on the estimated distance to the goal, prioritizing the most promising nodes without considering the full path cost. It iteratively explores nodes closest to the destination until the goal is reached or no paths remain.
  - Advantages:
    i. Fast for simple pathfinding tasks.
    ii. Focuses directly on the goal.
    iii. Easy to implement.
  - Limitations:
    i. Does not guarantee the shortest path.
    ii. Prone to getting stuck in suboptimal paths.
  - Applications:
    Game AI for quick decisions, heuristic-based search in static environments, approximate solutions in large-scale graphs.
- **D* Lite**
  - This algorithm efficiently handles dynamic environments by recalculating paths only when changes occur. It maintains a

priority queue of nodes and updates costs (g and rhs values) incrementally. It uses heuristic-based keys to explore nodes and supports replanning when obstacles or costs change in the graph.

- Advantages:
    i. Optimized for dynamic or changing environments.
    ii. Avoids recomputing the entire path after updates.
    iii. Guarantees shortest path.
- Limitations:
    i. Computationally expensive for dense or highly connected graphs.
    ii. Complexity increases with frequent environment updates.
- Applications:
    Robotics for navigation in uncertain environments, dynamic route planning in autonomous vehicles.

- **Fringe Search**
    - The Fringe Search algorithm is a memory-efficient alternative to A*. It expands nodes in a priority-based fringe list and calculates heuristic values for neighbors. Nodes are added to the fringe list only if they improve the path estimate, ensuring efficient exploration. Upon reaching the goal node, the path is reconstructed from
    - Advantages:
        i. Reduced memory usage compared to A*.
        ii. Handles large graphs efficiently.
        iii. Maintains optimal pathfinding.
    - Limitations:
        i. Slower in cases with uniform costs.
        ii. Requires fine-tuning of heuristics for specific use cases..
    - Applications:
        Pathfinding in grid-based games, route optimization in large maps, robotics navigation where memory is constrained.

ii)Architecture Diagram:

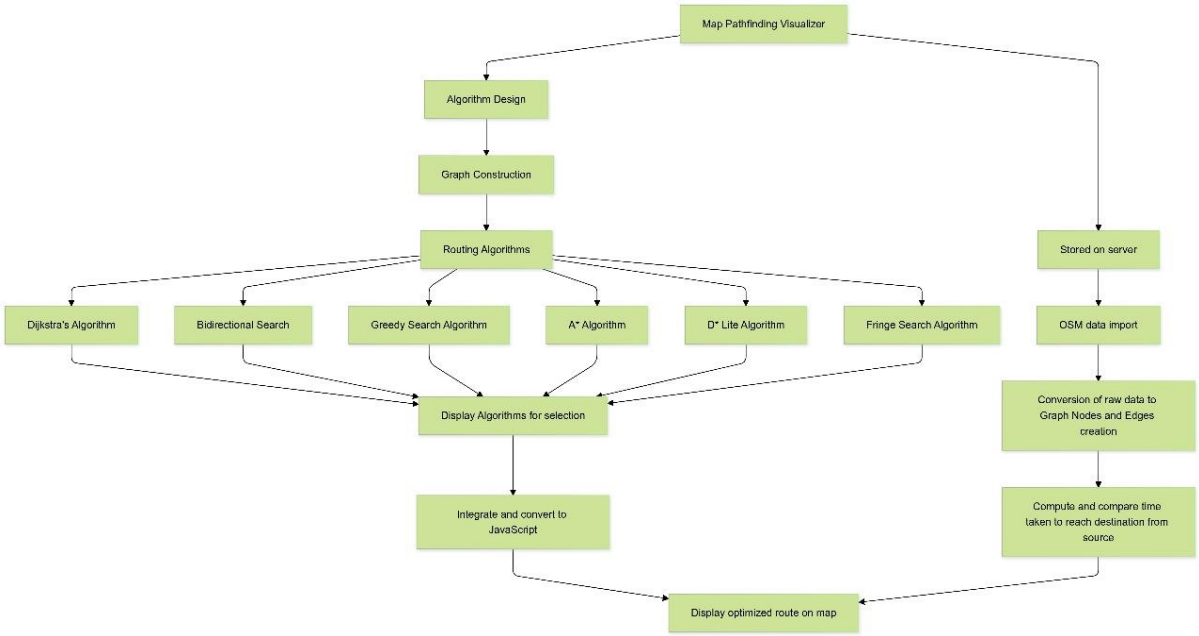


Fig. 1. Depicts the flow of the process carried out in the project. Map Pathfinding Visualizer is the software which is developed in this project. It contains algorithms including Dijkstra, Bidirectional, Greedy, A*, D* Lite, Fringe which are coded to find the distance between two data points which in this case is the distance between 2 addresses. The addresses are chosen from a map. This map, which includes details of all the nodes, edges, possible paths, junctions and endings, is taken from the OSM (Open Street Map) data.

The OSM data is stored on a server as and when a user wants to run the software. This is done to avoid high computational time and unnecessary memory utilisation , had it been OSM data to be downloaded on the pc. The OSM data in its original form can't be used for the software. Thereby, converting it from its raw form to a form where all the paths, junctions and nodes are clearly defined.

The software's frontend then comes into play to collect the input data from the user which are the source address point, destination address point and the preferred algorithm from the list of algorithms.

After these data are collected, it is sent to the algorithm chosen along with the converted OSM data. Code runs to find the optimum path between the addresses. The process of tracking paths is also displayed in the frontend.

As we run the algorithm, in parallel runs a timer to compute the time required to find the optimum path. If and when source and destination points get connected for the first time, the process of tracking paths is stopped and from the point where the connection was made, backtracking takes place and the final optimum path is highlighted.

At the end, with every input address points, and algorithms available, one can find the optimised route and the computational time displayed on screen helps to analyse many factors such as which algorithm takes least time, or highest time. Average computational time of all the algorithms and which all factors affect the time such as distance between the address points is also studied.

IV. PERFORMANCE EVALUATION

A. SYSTEM TOOL SPECIFICATIONS

1. Hardware Specifications:

- CPU: Intel Core i7 (8th Generation)
- RAM: 16GB DDR4
- Storage: 512GB SSD
- GPU: Integrated Intel UHD 620 (suitable for general computing tasks)

2. Software Specifications :

- Programming Language : NodeJS v22.11.0
- Libraries :
  - Array and Set for route lookups.
  - Map for GUI development.
  - Basic Math Libraries for plotting comparative analysis graphs.

SOFTWARE REQUIREMENTS:

To develop and implement the Maze Generator and Solver application, the following software requirements are necessary:

1. Programming Language:

• NodeJS v 22.11.0 for building of the map and back-end front-end integration.

2. Libraries and Modules:

• Lodash: Advanced array manipulations like sorting and filtering.

• Heap.js: For performing priority queue operations.

• D3.js and PixiJS: For visualization.

•Graphlib: Running Graph functions.

3. Integrated Development Environment (IDE):

• NodeJS

• VS Code

• JetBrains IntelliJ IDEA

• Sublime Text

4. Operating System:

• Windows 10 or later, Linux, or macOS: The application is cross-platform, provided Python is installed.

5. Visualization Tools (Optional):

• Tools like Excel or Tableau can be used for analyzing the performance metrics if exported to CSV files.

HARDWARE REQUIREMENTS:

To ensure smooth development and execution of the Map Pathfinding Visualizer application, the following hardware specifications are required:

1.Processor:

• Minimum: Intel Core i3 or equivalent

• Recommended: Intel Core i5/i7 or AMD Ryzen 5/7

2. Memory (RAM):

• Minimum: 4 GB

• Recommended: 8 GB or higher for better performance, especially when handling larger mazes.

3.Storage:

• Minimum: 500 MB of free space for NodeJS installation, libraries, and project files.

• Recommended: SSD for faster execution of code and reduced processing time.

4. Display:

• Minimum: Resolution of 1024x768 pixels.

• Recommended: Full HD (1920x1080 pixels) or higher for clear visualization of the GUI and graph layout.

5. Peripherals:

• Input: Keyboard and mouse for interaction with the GUI.

• Optional: Touchscreen (if applicable) for interactive GUI control.

6. Graphics Support:

• Basic integrated graphics support is sufficient as the project does not involve heavy graphical computation.

The software and hardware requirements specified above are essential for the successful implementation and execution of the Map Pathfinding Visualizer project. While the software requirements ensure the application runs with the necessary tools and libraries, the hardware requirements guarantee optimal performance during maze generation, solving, and analysis.

B. SIMULATION SETUP / IMPLEMENTATION DETAILS

The implementation phase focuses on the translation of the project design into functional code, ensuring that all the components are developed cohesively to achieve the desired objectives. This chapter provides an in-depth explanation of the implementation details, including algorithms, modules, and their integration into a complete system.

1. OSM Data: The data was imported from a server hosting complete information and use of the map data.

2. Graph and Node Construction: The front-end of the project is based on the map data collected from Open Street Maps. Using the map data, graphs and nodes were constructed for creating paths.

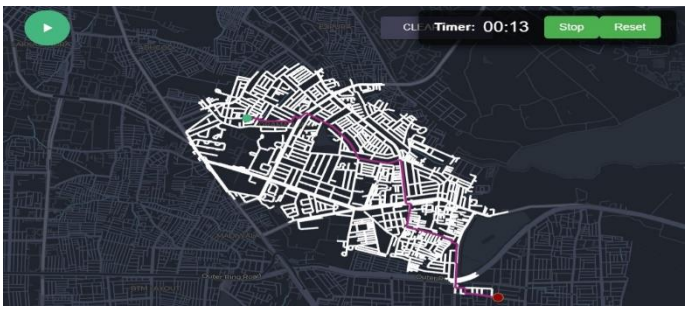
3.NodeJS and OSM data integration: The integration leads to the connection of the data that is the front end giving a GUI and the back end involves the 6 algorithm functions:
- A* Algorithm
- Bi-directional search Algorithm
- D* Lite Algorithm
- Dijkstra's Algorithm
- Fringe search Algorithm
- Greedy Algorithm

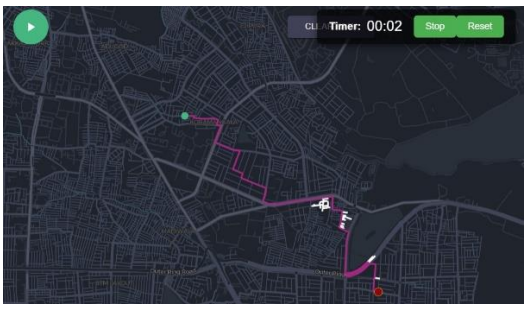
4.Performance Metrics: The performance metrics include computation of time in seconds, indicating the time taken by the algorithm to route from the source to the destination.

C. RESULTS

The Pathfinding Visualizer was tested multiple times on multiple factors such as short range distance and long range distance.
These are the following routes and the respective time taken to reach the destination from the source.

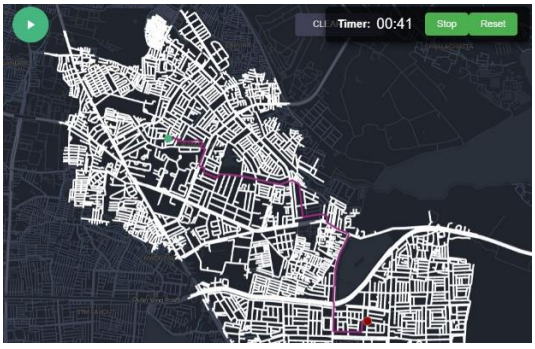
1. Long Range Distance: (~4 kilometers)





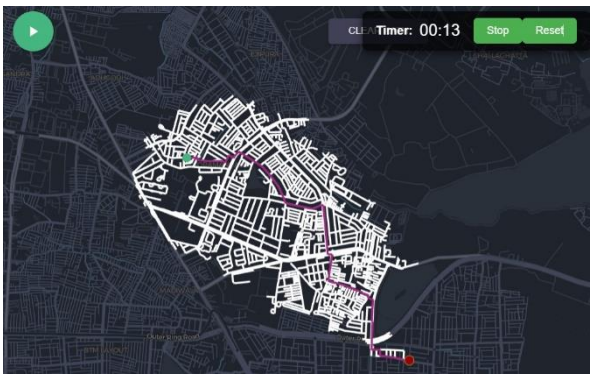
A* Algorithm

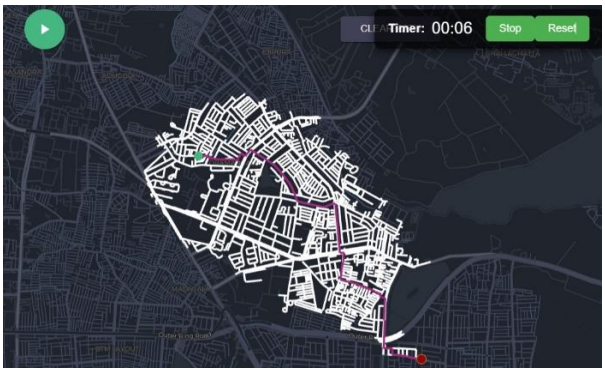
Greedy Algorithm



Dijkstra’s Algorithm

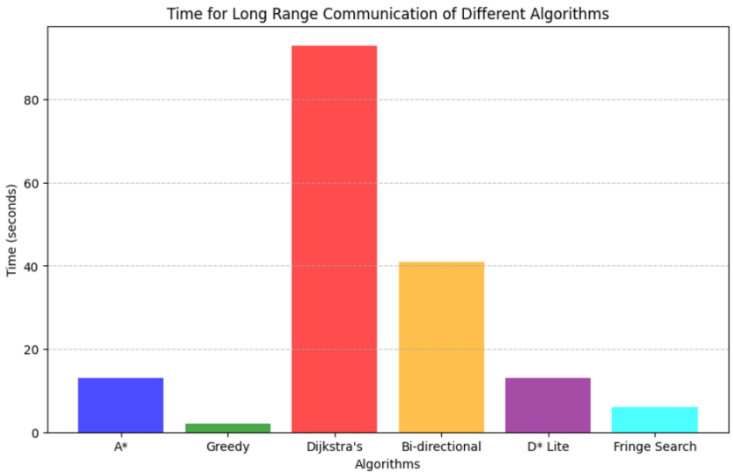
Bi-directional Search Algorithm

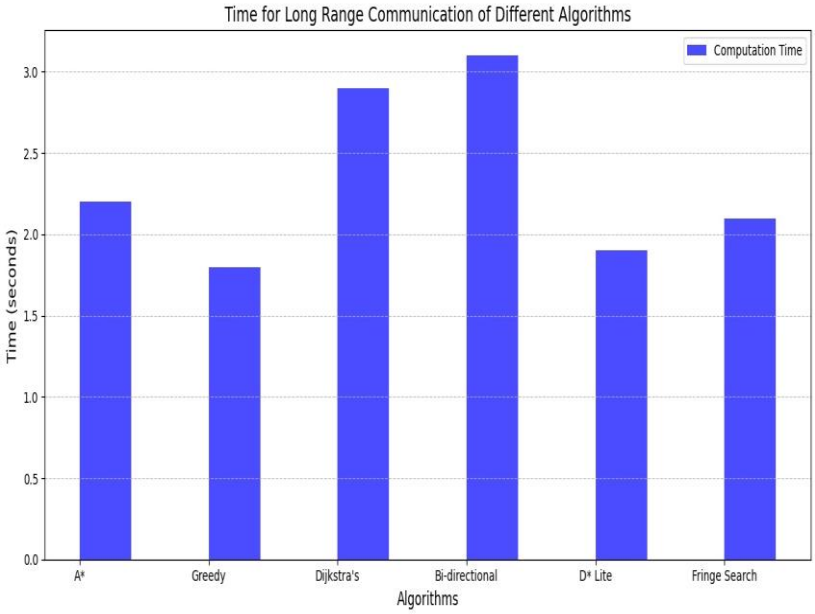




D* Lite Algorithm

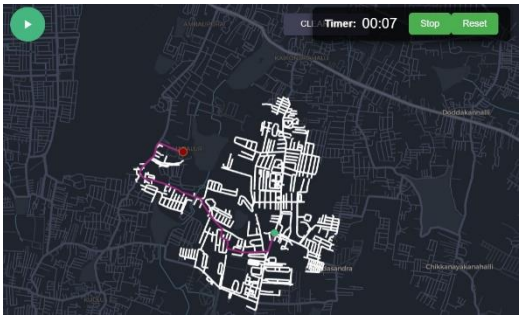
Fringe Search Algorithm



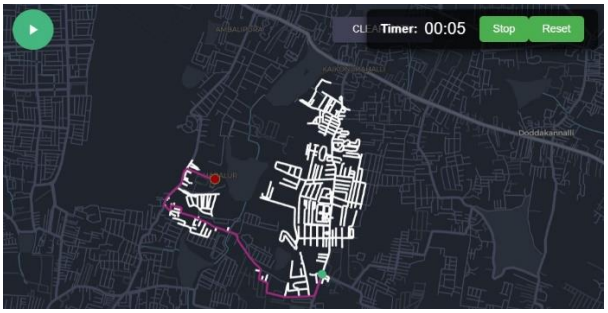


Based on the time periods observed for the following, graphs were plotted to analyze and conclude the best algorithm, and it turned out that Greedy takes the least time and Dijkstra's algorithm takes the most time for long range distances.

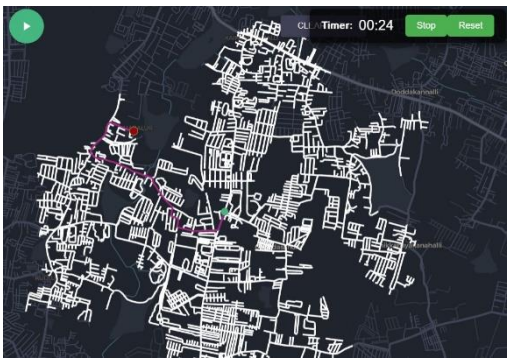
2. Short Range Distance: (~2 kilometers)



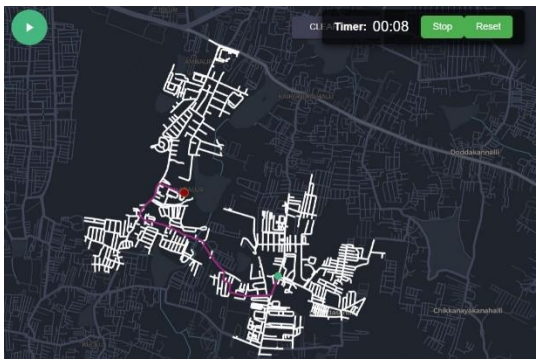
A* Algorithm



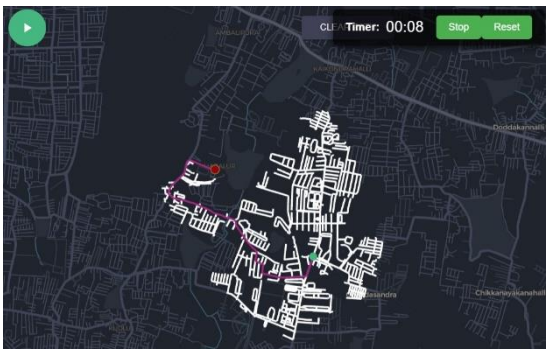
Greedy Algorithm



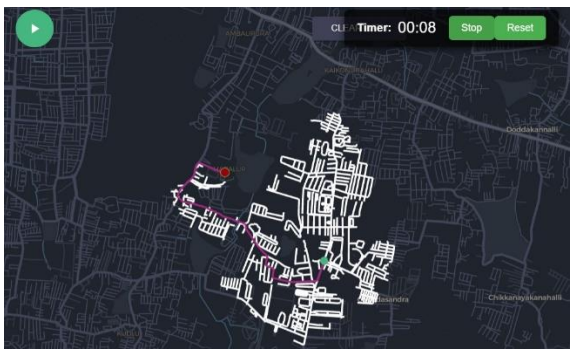
Dijkstra’s Algorithm



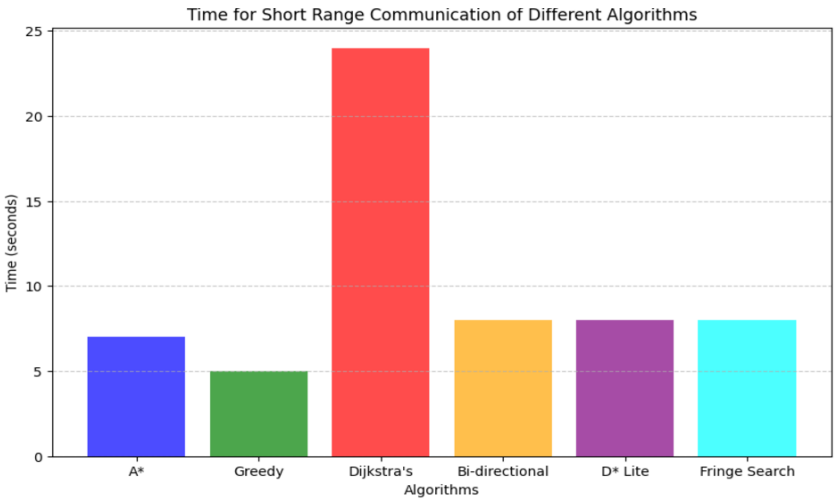
Bi-directional Search Algorithm

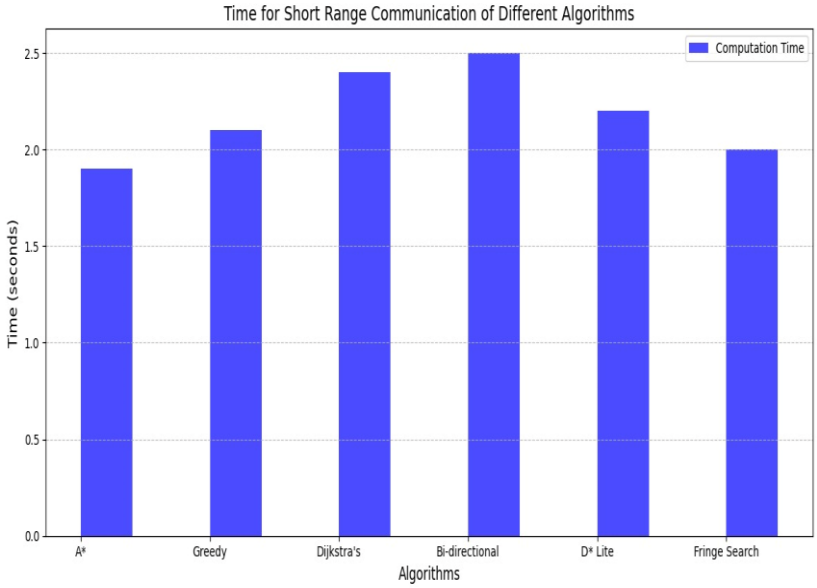


D* Lite Algorithm



Fringe Search Algorithm





Based on the time periods observed for the following, graphs were plotted to analyze and conclude the best algorithm, and it turned out that Greedy takes the least time and Dijkstra's algorithm takes the most time for short range distances.

V. CONCLUSION

For Long Distance Commute, it is evident that the Greedy algorithm is the best-performing algorithm, with the shortest execution time, close to 0 seconds. In contrast, Dijkstra's algorithm is the worst-performing, taking the longest time at over 80 seconds. This highlights a significant performance difference of approximately 80 seconds between the best and worst algorithms, emphasizing

the efficiency of Greedy in this specific context and the comparative inefficiency of Dijkstra's for long-range communication.

For Short Distance Commute, the Greedy algorithm once again demonstrates the best performance, with the shortest execution time of approximately 5 seconds. On the other hand, Dijkstra's algorithm is the worst-performing, taking the longest time of around 25 seconds. This reveals a significant performance gap of roughly 20 seconds between the best and worst algorithms, underscoring the Greedy algorithm's efficiency for short-range scenarios and Dijkstra's inefficiency in the same context.

VI. REFERENCES

[1] Yan Yang and Yanlian Du ,``Application of Improved A* Algorithm in Game Pathfinding,'' ,Dept. of Information and Communication Engineering, Haian University, China, April, 2024.

[2] Krzysztof Pokonieczny, Wojciech Dawid and Marek Wyszyński ,``Methodology of using pathfinding methods in military passability maps,'' ,Dept. of Civil Engineering and Geodesy, Military University of Technology, Poland, July, 2023.

[3] Soohwan Song, Ki-In Na and Wonpil Yu ,``Corridor Occupancy-Based Multi-Agent Pathfinding in Topological Maps,'' ,Dept. of Electronics and Telecommunication, Research Institute, Daejeon, Korea, November, 2022.

[4] Teddy Mantoro, Zaenal Alamsyah and Media Anugerah Ayu ,``Pathfinding for Disaster Emergency Route Using Sparse A* and Dijkstra Algorithm with Augmented Reality,'' ,Dept. of Computer Science Engineering, Sampoerna University and Nusa Putra University, Indonesia, January, 2022.

[5] Y. He, P. Wang and J. Zhang, "A Comparison Between A* and RRT in Maze Solving Problem," 2021 3rd International Symposium on Robotics and Intelligent

Manufacturing Technology (ISRIMT), Changzhou, China, 2021, pp. 333-338, doi: 10.1109/ISRIMT53730.2021.9596830.

[6] Ying Fung Yiu and Rabi Mahapatra ,``Regions Discovery Algorithm for Pathfinding in Grid Based Maps,'' ,Dept. of Computer Science Engineering, Texas A\&M University, USA, November, 2020.

[7] Yoppy Sazaki, Anggina Primanita and Muhammad Syahroyni ,``Pathfinding car racing game using dynamic pathfinding algorithm and algorithm A*,'' ,Faculty of Computer Science Engineering, Universitas Sriwijaya, Indonesia, February, 2018.

[8] Yoppy Sazaki, Hadipurnawan Satria and Muhammad Syahroyni ,``Comparison of A* and dynamic pathfinding algorithm with dynamic pathfinding algorithm for NPC on car racing game,'' ,Faculty of Computer Science Engineering, Universitas Sriwijaya, Indonesia, February, 2018.

[9] Xushen Chen and Shiyin Qin ,``Approach to high efficient hierarchical pathfinding of indoor mobile service robots based on grid map and Floyd-Warshall algorithm,'' ,School of Automation Science and Electrical Engineering, Beihang University, Beijing, China, January, 2018.

[10] Eka Risky Firmansyah, Siti Ummi Masruroh and Feri Fahrianto ,``Comparative Analysis of A* and Basic Theta* Algorithm in Android-Based Pathfinding Games,'' Syarif Hidayatullah State Islamic University, Indonesia, January, 2017.

[11] A. S. Hidayatullah, A. N. Jati and C. Setianingsih, "Realization of depth first search algorithm on line maze solver robot," 2017 International Conference on Control, Electronics, Renewable Energy and Communications (ICCREC), Yogyakarta, Indonesia, 2017, pp. 247-251, doi: 10.1109/ICCEREC.2017.8226690.

[12] "Pathfinding in Strategy Games and Maze Solving Using A* Search Algorithm" written by Nawaf Hazim Barnouti, Sinan Sameer Mahmood Al-Dabbagh, Mustafa Abdul Sahib Naser,published by Journal of Computer and Communications, Vol.4 No.11, 2016

[13] B. Rahnama, M. C. Özdemir, Y. Kiran and A. Elçi, "Design and Implementation of a Novel Weighted Shortest Path Algorithm for Maze Solving Robots," 2013 IEEE 37th Annual Computer Software and Applications Conference Workshops, Japan, 2013, pp. 328-332, doi: 10.1109/COMPSACW.2013.49

[14] Liu, Xiang, and Daoxiong Gong. "A comparative study of A-star algorithms for search and rescue in perfect maze." 2011 international conference on electric information and control engineering. IEEE, 2011.

[15] Elvina Motard, Bogdan Raducanu, Viviane Cadenat and Jordi Vitria ,``Incremental On-Line Topological Map Learning for A Visual Homing Application,'', University of Paul Sabatier, France, May, 2007.